



# Process Management Interface for Exascale (PMIx) Standard

**Version 2.0**

**September 2018**

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

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

Copyright © 2018 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?	3
1.2.2. What is defined in the standard?	3
1.2.3. What is <i>not</i> defined in the standard?	3
1.2.4. General Guidance for PMIx Users and Implementors	4
1.3. PMIx Architecture Overview	5
1.3.1. The PMIx Reference Implementation (PRI)	6
1.3.2. The PMIx Reference RunTime Environment (PRRTE)	7
1.4. Organization of this document	7
1.5. Version 1.0: June 12, 2015	8
1.6. Version 2.0: Sept. 2018	9
<b>2. PMIx Terms and Conventions</b>	<b>10</b>
2.1. Notational Conventions	11
2.2. Semantics	12
2.3. Naming Conventions	13
2.4. Procedure Conventions	13
2.5. Standard vs Reference Implementation	13
<b>3. Data Structures and Types</b>	<b>15</b>
3.1. Constants	15
3.1.1. Error Constants	16
3.2. Data Types	19
3.2.1. Key Structure	19
3.2.2. Namespace Structure	20
3.2.3. Rank Structure	21
3.2.4. Process Structure	21

3.2.5.	Process structure support macros . . . . .	21
3.2.6.	Process State Structure . . . . .	23
3.2.7.	Process Information Structure . . . . .	24
3.2.8.	Process Information Structure support macros . . . . .	25
3.2.9.	Scope of Put Data . . . . .	26
3.2.10.	Range of Published Data . . . . .	27
3.2.11.	Data Persistence Structure . . . . .	27
3.2.12.	Value Structure . . . . .	28
3.2.13.	Value structure support macros . . . . .	29
3.2.14.	Load a <code>pmix_value_t</code> structure . . . . .	30
3.2.15.	Info and Info Array Structures . . . . .	31
3.2.16.	Info structure support macros . . . . .	32
3.2.17.	Info Type Directives . . . . .	35
3.2.18.	Info Directive support macros . . . . .	35
3.2.19.	Job Allocation Directives . . . . .	36
3.2.20.	Lookup Returned Data Structure . . . . .	37
3.2.21.	Lookup data structure support macros . . . . .	37
3.2.22.	Application Structure . . . . .	40
3.2.23.	App structure support macros . . . . .	40
3.2.24.	Query Structure . . . . .	42
3.2.25.	Query structure support macros . . . . .	42
3.2.26.	Modex Structure . . . . .	43
3.2.27.	Modex data structure support macros . . . . .	44
3.3.	Data Packing/Unpacking Types and Structures . . . . .	45
3.3.1.	Byte Object Type . . . . .	45
3.3.2.	Byte object support macros . . . . .	45
3.3.3.	Data Buffer Type . . . . .	47
3.3.4.	Data buffer support macros . . . . .	47
3.3.5.	Data Array Structure . . . . .	49
3.3.6.	Generalized Data Types Used for Packing/Unpacking . . . . .	49
3.4.	Reserved attributes . . . . .	51
3.4.1.	Initialization attributes . . . . .	51
3.4.2.	Tool-related attributes . . . . .	52

3.4.3.	Identification attributes . . . . .	52
3.4.4.	UNIX socket rendezvous socket attributes . . . . .	53
3.4.5.	TCP connection attributes . . . . .	53
3.4.6.	Global Data Storage (GDS) attributes . . . . .	54
3.4.7.	General process-level attributes . . . . .	54
3.4.8.	Scratch directory attributes . . . . .	54
3.4.9.	Relative Rank Descriptive Attributes . . . . .	55
3.4.10.	Size information attributes . . . . .	56
3.4.11.	Memory information attributes . . . . .	56
3.4.12.	Topology information attributes . . . . .	57
3.4.13.	Request-related attributes . . . . .	57
3.4.14.	Server-to-PMIx library attributes . . . . .	58
3.4.15.	Server-to-Client attributes . . . . .	59
3.4.16.	Event handler registration and notification attributes . . . . .	59
3.4.17.	Fault tolerance attributes . . . . .	61
3.4.18.	Spawn attributes . . . . .	61
3.4.19.	Query attributes . . . . .	63
3.4.20.	Log attributes . . . . .	64
3.4.21.	Debugger attributes . . . . .	64
3.4.22.	Resource manager attributes . . . . .	65
3.4.23.	Environment variable attributes . . . . .	65
3.4.24.	Job Allocation attributes . . . . .	65
3.4.25.	Job control attributes . . . . .	66
3.4.26.	Monitoring attributes . . . . .	67
3.5.	Callback Functions . . . . .	67
3.5.1.	Release Callback Function . . . . .	68
3.5.2.	Modex Callback Function . . . . .	68
3.5.3.	Spawn Callback Function . . . . .	69
3.5.4.	Op Callback Function . . . . .	70
3.5.5.	Lookup Callback Function . . . . .	70
3.5.6.	Value Callback Function . . . . .	71
3.5.7.	Info Callback Function . . . . .	71
3.5.8.	Event Handler Registration Callback Function . . . . .	72

3.5.9.	Notification Handler Completion Callback Function . . . . .	73
3.5.10.	Notification Function . . . . .	74
3.5.11.	Server Setup Application Callback Function . . . . .	76
3.5.12.	Server Direct Modex Response Callback Function . . . . .	77
3.5.13.	<code>pmix_connection_cbfunc_t</code> . . . . .	78
3.5.14.	<code>pmix_tool_connection_cbfunc_t</code> . . . . .	79
3.5.15.	Constant String Functions . . . . .	79
<b>4.</b>	<b>Initialization and Finalization</b>	<b>82</b>
4.1.	Query . . . . .	82
4.1.1.	<code>PMIx_Initialized</code> . . . . .	82
4.1.2.	<code>PMIx_Get_version</code> . . . . .	83
4.2.	Client Initialization and Finalization . . . . .	83
4.2.1.	<code>PMIx_Init</code> . . . . .	83
4.2.2.	<code>PMIx_Finalize</code> . . . . .	86
4.3.	Tool Initialization and Finalization . . . . .	87
4.3.1.	<code>PMIx_tool_init</code> . . . . .	87
4.3.2.	<code>PMIx_tool_finalize</code> . . . . .	90
4.4.	Server Initialization and Finalization . . . . .	91
4.4.1.	<code>PMIx_server_init</code> . . . . .	91
4.4.2.	<code>PMIx_server_finalize</code> . . . . .	93
<b>5.</b>	<b>Key/Value Management</b>	<b>95</b>
5.1.	Setting and Accessing Key/Value Pairs . . . . .	95
5.1.1.	<code>PMIx_Put</code> . . . . .	95
5.1.2.	<code>PMIx_Get</code> . . . . .	96
5.1.3.	<code>PMIx_Get_nb</code> . . . . .	98
5.1.4.	<code>PMIx_Store_internal</code> . . . . .	100
5.2.	Exchanging Key/Value Pairs . . . . .	101
5.2.1.	<code>PMIx_Commit</code> . . . . .	101
5.2.2.	<code>PMIx_Fence</code> . . . . .	102
5.2.3.	<code>PMIx_Fence_nb</code> . . . . .	103
5.3.	Publish and Lookup Data . . . . .	105
5.3.1.	<code>PMIx_Publish</code> . . . . .	105

5.3.2.	<b>PMIx_Publish_nb</b>	107
5.3.3.	<b>PMIx_Lookup</b>	109
5.3.4.	<b>PMIx_Lookup_nb</b>	111
5.3.5.	<b>PMIx_Unpublish</b>	113
5.3.6.	<b>PMIx_Unpublish_nb</b>	114
<b>6.</b>	<b>Process Management</b>	<b>117</b>
6.1.	Abort	117
6.1.1.	<b>PMIx_Abort</b>	117
6.2.	Process Creation	118
6.2.1.	<b>PMIx_Spawn</b>	118
6.2.2.	<b>PMIx_Spawn_nb</b>	122
6.3.	Connecting and Disconnecting Processes	126
6.3.1.	<b>PMIx_Connect</b>	127
6.3.2.	<b>PMIx_Connect_nb</b>	129
6.3.3.	<b>PMIx_Disconnect</b>	130
6.3.4.	<b>PMIx_Disconnect_nb</b>	132
<b>7.</b>	<b>Job Allocation Management and Reporting</b>	<b>134</b>
7.1.	Query	134
7.1.1.	<b>PMIx_Resolve_peers</b>	135
7.1.2.	<b>PMIx_Resolve_nodes</b>	135
7.1.3.	<b>PMIx_Query_info_nb</b>	136
7.2.	Allocation Requests	139
7.2.1.	<b>PMIx_Allocation_request_nb</b>	139
7.2.2.	<b>PMIx_Job_control_nb</b>	141
7.3.	Process and Job Monitoring	144
7.3.1.	<b>PMIx_Process_monitor_nb</b>	144
7.3.2.	<b>PMIx_Heartbeat</b>	146
7.4.	Logging	147
7.4.1.	<b>PMIx_Log_nb</b>	147
<b>8.</b>	<b>Event Notification</b>	<b>150</b>
8.1.	Notification and Management	150
8.1.1.	<b>PMIx_Register_event_handler</b>	152

8.1.2.	<code>PMIx_Deregister_event_handler</code>	155
8.1.3.	<code>PMIx_Notify_event</code>	156
<b>9.</b>	<b>Data Packing and Unpacking</b>	<b>159</b>
9.1.	Support Macros	159
9.1.1.	<code>PMIX_DATA_BUFFER_CREATE</code>	159
9.1.2.	<code>PMIX_DATA_BUFFER_RELEASE</code>	160
9.1.3.	<code>PMIX_DATA_BUFFER_CONSTRUCT</code>	160
9.1.4.	<code>PMIX_DATA_BUFFER_DESTRUCT</code>	160
9.1.5.	<code>PMIX_DATA_BUFFER_LOAD</code>	161
9.1.6.	<code>PMIX_DATA_BUFFER_UNLOAD</code>	161
9.2.	General Routines	162
9.2.1.	<code>PMIx_Data_pack</code>	162
9.2.2.	<code>PMIx_Data_unpack</code>	164
9.2.3.	<code>PMIx_Data_copy</code>	166
9.2.4.	<code>PMIx_Data_print</code>	166
9.2.5.	<code>PMIx_Data_copy_payload</code>	167
<b>10.</b>	<b>Server-Specific Interfaces</b>	<b>169</b>
10.1.	Server Support Functions	169
10.1.1.	<code>PMIx_generate_regex</code>	169
10.1.2.	<code>PMIx_generate_ppn</code>	170
10.1.3.	<code>PMIx_server_register_nspace</code>	171
10.1.4.	<code>PMIx_server_deregister_nspace</code>	175
10.1.5.	<code>PMIx_server_register_client</code>	176
10.1.6.	<code>PMIx_server_deregister_client</code>	177
10.1.7.	<code>PMIx_server_setup_fork</code>	178
10.1.8.	<code>PMIx_server_dmodex_request</code>	179
10.1.9.	<code>PMIx_server_setup_application</code>	180
10.1.10.	<code>PMIx_server_setup_local_support</code>	181
10.2.	Server Function Pointers	183
10.2.1.	<code>pmix_server_module_t</code> Module	183
10.2.2.	<code>pmix_server_client_connected_fn_t</code>	184
10.2.3.	<code>pmix_server_client_finalized_fn_t</code>	185



10.2.4.	<code>pmix_server_abort_fn_t</code>	187
10.2.5.	<code>pmix_server_fence_nb_fn_t</code>	188
10.2.6.	<code>pmix_server_dmodex_req_fn_t</code>	190
10.2.7.	<code>pmix_server_publish_fn_t</code>	191
10.2.8.	<code>pmix_server_lookup_fn_t</code>	193
10.2.9.	<code>pmix_server_unpublish_fn_t</code>	195
10.2.10.	<code>pmix_server_spawn_fn_t</code>	197
10.2.11.	<code>pmix_server_connect_fn_t</code>	201
10.2.12.	<code>pmix_server_disconnect_fn_t</code>	203
10.2.13.	<code>pmix_server_register_events_fn_t</code>	204
10.2.14.	<code>pmix_server_deregister_events_fn_t</code>	206
10.2.15.	<code>pmix_server_notify_event_fn_t</code>	207
10.2.16.	<code>pmix_server_listener_fn_t</code>	209
10.2.17.	<code>pmix_server_query_fn_t</code>	210
10.2.18.	<code>pmix_server_tool_connection_fn_t</code>	212
10.2.19.	<code>pmix_server_log_fn_t</code>	213
10.2.20.	<code>pmix_server_alloc_fn_t</code>	215
10.2.21.	<code>pmix_server_job_control_fn_t</code>	218
10.2.22.	<code>pmix_server_monitor_fn_t</code>	221
<b>A. Acknowledgements</b>		<b>224</b>
A.1.	Version 2.0	224
A.2.	Version 1.0	225
<b>Bibliography</b>		<b>227</b>
<b>Index</b>		<b>228</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 Application  
27 Programming Interfaces (APIs) – but rather to:

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

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

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

## 1.1 Charter

The charter of the PMIx community is to:

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

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

## 1.2 PMIx Standard Overview

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

## 1 1.2.1 Who should use the standard?

2 The PMIx Standard informs PMIx clients and RMs of the syntax and semantics of the PMIx APIs.  
3 PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to  
4 understand the set of attributes provided by various APIs of the PRI and their intended behavior.  
5 Additional information about the rationale for the selection of specific interfaces and attributes is  
6 also provided.  
7 PMIx-enabled RMs can use this standard to understand the expected behavior required of them  
8 when they support various interfaces/attributes. In addition, optional features and suggestions on  
9 behavior are also included in the discussion to help guide RM design and implementation.

## 10 1.2.2 What is defined in the standard?

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

## 15 1.2.3 What is *not* defined in the standard?

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

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

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

31 If an application (for example) truly relies on the [PMIX\\_TIMEOUT](#) attribute in a call to  
32 [PMIx\\_Fence](#) , it should set the required flag in the `pmix_info_t` for that attribute. This  
33 informs the library and its SMS host that it must return an immediate error if this attribute is not

1 supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as  
2 optional, ignoring it if support is not available.

3 It is therefore critical that users and application implementers:

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

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

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

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

## 27 **1.2.4 General Guidance for PMIx Users and Implementors**

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

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

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

## 6 1.3 PMIx Architecture Overview

7 This section presents a brief overview of the PMIx Architecture [1]. Note that this is a conceptual  
8 model solely used to help guide the standards process — it does not represent a design requirement  
9 on any PMIx implementation. Instead, the model is used by the PMIx community as a sounding  
10 board for evaluating proposed interfaces and avoid unintentionally imposing constraints on  
11 implementers. Built into the model are two guiding principles also reflected in the standard. First,  
12 PMIx operates in the mode of a *messenger*, and not a *doer* — i.e., the role of PMIx is to provide  
13 communication between the various participants, relaying requests and returning responses. The  
14 intent of the standard is not to suggest that PMIx itself actually perform any of the defined  
15 operations — this is left to the various SMS elements and/or the application. Any exceptions to that  
16 intent are left to the discretion of the particular implementation.

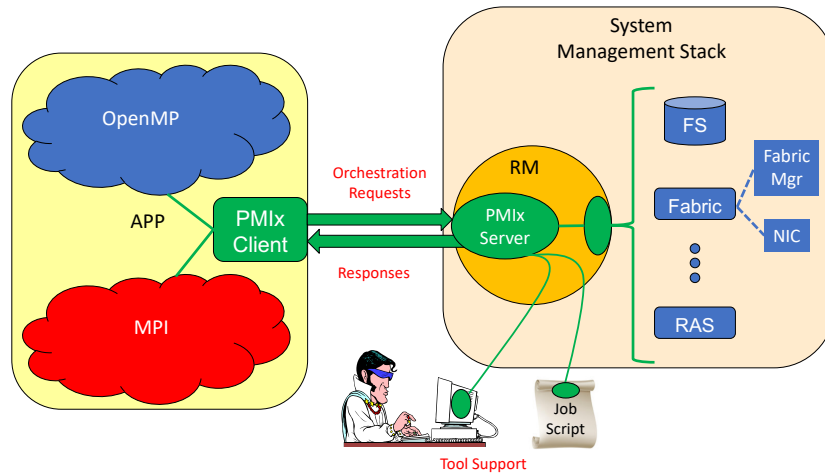


Figure 1.1.: PMIx-SMS Interactions

17 Thus, as the diagram in Fig. 1.1 shows, the application is built against a PMIx client library that  
18 contains the client-side APIs, attribute definitions, and communication support for interacting with  
19 the local PMIx server. Intra-process cross-library interactions are supported at the client level to  
20 avoid unnecessary burdens on the server. Orchestration requests are sent to the local PMIx server,  
21 which subsequently passes them to the host SMS (here represented by an RM daemon) using the

1 PMIx server callback functions the host SMS registered during `PMIx_server_init`. The host SMS  
2 can indicate its lack of support for any operation by simply providing a `NULL` for the associated  
3 callback function, or can create a function entry that returns *not supported* when called.

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

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

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

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

### 29 **1.3.1 The PMIx Reference Implementation (PRI)**

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

- 34 • Validation of the standard.  
35 No proposed change and/or extension to the PMIx standard is accepted without an accompanying  
36 prototype implementation in the PRI. This ensures that the proposal has undergone at least some  
37 minimal level of scrutiny and testing before being considered.

- Ease of adoption.

The PRI is designed to be particularly easy for resource managers (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for application portability. Both client and server PMIx libraries are included, along with examples of client usage and server-side integration. A list of supported environments and versions is maintained on the PMIx web site <https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/>

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

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

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

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

<https://pmix.org/code/getting-the-pmix-reference-server/>

## 1.4 Organization of this document

The remainder of this document is structured as follows:

- Introduction and Overview in Chapter 1 on page 1
- Terms and Conventions in Chapter 2 on page 10
- Data Structures and Types in Chapter 3 on page 15



- 1 • PMIx Initialization and Finalization in Chapter 4 on page 82
- 2 • Key/Value Management in Chapter 5 on page 95
- 3 • Process Management in Chapter 6 on page 117
- 4 • Job Management in Chapter 7 on page 134
- 5 • Event Notification in Chapter 8 on page 150
- 6 • Data Packing and Unpacking in Chapter 9 on page 159
- 7 • PMIx Server Specific Interfaces in Chapter 10 on page 169

## 8 1.5 Version 1.0: June 12, 2015

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

- 12 • Client APIs
  - 13 – PMIx\_Init, **PMIx\_Initialized**, **PMIx\_Abort**, **PMIx\_Finalize**
  - 14 – **PMIx\_Put**, **PMIx\_Commit**,
  - 15 – **PMIx\_Fence**, **PMIx\_Fence\_nb**
  - 16 – **PMIx\_Get**, **PMIx\_Get\_nb**
  - 17 – **PMIx\_Publish**, **PMIx\_Publish\_nb**
  - 18 – **PMIx\_Lookup**, **PMIx\_Lookup**
  - 19 – **PMIx\_Unpublish**, **PMIx\_Unpublish\_nb**
  - 20 – **PMIx\_Spawn**, **PMIx\_Spawn\_nb**
  - 21 – **PMIx\_Connect**, **PMIx\_Connect\_nb**
  - 22 – **PMIx\_Disconnect**, **PMIx\_Disconnect\_nb**
  - 23 – **PMIx\_Resolve\_nodes**, **PMIx\_Resolve\_peers**
- 24 • Server APIs
  - 25 – **PMIx\_server\_init**, **PMIx\_server\_finalize**
  - 26 – **PMIx\_generate\_regex**, **PMIx\_generate\_ppn**
  - 27 – **PMIx\_server\_register\_namespace**, **PMIx\_server\_deregister\_namespace**
  - 28 – **PMIx\_server\_register\_client**, **PMIx\_server\_deregister\_client**

- 1       – `PMIx_server_setup_fork`, `PMIx_server_dmodex_request`
- 2     • Common APIs
- 3       – `PMIx_Get_version`, `PMIx_Store_internal`, `PMIx_Error_string`
- 4       – `PMIx_Register_errhandler`, `PMIx_Deregister_errhandler`,
- 5         `PMIx_Notify_error`

6     The `PMIx_Init` API was subsequently modified in the PRI release v1.1.0.

## 7   1.6   Version 2.0: Sept. 2018

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

- 9     • Client APIs
- 10       – `PMIx_Query_info_nb`, `PMIx_Log_nb`
- 11       – `PMIx_Allocation_request_nb`, `PMIx_Job_control_nb`,
- 12         `PMIx_Process_monitor_nb`, `PMIx_Heartbeat`
- 13     • Server APIs
- 14       – `PMIx_server_setup_application`, `PMIx_server_setup_local_support`
- 15     • Tool APIs
- 16       – `PMIx_tool_init`, `PMIx_tool_finalize`
- 17     • Common APIs
- 18       – `PMIx_Register_event_handler`, `PMIx_Deregister_event_handler`
- 19       – `PMIx_Notify_event`
- 20       – `PMIx_Proc_state_string`, `PMIx_Scope_string`
- 21       – `PMIx_Persistence_string`, `PMIx_Data_range_string`
- 22       – `PMIx_Info_directives_string`, `PMIx_Data_type_string`
- 23       – `PMIx_Alloc_directive_string`
- 24       – `PMIx_Data_pack`, `PMIx_Data_unpack`, `PMIx_Data_copy`
- 25       – `PMIx_Data_print`, `PMIx_Data_copy_payload`

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

## CHAPTER 2

# PMIx Terms and Conventions

---

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

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

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

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

### Advice to users

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



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

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

## 12 2.1 Notational Conventions

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



15 C specific text...

16 `int foo = 42;`



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



19 Note: General text...



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



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

---

### Advice to PMIx library implementers

---

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

---

### Advice to PMIx server hosts

---

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

---

## 2.2 Semantics

9 The following terms will be taken to mean:

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

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

- 15 • *session* refers to an allocated set of resources assigned to a particular user by the system WLM.
- 16 • *job* refers to an application executed by the user within a session
- 17 • *resource manager* is used in a generic sense to represent the system that will host the PMIx  
18 server library. This could be a vendor's RM, a programming library's RTE, or some other agent.

## 1 2.3 Naming Conventions

2 The PMIx standard has adopted the following conventions:

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

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

## 15 2.4 Procedure Conventions

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

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

## 25 2.5 Standard vs Reference Implementation

26 The *PMIx Standard* is implementation independent. The *PMIx Reference Implementation* (PRI) is  
27 one implementation of the Standard and the PMIx community strives to ensure that it fully  
28 implements the Standard. Given its role as the community's testbed and its widespread use, this  
29 document cites the attributes supported by the PRI for each API where relevant by marking them in

1            **red**. This is not meant to imply nor confer any special role to the PRI with respect to the Standard  
2 itself, but instead to provide a convenience to users of the Standard and PRI.

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

## CHAPTER 3

# Data Structures and Types

---

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

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

### ▼ Advice to PMIx library implementers ▼

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

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

## 12 3.1 Constants

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

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

### ▼ Advice to PMIx library implementers ▼

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

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



### Advice to PMIx library implementers

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

## 4 3.1.1 Error Constants

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

6 The tables shown in this section define the possible values for **pmix\_status\_t** . PMIx errors are  
7 required to always be negative, with 0 reserved for **PMIX\_SUCCESS** .

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

### Advice to users

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

### 1 3.1.1.1 PMIx v1 Error Constants

2 The following list contains those constants defined in the PMIx v1 standard. Those values in the list  
3 that were deprecated in later standards are denoted as such. PMIx errors are always negative, with 0  
4 reserved for success.

5	<b>PMIX_SUCCESS</b>	Success
6	<b>PMIX_ERROR</b>	General Error
7	<b>PMIX_ERR_SILENT</b>	Silent error
8	<b>PMIX_ERR_DEBUGGER_RELEASE</b>	Error in debugger release
9	<b>PMIX_ERR_PROC_RESTART</b>	Fault tolerance: Error in process restart
10	<b>PMIX_ERR_PROC_CHECKPOINT</b>	Fault tolerance: Error in process checkpoint
11	<b>PMIX_ERR_PROC_MIGRATE</b>	Fault tolerance: Error in process migration
12	<b>PMIX_ERR_PROC_ABORTED</b>	Process was aborted
13	<b>PMIX_ERR_PROC_REQUESTED_ABORT</b>	Process is already requested to abort
14	<b>PMIX_ERR_PROC_ABORTING</b>	Process is being aborted
15	<b>PMIX_ERR_SERVER_FAILED_REQUEST</b>	Failed to connect to the server
16	<b>PMIX_EXISTS</b>	Requested operation would overwrite an existing value
17	<b>PMIX_ERR_INVALID_CRED</b>	Invalid security credentials
18	<b>PMIX_ERR_HANDSHAKE_FAILED</b>	Connection handshake failed
19	<b>PMIX_ERR_READY_FOR_HANDSHAKE</b>	Ready for handshake
20	<b>PMIX_ERR_WOULD_BLOCK</b>	Operation would block
21	<b>PMIX_ERR_UNKNOWN_DATA_TYPE</b>	Unknown data type
22	<b>PMIX_ERR_PROC_ENTRY_NOT_FOUND</b>	Process not found
23	<b>PMIX_ERR_TYPE_MISMATCH</b>	Invalid type
24	<b>PMIX_ERR_UNPACK_INADEQUATE_SPACE</b>	Inadequate space to unpack data
25	<b>PMIX_ERR_UNPACK_FAILURE</b>	Unpack failed
26	<b>PMIX_ERR_PACK_FAILURE</b>	Pack failed
27	<b>PMIX_ERR_PACK_MISMATCH</b>	Pack mismatch
28	<b>PMIX_ERR_NO_PERMISSIONS</b>	No permissions
29	<b>PMIX_ERR_TIMEOUT</b>	Timeout expired
30	<b>PMIX_ERR_UNREACH</b>	Unreachable
31	<b>PMIX_ERR_IN_ERRNO</b>	Error defined in <b>errno</b>
32	<b>PMIX_ERR_BAD_PARAM</b>	Bad parameter
33	<b>PMIX_ERR_RESOURCE_BUSY</b>	Resource busy
34	<b>PMIX_ERR_OUT_OF_RESOURCE</b>	Resource exhausted
35	<b>PMIX_ERR_DATA_VALUE_NOT_FOUND</b>	Data value not found
36	<b>PMIX_ERR_INIT</b>	Error during initialization
37	<b>PMIX_ERR_NOMEM</b>	Out of memory
38	<b>PMIX_ERR_INVALID_ARG</b>	Invalid argument
39	<b>PMIX_ERR_INVALID_KEY</b>	Invalid key
40	<b>PMIX_ERR_INVALID_KEY_LENGTH</b>	Invalid key length
41	<b>PMIX_ERR_INVALID_VAL</b>	Invalid value

1 **PMIX\_ERR\_INVALID\_VAL\_LENGTH** Invalid value length  
 2 **PMIX\_ERR\_INVALID\_LENGTH** Invalid argument length  
 3 **PMIX\_ERR\_INVALID\_NUM\_ARGS** Invalid number of arguments  
 4 **PMIX\_ERR\_INVALID\_ARGS** Invalid arguments  
 5 **PMIX\_ERR\_INVALID\_NUM\_PARSED** Invalid number parsed  
 6 **PMIX\_ERR\_INVALID\_KEYVALP** Invalid key/value pair  
 7 **PMIX\_ERR\_INVALID\_SIZE** Invalid size  
 8 **PMIX\_ERR\_INVALID\_NAMESPACE** Invalid namespace  
 9 **PMIX\_ERR\_SERVER\_NOT\_AVAIL** Server is not available  
 10 **PMIX\_ERR\_NOT\_FOUND** Not found  
 11 **PMIX\_ERR\_NOT\_SUPPORTED** Not supported  
 12 **PMIX\_ERR\_NOT\_IMPLEMENTED** Not implemented  
 13 **PMIX\_ERR\_COMM\_FAILURE** Communication failure  
 14 **PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER** Unpacking past the end of the buffer  
 15 provided

### 16 3.1.1.2 PMIx v2 Error Constants

17 The following list contains constants added in the PMIx v2 standard.

18 **PMIX\_ERR\_LOST\_CONNECTION\_TO\_SERVER** Lost connection to server  
 19 **PMIX\_ERR\_LOST\_PEER\_CONNECTION** Lost connection to peer  
 20 **PMIX\_ERR\_LOST\_CONNECTION\_TO\_CLIENT** Lost connection to client  
 21 **PMIX\_QUERY\_PARTIAL\_SUCCESS** Query partial success (used by query system)  
 22 **PMIX\_NOTIFY\_ALLOC\_COMPLETE** Notify that allocation is complete  
 23 **PMIX\_JCTRL\_CHECKPOINT** Job control: Monitored by PMIx client to trigger checkpoint  
 24 operation  
 25 **PMIX\_JCTRL\_CHECKPOINT\_COMPLETE** Job control: Sent by PMIx client and monitored  
 26 by PMIx server to notify that requested checkpoint operation has completed.  
 27 **PMIX\_JCTRL\_PREEMPT\_ALERT** Job control: Monitored by PMIx client to detect an RM  
 28 intending to preempt the job.  
 29 **PMIX\_MONITOR\_HEARTBEAT\_ALERT** Job monitoring: Heartbeat alert  
 30 **PMIX\_MONITOR\_FILE\_ALERT** Job monitoring: File alert  
 31 **PMIX\_PROC\_TERMINATED** Process terminated - can be either normal or abnormal  
 32 termination  
 33 **PMIX\_ERR\_INVALID\_TERMINATION** Process terminated without calling  
 34 **PMIx\_Finalize**, or was a member of an assemblage formed via **PMIx\_Connect** and  
 35 terminated or called **PMIx\_Finalize** without first calling **PMIx\_Disconnect** (or its  
 36 non-blocking form) from that assemblage.

37 The following list contains operational error constants introduced in the v2 standard.

38 **PMIX\_ERR\_EVENT\_REGISTRATION** Error in event registration  
 39 **PMIX\_ERR\_JOB\_TERMINATED** Error job terminated  
 40 **PMIX\_ERR\_UPDATE\_ENDPOINTS** Error updating endpoints

1 **PMIX\_MODEL\_DECLARED** Model declared  
2 **PMIX\_GDS\_ACTION\_COMPLETE** The global data storage (GDS) action has completed  
3 **PMIX\_ERR\_INVALID\_OPERATION** The requested operation is supported by the  
4 implementation and host environment, but fails to meet a requirement (e.g., requesting to  
5 *disconnect* from processes without first *connecting* to them).

6 The following list contains system error constants introduced in the v2 standard.

7 **PMIX\_ERR\_NODE\_DOWN** Node down  
8 **PMIX\_ERR\_NODE\_OFFLINE** Node is marked as offline

9 The following list contains event handler error constants introduced in the v2 standard.

10 **PMIX\_EVENT\_NO\_ACTION\_TAKEN** Event handler: No action taken  
11 **PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN** Event handler: Partial action taken  
12 **PMIX\_EVENT\_ACTION\_DEFERRED** Event handler: Action deferred  
13 **PMIX\_EVENT\_ACTION\_COMPLETE** Event handler: Action complete

### 14 3.1.1.3 User-Defined Error Constants

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

18 **PMIX\_EXTERNAL\_ERR\_BASE** A starting point for user-level defined error constants.  
19 Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions  
20 should always be based on the **PMIX\_EXTERNAL\_ERR\_BASE** constant and *not* a specific  
21 value as the value of the constant may change.

## 22 3.2 Data Types

23 This section defines various data types used by the PMIx APIs.

### 24 3.2.1 Key Structure

25 The **pmix\_key\_t** structure is a statically defined character array of length **PMIX\_MAX\_KEYLEN**  
26 +1, thus supporting keys of maximum length **PMIX\_MAX\_KEYLEN** while preserving space for a  
27 mandatory **NULL** terminator.

PMIx v2.0

```
28 typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];
```

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

### Advice to users

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

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

## 10 3.2.2 Namespace Structure

11 The `pmix_namespace_t` structure is a statically defined character array of length  
 12 **PMIX\_MAX\_NSLEN+1**, thus supporting namespaces of maximum length **PMIX\_MAX\_NSLEN**  
 13 while preserving space for a mandatory **NULL** terminator.

PMIx v2.0

```
14 typedef char pmix_namespace_t [PMIX_MAX_NSLEN+1];
```

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

### Advice to users

17 References to namespace values in PMIx v1 rwere defined simply as an array of characters of size  
 18 **PMIX\_MAX\_NSLEN+1**. The `pmix_namespace_t` type definition was introduced in version 2 of the  
 19 standard. The two definitions are code-compatible and thus do not represent a break in backward  
 20 compatibility.

21 Passing a `pmix_namespace_t` value to the standard *sizeof* utility can result in compiler warnings of  
 22 incorrect returned value. Users are advised to avoid using *sizeof(pmix\_namespace\_t)* and instead rely  
 23 on the **PMIX\_MAX\_NSLEN** constant.

### 1 3.2.3 Rank Structure

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

*PMIx v1.0*

```
▼ _____ C _____ ▼  
3 typedef uint32_t pmix_rank_t;  
▲ _____ C _____ ▲
```

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

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

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

12 *PMIx v2.0* **PMIX\_RANK\_LOCAL\_NODE** Special rank value used to define groups of ranks for use in  
13 collectives. This constant defines the group of all ranks on a local node.

### 14 3.2.4 Process Structure

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

*PMIx v1.0*

```
▼ _____ C _____ ▼  
17 typedef struct pmix_proc {  
18     pmix_namespace_t nspace;  
19     pmix_rank_t rank;  
20 } pmix_proc_t;  
▲ _____ C _____ ▲
```

### 21 3.2.5 Process structure support macros

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

### 1 3.2.5.1 Initialize the `pmix_proc_t` structure

2 Initialize the `pmix_proc_t` fields

*PMIx v1.0*

▼  C  ▼

3 **PMIX\_PROC\_CONSTRUCT** (m)

▲  C  ▲

4 **IN** m

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

### 6 3.2.5.2 Destruct the `pmix_proc_t` structure

7 Clear the `pmix_proc_t` fields

*PMIx v1.0*

▼  C  ▼

8 **PMIX\_PROC\_DESTRUCT** (m)

▲  C  ▲

9 **IN** m

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

11 This macro performs the identical operations as **PMIX\_PROC\_CONSTRUCT**, but is provided for  
12 symmetry in user code.

### 13 3.2.5.3 Create a `pmix_proc_t` array

14 Allocate and initialize an array of `pmix_proc_t` structures

*PMIx v1.0*

▼  C  ▼

15 **PMIX\_PROC\_CREATE** (m, n)

▲  C  ▲

16 **INOUT** m

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

18 **IN** n

19 Number of structures to be allocated (**size\_t**)

### 1 3.2.5.4 Free a `pmix_proc_t` array

2 Release an array of `pmix_proc_t` structures

*PMIx v1.0*



3 **PMIX\_PROC\_FREE** (m, n)



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

### 8 3.2.5.5 Load a `pmix_proc_t` structure

9 Load values into a `pmix_proc_t`

*PMIx v2.0*



10 **PMIX\_PROC\_LOAD** (m, n, r)



- 11 **IN** m
- 12     Pointer to the structure to be loaded (pointer to `pmix_proc_t` )
- 13 **IN** n
- 14     Namespace to be loaded ( `pmix_namespace_t` )
- 15 **IN** r
- 16     Rank to be assigned ( `pmix_rank_t` )

## 17 3.2.6 Process State Structure

18 *PMIx v2.0* The `pmix_proc_state_t` structure is a `uint8_t` type for process state values. The following  
19 constants can be used to set a variable of the type `pmix_proc_state_t` . All values were  
20 originally defined in version 2 of the standard unless otherwise marked.



Advice to users

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



---

1     **PMIX\_PROC\_STATE\_UNDEF**     Undefined process state  
2     **PMIX\_PROC\_STATE\_PREPPED**     Process is ready to be launched  
3     **PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY**     Process launch is underway  
4     **PMIX\_PROC\_STATE\_RESTART**     Process is ready for restart  
5     **PMIX\_PROC\_STATE\_TERMINATE**     Process is marked for termination  
6     **PMIX\_PROC\_STATE\_RUNNING**     Process has been locally **fork**'ed by the RM  
7     **PMIX\_PROC\_STATE\_CONNECTED**     Process has connected to PMIx server  
8     **PMIX\_PROC\_STATE\_UNTERMINATED**     Define a "boundary" between this constant and  
9         **PMIX\_PROC\_STATE\_CONNECTED** so users can easily and quickly determine if a process  
10         is still running or not. Any value less than this constant means that the process has not  
11         terminated.  
12     **PMIX\_PROC\_STATE\_TERMINATED**     Process has terminated and is no longer running  
13     **PMIX\_PROC\_STATE\_ERROR**     Define a boundary so users can easily and quickly determine if  
14         a process abnormally terminated. Any value above this constant means that the process has  
15         terminated abnormally.  
16     **PMIX\_PROC\_STATE\_KILLED\_BY\_CMD**     Process was killed by a command  
17     **PMIX\_PROC\_STATE\_ABORTED**     Process was aborted by a call to **PMIx\_Abort**  
18     **PMIX\_PROC\_STATE\_FAILED\_TO\_START**     Process failed to start  
19     **PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG**     Process aborted by a signal  
20     **PMIX\_PROC\_STATE\_TERM\_WO\_SYNC**     Process exited without calling **PMIx\_Finalize**  
21     **PMIX\_PROC\_STATE\_COMM\_FAILED**     Process communication has failed  
22     **PMIX\_PROC\_STATE\_CALLED\_ABORT**     Process called **PMIx\_Abort**  
23     **PMIX\_PROC\_STATE\_MIGRATING**     Process failed and is waiting for resources before  
24         restarting  
25     **PMIX\_PROC\_STATE\_CANNOT\_RESTART**     Process failed and cannot be restarted  
26     **PMIX\_PROC\_STATE\_TERM\_NON\_ZERO**     Process exited with a non-zero status  
27     **PMIX\_PROC\_STATE\_FAILED\_TO\_LAUNCH**     Unable to launch process

## 28   **3.2.7 Process Information Structure**

29     The **pmix\_proc\_info\_t** structure defines a set of information about a specific process  
30     including it's name, location, and state.

*PMIx v2.0*

```

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

```

### 15 3.2.8 Process Information Structure support macros

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

#### 17 3.2.8.1 Initialize the `pmix_proc_info_t` structure

18 Initialize the `pmix_proc_info_t` fields

*PMIx v2.0*

```

19 PMIX_PROC_INFO_CONSTRUCT(m)

```

20 **IN** m

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

#### 22 3.2.8.2 Destruct the `pmix_proc_info_t` structure

23 Destruct the `pmix_proc_info_t` fields

*PMIx v2.0*

```

24 PMIX_PROC_INFO_DESTRUCT(m)

```

25 **IN** m

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

### 1 3.2.8.3 Create a `pmix_proc_info_t` array

2 Allocate and initialize a `pmix_proc_info_t` array

*PMIx v2.0*

▼ `PMIX_PROC_INFO_CREATE` C ▼

3 `PMIX_PROC_INFO_CREATE` (*m*, *n*)

▲ `PMIX_PROC_INFO_CREATE` C ▲

4 **INOUT** *m*

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

7 **IN** *n*

8 Number of structures to be allocated (**size\_t**)

### 9 3.2.8.4 Free a `pmix_proc_info_t` array

10 Release an array of `pmix_proc_info_t` structures

*PMIx v2.0*

▼ `PMIX_PROC_INFO_FREE` C ▼

11 `PMIX_PROC_INFO_FREE` (*m*, *n*)

▲ `PMIX_PROC_INFO_FREE` C ▲

12 **IN** *m*

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

14 **IN** *n*

15 Number of structures in the array (**size\_t**)

## 16 3.2.9 Scope of Put Data

17 *PMIx v1.0* The `pmix_scope_t` structure is a `uint8_t` type that defines the scope for data passed to  
18 `PMIx_Put`. The following constants can be used to set a variable of the type `pmix_scope_t`.  
19 All definitions were introduced in version 1 of the standard unless otherwise marked.

20 Specific implementations may support different scope values, but all implementations must support  
21 at least `PMIX_GLOBAL`. If a scope value is not supported, then the `PMIx_Put` call must return  
22 `PMIX_ERR_NOT_SUPPORTED`.

23 **PMIX\_SCOPE\_UNDEF** Undefined scope

24 **PMIX\_LOCAL** The data is intended only for other application processes on the same node.

25 Data marked in this way will not be included in data packages sent to remote requestors —  
26 i.e., it is only available to processes on the local node.

27 **PMIX\_REMOTE** The data is intended solely for applications processes on remote nodes. Data  
28 marked in this way will not be shared with other processes on the same node — i.e., it is only  
29 available to processes on remote nodes.

1           **PMIX\_GLOBAL**     The data is to be shared with all other requesting processes, regardless of  
2                            location.  
3 *PMIx v2.0* **PMIX\_INTERNAL**   The data is intended solely for this process and is not shared with other  
4                            processes.

## 5 **3.2.10 Range of Published Data**

6 *PMIx v1.0* The **pmix\_data\_range\_t** structure is a **uint8\_t** type that defines a range for data *published*  
7           via functions other than **PMIx\_Put** - e.g., the **PMIx\_Publish** API. The following constants  
8           can be used to set a variable of the type **pmix\_data\_range\_t** . Several values were initially  
9           defined in version 1 of the standard but subsequently renamed and other values added in version 2.  
10          Thus, all values shown below are as they were defined in version 2 except where noted.

11         **PMIX\_RANGE\_UNDEF**    Undefined range  
12         **PMIX\_RANGE\_RM**     Data is intended for the host resource manager.  
13         **PMIX\_RANGE\_LOCAL**    Data is only available to processes on the local node.  
14         **PMIX\_RANGE\_NAMESPACE** Data is only available to processes in the same namespace.  
15         **PMIX\_RANGE\_SESSION** Data is only available to all processes in the session.  
16         **PMIX\_RANGE\_GLOBAL**   Data is available to all processes.  
17         **PMIX\_RANGE\_CUSTOM**   Range is specified in the **pmix\_info\_t** associated with this call.  
18         **PMIX\_RANGE\_PROC\_LOCAL** Data is only available to this process.

### Advice to users

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

## 21 **3.2.11 Data Persistence Structure**

22 *PMIx v1.0* The **pmix\_persistence\_t** structure is a **uint8\_t** type that defines the policy for data  
23           published by clients via the **PMIx\_Publish** API. The following constants can be used to set a  
24           variable of the type **pmix\_persistence\_t** . All definitions were introduced in version 1 of the  
25           standard unless otherwise marked.

26         **PMIX\_PERSIST\_INDEF**    Retain data until specifically deleted.  
27         **PMIX\_PERSIST\_FIRST\_READ** Retain data until the first access, then the data is deleted.  
28         **PMIX\_PERSIST\_PROC**     Retain data until the publishing process terminates.  
29         **PMIX\_PERSIST\_APP**     Retain data until the application terminates.  
30         **PMIX\_PERSIST\_SESSION** Retain data until the session/allocation terminates.

## 1 3.2.12 Value Structure

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

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

*PMIx v1.0*

C

```
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; // version 2.0
30        pmix_status_t status; // version 2.0
31        pmix_rank_t rank; // version 2.0
32        pmix_proc_t *proc; // version 2.0
33        pmix_byte_object_t bo;
34        pmix_persistence_t persist; // version 2.0
35        pmix_scope_t scope; // version 2.0
36        pmix_data_range_t range; // version 2.0
37        pmix_proc_state_t state; // version 2.0
38        pmix_proc_info_t *pinfo; // version 2.0
39        pmix_data_array_t *darray; // version 2.0
40        void *ptr; // version 2.0
```

```

1         pmix_alloc_directive_t adir;    // version 2.0
2         /**** DEPRECATED in PMIx 2 ****/
3         pmix_info_array_t *array;
4         /*****
5         } data;
6     } pmix_value_t;

```



C

### 7 3.2.13 Value structure support macros

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

#### 9 3.2.13.1 Initialize the `pmix_value_t` structure

10 Initialize the `pmix_value_t` fields

*PMIx v1.0*



C

11 **PMIX\_VALUE\_CONSTRUCT** (m)



C

12 **IN** m

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

#### 14 3.2.13.2 Destruct the `pmix_value_t` structure

15 Destruct the `pmix_value_t` fields

*PMIx v1.0*



C

16 **PMIX\_VALUE\_DESTRUCT** (m)



C

17 **IN** m

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

#### 19 3.2.13.3 Create a `pmix_value_t` array

20 Allocate and initialize an array of `pmix_value_t` structures

*PMIx v1.0*



C

21 **PMIX\_VALUE\_CREATE** (m, n)



C

22 **INOUT** m

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

25 **IN** n

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

### 1 3.2.13.4 Free a `pmix_value_t` array

2 Release an array of `pmix_value_t` structures

*PMIx v1.0*

▼ `PMIX_VALUE_FREE` C

3 `PMIX_VALUE_FREE(m, n)`

▲ `PMIX_VALUE_FREE` C

4 **IN** `m`

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

6 **IN** `n`

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

### 8 3.2.14 Load a `pmix_value_t` structure

#### 9 Summary

10 Load data into a `pmix_value_t` structure.

*PMIx v2.0*

▼ `PMIX_VALUE_LOAD` C

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

▲ `PMIX_VALUE_LOAD` C

12 **IN** `v`

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

14 **IN** `d`

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

16 **IN** `t`

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

#### 18 Description

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

▼ **Advice to users** ▼

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

▲

### 1 3.2.14.1 Transfer data between `pmix_value_t` structures

#### 2 Summary

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

*PMIx v2.0*

```
▼ C ▼  
4 PMIX_VALUE_XFER(r, d, s);  
▲ C ▲
```

5 **OUT** r

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

7 **IN** d

8 Pointer to the `pmix_value_t` destination (handle)

9 **IN** s

10 Pointer to the `pmix_value_t` source (handle)

#### 11 Description

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

▼ Advice to users ▼

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

▲

### 16 3.2.15 Info and Info Array Structures

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

*PMIx v1.0*

```
▼ C ▼  
19 typedef struct pmix_info_t {  
20     pmix_key_t key;  
21     pmix_info_directives_t flags;    // version 2.0  
22     pmix_value_t value;  
23 } pmix_info_t;  
▲ C ▲
```

24 The `pmix_info_array` structure defines an array of `pmix_info_t` structures.



1 Note: The `pmix_info_array` structure has been deprecated and will be removed in future  
2 versions of the PMIx Standard.

*PMIx v1.0*

```
3 typedef struct pmix_info_array {  
4     size_t size;  
5     pmix_info_t *array;  
6 } pmix_info_array_t;
```

## 7 3.2.16 Info structure support macros

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

### 9 3.2.16.1 Initialize the `pmix_info_t` structure

10 Initialize the `pmix_info_t` fields

*PMIx v1.0*

```
11 PMIX_INFO_CONSTRUCT (m)
```

12 **IN** m

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

### 14 3.2.16.2 Destruct the `pmix_info_t` structure

15 Destruct the `pmix_info_t` fields

*PMIx v1.0*

```
16 PMIX_INFO_DESTRUCT (m)
```

17 **IN** m

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

### 1 3.2.16.3 Create a `pmix_info_t` array

2 Allocate and initialize an array of `pmix_info_t` structures

*PMIx v1.0*

▼ `PMIX_INFO_CREATE` C ▲

3 `PMIX_INFO_CREATE(m, n)`

▲ `PMIX_INFO_CREATE` C ▲

4 **INOUT** `m`

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

6 **IN** `n`

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

### 8 3.2.16.4 Free a `pmix_info_t` array

9 Release an array of `pmix_info_t` structures

*PMIx v1.0*

▼ `PMIX_INFO_FREE` C ▲

10 `PMIX_INFO_FREE(m, n)`

▲ `PMIX_INFO_FREE` C ▲

11 **IN** `m`

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

13 **IN** `n`

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

### 15 3.2.16.5 Load key and value data into a `pmix_info_t`

*PMIx v1.0*

▼ `PMIX_INFO_LOAD` C ▲

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

▲ `PMIX_INFO_LOAD` C ▲

17 **IN** `v`

18 Pointer to the `pmix_info_t` into which the key and data are to be loaded (pointer to  
19 `pmix_info_t`)

20 **IN** `k`

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

23 **IN** `d`

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

25 **IN** `t`

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

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

### Advice to users

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

#### 3.2.16.6 Copy data between `pmix_info_t` structures

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

*PMIx v2.0*

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

**IN** `d`

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

**IN** `s`

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

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

### Advice to users

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

#### 3.2.16.7 Test a boolean `pmix_info_t`

A special macro for checking if a boolean `pmix_info_t` is `true`

*PMIx v2.0*

```
PMIX_INFO_TRUE(m)
```

**IN** `m`

Pointer to a `pmix_info_t` structure (handle)

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

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

## 1 3.2.17 Info Type Directives

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

### ▼ Advice to users ▼

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

### ▼ Advice to PMIx library implementers ▼

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

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

18 `PMIX_INFO_REQD` The behavior defined in the `pmix_info_t` array is required, and not  
19 optional. This is a bit-mask value.

### ▼ Advice to PMIx server hosts ▼

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

## 23 3.2.18 Info Directive support macros

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

### 1 3.2.18.1 Mark an info structure as required

#### 2 Summary

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

*PMIx v2.0*

```
▼ _____ C _____ ▼  
4 PMIX_INFO_REQUIRED (info);  
▲ _____ C _____ ▲
```

5 **IN** `info`

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

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

### 8 3.2.18.2 Test an info structure for *required* directive

#### 9 Summary

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

*PMIx v2.0*

```
▼ _____ C _____ ▼  
11 PMIX_INFO_IS_REQUIRED (info);  
▲ _____ C _____ ▲
```

12 **IN** `info`

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

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

### 15 3.2.19 Job Allocation Directives

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

20 **PMIX\_ALLOC\_NEW** A new allocation is being requested. The resulting allocation will be  
21 disjoint (i.e., not connected in a job sense) from the requesting allocation.

22 **PMIX\_ALLOC\_EXTEND** Extend the existing allocation, either in time or as additional  
23 resources.

24 **PMIX\_ALLOC\_RELEASE** Release part of the existing allocation. Attributes in the  
25 accompanying `pmix_info_t` array may be used to specify permanent release of the  
26 identified resources, or “lending” of those resources for some period of time.

27 **PMIX\_ALLOC\_REAQUIRE** Reacquire resources that were previously “lent” back to the  
28 scheduler.

29 **PMIX\_ALLOC\_EXTERNAL** A value boundary above which implementers are free to define  
30 their own directive values.

## 1 3.2.20 Lookup Returned Data Structure

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

*PMIx v1.0*

```
3 typedef struct pmix_pdata {  
4     pmix_proc_t proc;  
5     pmix_key_t key;  
6     pmix_value_t value;  
7 } pmix_pdata_t;
```

## 8 3.2.21 Lookup data structure support macros

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

### 10 3.2.21.1 Initialize the `pmix_pdata_t` structure

11 Initialize the `pmix_pdata_t` fields

*PMIx v1.0*

```
12 PMIX_PDATA_CONSTRUCT (m)
```

13 **IN** m

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

### 15 3.2.21.2 Destruct the `pmix_pdata_t` structure

16 Destruct the `pmix_pdata_t` fields

*PMIx v1.0*

```
17 PMIX_PDATA_DESTRUCT (m)
```

18 **IN** m

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

### 1 3.2.21.3 Create a `pmix_pdata_t` array

2 Allocate and initialize an array of `pmix_pdata_t` structures

*PMIx v1.0*

▼ `PMIX_PDATA_CREATE` C `PMIX_PDATA_CREATE` ▼

3 `PMIX_PDATA_CREATE` (`m`, `n`)

▲ `PMIX_PDATA_CREATE` C `PMIX_PDATA_CREATE` ▲

4 **INOUT** `m`

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

7 **IN** `n`

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

### 9 3.2.21.4 Free a `pmix_pdata_t` array

10 Release an array of `pmix_pdata_t` structures

*PMIx v1.0*

▼ `PMIX_PDATA_FREE` C `PMIX_PDATA_FREE` ▼

11 `PMIX_PDATA_FREE` (`m`, `n`)

▲ `PMIX_PDATA_FREE` C `PMIX_PDATA_FREE` ▲

12 **IN** `m`

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

14 **IN** `n`

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

### 16 3.2.21.5 Load a lookup data structure

#### 17 Summary

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

*PMIx v1.0*

▼ `PMIX_PDATA_LOAD` C `PMIX_PDATA_LOAD` ▼

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

▲ `PMIX_PDATA_LOAD` C `PMIX_PDATA_LOAD` ▲

20 **IN** `m`

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

23 **IN** `p`

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

26 **IN** `k`

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

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

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

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

▼ **Advice to users** ▼

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



### 10 3.2.21.6 Transfer a lookup data structure

#### 11 Summary

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

*PMIx v2.0*



C

13 **PMIX\_PDATA\_XFER**(**d**, **s**);



C

14 **IN** **d**  
15     Pointer to the destination `pmix_pdata_t` (pointer to `pmix_pdata_t` )

16 **IN** **s**  
17     Pointer to the source `pmix_pdata_t` (pointer to `pmix_pdata_t` )

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

▼ **Advice to users** ▼

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





## 1 3.2.22 Application Structure

2 The `pmix_app_t` structure describes the application context for the `PMIX_Spawn` and  
3 `PMIX_Spawn_nb` operations.

*PMIx v1.0*

C

```
4 typedef struct pmix_app {  
5     /** Executable */  
6     char *cmd;  
7     /** Argument set, NULL terminated */  
8     char **argv;  
9     /** Environment set, NULL terminated */  
10    char **env;  
11    /** Current working directory */  
12    char *cwd;  
13    /** Maximum processes with this profile */  
14    int maxprocs;  
15    /** Array of info keys describing this application*/  
16    pmix_info_t *info;  
17    /** Number of info keys in 'info' array */  
18    size_t ninfo;  
19 }
```

```
} pmix_app_t;
```

C

## 20 3.2.23 App structure support macros

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

### 22 3.2.23.1 Initialize the `pmix_app_t` structure

23 Initialize the `pmix_app_t` fields

*PMIx v1.0*

C

```
24 PMIX_APP_CONSTRUCT (m)
```

C

25 **IN** m

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

### 1 3.2.23.2 Destruct the `pmix_app_t` structure

2 Destruct the `pmix_app_t` fields

*PMIx v1.0*

▼  C  ▼

3 **PMIX\_APP\_DESTRUCT** (m)

▲  C  ▲

4 **IN** m

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

### 6 3.2.23.3 Create a `pmix_app_t` array

7 Allocate and initialize an array of `pmix_app_t` structures

*PMIx v1.0*

▼  C  ▼

8 **PMIX\_APP\_CREATE** (m, n)

▲  C  ▲

9 **INOUT** m

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

11 **IN** n

12 Number of structures to be allocated (**size\_t**)

### 13 3.2.23.4 Free a `pmix_app_t` array

14 Release an array of `pmix_app_t` structures

*PMIx v1.0*

▼  C  ▼

15 **PMIX\_APP\_FREE** (m, n)

▲  C  ▲

16 **IN** m

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

18 **IN** n

19 Number of structures in the array (**size\_t**)

## 1 3.2.24 Query Structure

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

*PMIx v2.0*

```
4 typedef struct pmix_query {  
5     char **keys;  
6     pmix_info_t *qualifiers;  
7     size_t nqual;  
8 } pmix_query_t;
```

C

C

## 9 3.2.25 Query structure support macros

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

### 11 3.2.25.1 Initialize the `pmix_query_t` structure

12 Initialize the `pmix_query_t` fields

*PMIx v2.0*

```
13 PMIX_QUERY_CONSTRUCT (m)
```

14 **IN** m

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

C

C

### 16 3.2.25.2 Destruct the `pmix_query_t` structure

17 Destruct the `pmix_query_t` fields

*PMIx v2.0*

```
18 PMIX_QUERY_DESTRUCT (m)
```

19 **IN** m

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

C

C

### 1 3.2.25.3 Create a `pmix_query_t` array

2 Allocate and initialize an array of `pmix_query_t` structures

*PMIx v2.0*

▼ `PMIX_QUERY_CREATE` C ▲

3 `PMIX_QUERY_CREATE` (*m*, *n*)

▲ `PMIX_QUERY_CREATE` C ▲

4 **INOUT** *m*

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

7 **IN** *n*

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

### 9 3.2.25.4 Free a `pmix_query_t` array

10 Release an array of `pmix_query_t` structures

*PMIx v2.0*

▼ `PMIX_QUERY_FREE` C ▲

11 `PMIX_QUERY_FREE` (*m*, *n*)

▲ `PMIX_QUERY_FREE` C ▲

12 **IN** *m*

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

14 **IN** *n*

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

## 16 3.2.26 Modex Structure

17 The `pmix_modex_data_t` structure describes the business card exchange (BCX) information.

▼ `pmix_modex_data_t` ▲

18 Note: This structure and its supporting macros have been deprecated and will be removed in future  
19 versions of the PMIx Standard.

▲ `pmix_modex_data_t` ▲

*PMIx v1.0*

▼ `pmix_modex_data_t` C ▲

```
20 typedef struct pmix_modex_data {  
21     pmix_nspace_t nspace;  
22     int rank;  
23     uint8_t *blob;  
24     size_t size;  
25 } pmix_modex_data_t;
```

▲ `pmix_modex_data_t` C ▲

## 1 3.2.27 Modex data structure support macros

2 The following macros are provided to support the `pmix_modex_t` structure.

### 3 3.2.27.1 Initialize the `pmix_modex_t` structure

4 Initialize the `pmix_modex_t` fields

*PMIx v1.0*

▼  C  ▼

5 **PMIX\_MODEX\_CONSTRUCT** (m)

▲  C  ▲

6 **IN** m

7 Pointer to the structure to be initialized (pointer to `pmix_modex_t`)

### 8 3.2.27.2 Destruct the `pmix_modex_t` structure

9 Destruct the `pmix_modex_t` fields

*PMIx v1.0*

▼  C  ▼

10 **PMIX\_MODEX\_DESTRUCT** (m)

▲  C  ▲

11 **IN** m

12 Pointer to the structure to be destructed (pointer to `pmix_modex_t`)

### 13 3.2.27.3 Create a `pmix_modex_t` array

14 Allocate and initialize an array of `pmix_modex_t` structures

*PMIx v1.0*

▼  C  ▼

15 **PMIX\_MODEX\_CREATE** (m, n)

▲  C  ▲

16 **INOUT** m

17 Address where the pointer to the array of `pmix_modex_t` structures shall be stored  
18 (handle)

19 **IN** n

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

#### 1 3.2.27.4 Free a `pmix_modex_t` array

2 Release an array of `pmix_modex_t` structures

*PMIx v1.0*

▼  C 

3 `PMIX_MODEX_FREE(m, n)`

▲  C 

4 **IN** `m`

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

6 **IN** `n`

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

### 8 3.3 Data Packing/Unpacking Types and Structures

9 This section defines types and structures used to pack and unpack data passed through the PMIx  
10 API.

#### 11 3.3.1 Byte Object Type

12 The `pmix_byte_object_t` structure describes a raw byte sequence.

*PMIx v1.0*

▼  C 

```
13 typedef struct pmix_byte_object {  
14     char *bytes;  
15     size_t size;  
16 } pmix_byte_object_t;
```

▲  C 

#### 17 3.3.2 Byte object support macros

18 The following macros support the `pmix_byte_object_t` structure.

##### 19 3.3.2.1 Initialize the `pmix_byte_object_t` structure

20 Initialize the `pmix_byte_object_t` fields

*PMIx v2.0*

▼  C 

21 `PMIX_BYTE_OBJECT_CONSTRUCT(m)`

▲  C 

22 **IN** `m`

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

### 1 3.3.2.2 Destruct the `pmix_byte_object_t` structure

2 Clear the `pmix_byte_object_t` fields

*PMIx v2.0*

▼  

3 **PMIX\_BYTE\_OBJECT\_DESTRUCT** (m)

▲  

4 **IN** m

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

### 6 3.3.2.3 Create a `pmix_byte_object_t` structure

7 Allocate and initialize an array of `pmix_byte_object_t` structures

*PMIx v2.0*

▼  

8 **PMIX\_BYTE\_OBJECT\_CREATE** (m, n)

▲  

9 **INOUT** m

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

12 **IN** n

13 Number of structures to be allocated (**size\_t**)

### 14 3.3.2.4 Free a `pmix_byte_object_t` array

15 Release an array of `pmix_byte_object_t` structures

*PMIx v2.0*

▼  

16 **PMIX\_BYTE\_OBJECT\_FREE** (m, n)

▲  

17 **IN** m

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

19 **IN** n

20 Number of structures in the array (**size\_t**)

### 1 3.3.2.5 Load a `pmix_byte_object_t` structure

2 Load values into a `pmix_byte_object_t`

*PMIx v2.0*

▼ `PMIX_BYTE_OBJECT_LOAD(b, d, s)` C  ▼

3 `PMIX_BYTE_OBJECT_LOAD(b, d, s)`

▲ `PMIX_BYTE_OBJECT_LOAD(b, d, s)` C  ▲

4 **IN** `b`

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

6 **IN** `d`

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

8 **IN** `s`

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

### 10 3.3.3 Data Buffer Type

11 The `pmix_data_buffer_t` structure describes a data buffer used for packing and unpacking.

*PMIx v2.0*

▼ `pmix_data_buffer_t` C  ▼

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

▲ `pmix_data_buffer_t` C  ▲

### 27 3.3.4 Data buffer support macros

28 The following macros support the `pmix_data_buffer_t` structure.



1 **3.3.4.1 Initialize the `pmix_data_buffer_t` structure**

2 Initialize the `pmix_data_buffer_t` fields

*PMIx v2.0*



3 **PMIX\_DATA\_BUFFER\_CONSTRUCT (m)**



4 **IN m**

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

6 **3.3.4.2 Destruct the `pmix_data_buffer_t` structure**

7 Clear the `pmix_data_buffer_t` fields

*PMIx v2.0*



8 **PMIX\_DATA\_BUFFER\_DESTRUCT (m)**



9 **IN m**

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

11 **3.3.4.3 Create a `pmix_data_buffer_t` structure**

12 Allocate and initialize a `pmix_data_buffer_t` structure

*PMIx v2.0*



13 **PMIX\_DATA\_BUFFER\_CREATE (m)**



14 **INOUT m**

15 Address where the pointer to the `pmix_data_buffer_t` structure shall be stored  
16 (handle)

17 **3.3.4.4 Free a `pmix_data_buffer_t`**

18 Release a `pmix_data_buffer_t` structure

*PMIx v2.0*



19 **PMIX\_DATA\_BUFFER\_RELEASE (m)**



20 **IN m**

21 Pointer to the `pmix_data_buffer_t` structure to be released (handle)

## 1 3.3.5 Data Array Structure

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

*PMIx v2.0*

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

C

C

## 8 3.3.6 Generalized Data Types Used for Packing/Unpacking

9 The `pmix_data_type_t` structure is a `uint16_t` type for identifying the data type for  
10 packing/unpacking purposes.

### Advice to PMIx library implementers

11 The following constants can be used to set a variable of the type `pmix_data_type_t`. Data  
12 types in the PMIx Standard are defined in terms of the C-programming language. Implementers  
13 wishing to support other languages should provide the equivalent definitions in a  
14 language-appropriate manner. Additionally, a PMIx implementation may choose to add additional  
15 types.

### 16 3.3.6.1 PMIx v1 Data Types

17 The following types were introduced in version 1 of the PMIx Standard.

18	<code>PMIX_UNDEF</code>	Undefined
19	<code>PMIX_BOOL</code>	Boolean (converted to/from native <code>true/false</code> ) ( <code>bool</code> )
20	<code>PMIX_BYTE</code>	A byte of data ( <code>uint8_t</code> )
21	<code>PMIX_STRING</code>	<code>NULL</code> terminated string ( <code>char*</code> )
22	<code>PMIX_SIZE</code>	Size <code>size_t</code>
23	<code>PMIX_PID</code>	Operating process identifier (PID) ( <code>pid_t</code> )
24	<code>PMIX_INT</code>	Integer ( <code>int</code> )
25	<code>PMIX_INT8</code>	8-byte integer ( <code>int8_t</code> )
26	<code>PMIX_INT16</code>	16-byte integer ( <code>int16_t</code> )
27	<code>PMIX_INT32</code>	32-byte integer ( <code>int32_t</code> )
28	<code>PMIX_INT64</code>	64-byte integer ( <code>int64_t</code> )
29	<code>PMIX_UINT</code>	Unsigned integer ( <code>unsigned int</code> )
30	<code>PMIX_UINT8</code>	Unsigned 8-byte integer ( <code>uint8_t</code> )

1        **PMIX\_UINT16**    Unsigned 16-byte integer (**uint16\_t**)  
 2        **PMIX\_UINT32**    Unsigned 32-byte integer (**uint32\_t**)  
 3        **PMIX\_UINT64**    Unsigned 64-byte integer (**uint64\_t**)  
 4        **PMIX\_FLOAT**     Float (**float**)  
 5        **PMIX\_DOUBLE**    Double (**double**)  
 6        **PMIX\_TIMEVAL**    Time value (**struct timeval**)  
 7        **PMIX\_TIME**     Time (**time\_t**)  
 8        **PMIX\_VALUE**     Value (**pmix\_value\_t**)  
 9        **PMIX\_PROC**     Process (**pmix\_proc\_t**)  
 10       **PMIX\_APP**     Application context  
 11       **PMIX\_INFO**     Info object  
 12       **PMIX\_PDATA**    Pointer to data  
 13       **PMIX\_BUFFER**    Buffer  
 14       **PMIX\_BYTE\_OBJECT**    Byte object (**pmix\_byte\_object\_t**)  
 15       **PMIX\_KVAL**     Key/value pair  
 16       **PMIX\_MODEX (Deprecated in PMIx 2.0)**    Modex  
 17       **PMIX\_PERSIST**    Persistence (**pmix\_persistence\_t**)  
 18       **PMIX\_INFO\_ARRAY (Deprecated in PMIx 2.0)**    Info array

### 19    3.3.6.2    **PMIx v2 Data Types**

20        The following types were introduced in version 2 of the PMIx Standard.

21        **PMIX\_STATUS**    Status (**pmix\_status\_t**)  
 22        **PMIX\_POINTER**    Pointer (**void\***)  
 23        **PMIX\_SCOPE**     Scope (**pmix\_scope\_t**)  
 24        **PMIX\_DATA\_RANGE**    Data range (**pmix\_data\_range\_t**)  
 25        **PMIX\_COMMAND**    Command  
 26        **PMIX\_INFO\_DIRECTIVES**    Info directives  
 27        **PMIX\_DATA\_TYPE**    Data type  
 28        **PMIX\_PROC\_STATE**    Process state (**pmix\_proc\_state\_t**)  
 29        **PMIX\_PROC\_INFO**    Process info (**pmix\_proc\_info\_t**)  
 30        **PMIX\_DATA\_ARRAY**    Data array (**pmix\_data\_array\_t**)  
 31        **PMIX\_PROC\_RANK**    Process rank (**pmix\_rank\_t**)  
 32        **PMIX\_QUERY**     Query  
 33        **PMIX\_COMPRESSED\_STRING**    Compressed string (with zlib)  
 34        **PMIX\_ALLOC\_DIRECTIVE**    Allocation directive (**pmix\_alloc\_directive\_t**)  
 35        **PMIX\_DATA\_TYPE\_MAX**    A boundary for implementers above which they can add their own  
 36        data types.

## 1 3.4 Reserved attributes

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

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

8 Applications or associated libraries (e.g., MPI) may choose to define additional attributes. The  
9 attributes defined in this section are of the system and job as opposed to the attributes that the  
10 application (or associated libraries) might choose to expose. Due to this extensibility the  
11 `PMIx_Get` API will return `PMIX_ERR_NOT_FOUND` if the provided *key* cannot be found.

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

15 **PMIX\_ATTR\_UNDEF** `NULL` (`NULL`)

16 Constant representing an undefined attribute.

### 17 3.4.1 Initialization attributes

18 These attributes are defined to assist the caller with initialization.

19 **PMIX\_EVENT\_BASE** `"pmix.evbase"` (`struct event_base *`)

20 Pointer to libevent<sup>1</sup> `event_base` to use in place of the internal progress thread.

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

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

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

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

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

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

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

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

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

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

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

---

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

1 Registration is for the namespace only. Do not copy job data.  
 2 **PMIX\_SERVER\_ENABLE\_MONITORING** "pmix.srv.monitor" (bool)  
 3 Enable PMIx internal monitoring by the PMIx server.  
 4 **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (char\*)  
 5 Name of the namespace to use for this PMIx server.  
 6 **PMIX\_SERVER\_RANK** "pmix.srv.rank" (pmix\_rank\_t)  
 7 Rank of this PMIx server

## 8 3.4.2 Tool-related attributes

9 These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server.

10 **PMIX\_TOOL\_NAMESPACE** "pmix.tool.namespace" (char\*)  
 11 Name of the namespace to use for this tool.  
 12 **PMIX\_TOOL\_RANK** "pmix.tool.rank" (uint32\_t)  
 13 Rank of this tool.  
 14 **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (pid\_t)  
 15 PID of the target PMIx server for a tool.  
 16 **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (bool)  
 17 The requestor requires that a connection be made only to a local, system-level PMIx server.  
 18 **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (bool)  
 19 Preferentially, look for a system-level PMIx server first.  
 20 **PMIX\_SERVER\_URI** "pmix.srvr.uri" (char\*)  
 21 uniform resource identifier (URI) of the PMIx server to be contacted.  
 22 **PMIX\_SERVER\_HOSTNAME** "pmix.srvr.host" (char\*)  
 23 Host where target PMIx server is located.  
 24 **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)  
 25 Maximum number of times to try to connect to PMIx server.  
 26 **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)  
 27 Time in seconds between connection attempts to a PMIx server.  
 28 **PMIX\_TOOL\_DO\_NOT\_CONNECT** "pmix.tool.nocon" (bool)  
 29 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

## 30 3.4.3 Identification attributes

31 These attributes are defined to identify a process and it's associated PMIx-enabled library.

32 **PMIX\_USERID** "pmix.euid" (uint32\_t)  
 33 Effective user id.  
 34 **PMIX\_GRPID** "pmix.egid" (uint32\_t)  
 35 Effective group id.  
 36 **PMIX\_DSTPATH** "pmix.dstpath" (char\*)  
 37 Path to shared memory data storage (dstore) files.

1 **PMIX\_VERSION\_INFO** "pmix.version" (char\*)  
 2     PMIx version of contractor.

3 **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)  
 4     Programming model being initialized (e.g., "MPI" or "OpenMP")

5 **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)  
 6     Programming model implementation ID (e.g., "OpenMPI" or "MPICH")

7 **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (char\*)  
 8     Programming model version string (e.g., "2.1.1")

9 **PMIX\_THREADING\_MODEL** "pmix.threads" (char\*)  
 10     Threading model used (e.g., "pthreads")

11 **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)  
 12     The requesting process is a PMIx tool.

13 **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)  
 14     The requesting process is a PMIx client.

### 15 3.4.4 UNIX socket rendezvous socket attributes

16     These attributes are used to describe a UNIX socket for rendezvous with the local RM.

17 **PMIX\_USOCK\_DISABLE** "pmix.usock.disable" (bool)  
 18     Disable legacy UNIX socket (usock) support

19 **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)  
 20     POSIX *mode\_t* (9 bits valid)

21 **PMIX\_SINGLE\_LISTENER** "pmix.sing.listnr" (bool)  
 22     Use only one rendezvous socket, letting priorities and/or environment parameters select the  
 23     active transport.

### 24 3.4.5 TCP connection attributes

25     These attributes are used to describe a TCP socket for rendezvous with the local RM.

26 **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)  
 27     If provided, directs that the TCP URI be reported and indicates the desired method of  
 28     reporting: '-' for stdout, '+' for stderr, or filename.

29 **PMIX\_TCP\_URI** "pmix.tcp.uri" (char\*)  
 30     The URI of the PMIx server to connect to, or a file name containing it in the form of  
 31     **file:<name of file containing it>**.

32 **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)  
 33     Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to  
 34     include when establishing the TCP connection.

35 **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)  
 36     Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
 37     TCP connection.

1 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)  
2 The IPv4 port to be used.  
3 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)  
4 The IPv6 port to be used.  
5 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
6 Set to **true** to disable IPv4 family of addresses.  
7 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
8 Set to **true** to disable IPv6 family of addresses.

### 9 3.4.6 Global Data Storage (GDS) attributes

10 These attributes are used to define the behavior of the GDS used to manage key/value pairs.

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

### 13 3.4.7 General process-level attributes

14 These attributes are used to define process attributes.

15 **PMIX\_CPuset** "pmix.cpuset" (char\*)  
16 hwloc<sup>2</sup> bitmap to be applied to the process upon launch.  
17 **PMIX\_CREDENTIAL** "pmix.cred" (char\*)  
18 Security credential assigned to the process.  
19 **PMIX\_SPawnED** "pmix.spawned" (bool)  
20 **true** if this process resulted from a call to **PMIx\_Spawn**.  
21 **PMIX\_ARCH** "pmix.arch" (uint32\_t)  
22 Architecture flag.

### 23 3.4.8 Scratch directory attributes

24 These attributes are used to define an application scratch directory.

25 **PMIX\_TMPDIR** "pmix.tmpdir" (char\*)  
26 Full path to the top-level temporary directory assigned to the session.  
27 **PMIX\_NSdir** "pmix.nsdire" (char\*)  
28 Full path to the temporary directory assigned to the namespace, under **PMIX\_TMPDIR**.  
29 **PMIX\_PROCDIR** "pmix.pdire" (char\*)  
30 Full path to the subdirectory under **PMIX\_NSdir** assigned to the process.  
31 **PMIX\_Tdir\_RMcLEAN** "pmix.tdire.rmclean" (bool)  
32 Resource Manager will clean session directories

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

## 1 3.4.9 Relative Rank Descriptive Attributes

2 These attributes are used to describe information about relative ranks as assigned by the RM.

3 **PMIX\_PROCID** "pmix.procid" (pmix\_proc\_t)

4 Process identifier

5 **PMIX\_NAMESPACE** "pmix.namespace" (char\*)

6 Namespace of the job.

7 **PMIX\_JOBID** "pmix.jobid" (char\*)

8 Job identifier assigned by the scheduler.

9 **PMIX\_APPNUM** "pmix.appnum" (uint32\_t)

10 Application number within the job.

11 **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)

12 Process rank within the job.

13 **PMIX\_GLOBAL\_RANK** "pmix.grank" (pmix\_rank\_t)

14 Process rank spanning across all jobs in this session.

15 **PMIX\_APP\_RANK** "pmix.apprank" (pmix\_rank\_t)

16 Process rank within this application.

17 **PMIX\_NPROC\_OFFSET** "pmix.offset" (pmix\_rank\_t)

18 Starting global rank of this job.

19 **PMIX\_LOCAL\_RANK** "pmix.lrank" (uint16\_t)

20 Local rank on this node within this job.

21 **PMIX\_NODE\_RANK** "pmix.nrank" (uint16\_t)

22 Process rank on this node spanning all jobs.

23 **PMIX\_LOCALLDR** "pmix.lldr" (pmix\_rank\_t)

24 Lowest rank on this node within this job.

25 **PMIX\_APPLDR** "pmix.aldr" (pmix\_rank\_t)

26 Lowest rank in this application within this job.

27 **PMIX\_PROC\_PID** "pmix.ppid" (pid\_t)

28 PID of specified process.

29 **PMIX\_SESSION\_ID** "pmix.session.id" (uint32\_t)

30 Session identifier.

31 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

32 Comma-delimited list of nodes running processes for the specified namespace.

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

34 Comma-delimited list of all nodes in this allocation regardless of whether or not they  
35 currently host processes.

36 **PMIX\_HOSTNAME** "pmix.hname" (char\*)

37 Name of the host where the specified process is running.

38 **PMIX\_NODEID** "pmix.nodeid" (uint32\_t)

39 Node identifier where the specified process is located, expressed as the node's index  
40 (beginning at zero) in an array of nodes comprising the users allocation

41 **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)



1 Comma-delimited list of ranks on this node within the specified namespace.

2 **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (pmix\_proc\_t array)

3 Array of pmix\_proc\_t of processes on the specified node.

4 **PMIX\_LOCAL\_CPUSSETS** "pmix.lcpus" (char\*)

5 Colon-delimited cpusets of local peers within the specified namespace.

6 **PMIX\_PROC\_URI** "pmix.puri" (char\*)

7 URI containing contact information for a given process.

8 **PMIX\_LOCALITY** "pmix.loc" (uint16\_t)

9 Relative locality of two processes.

10 **PMIX\_PARENT\_ID** "pmix.parent" (pmix\_proc\_t)

11 Process identifier of the parent process of the calling process.

## 12 3.4.10 Size information attributes

13 These attributes are used to describe the size of various dimensions of the PMIx universe.

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

15 Number of processes in this namespace.

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

17 Number of processes in this job.

18 **PMIX\_JOB\_NUM\_APPS** "pmix.job.napps" (uint32\_t)

19 Number of applications in this job.

20 **PMIX\_APP\_SIZE** "pmix.app.size" (uint32\_t)

21 Number of processes in this application.

22 **PMIX\_LOCAL\_SIZE** "pmix.local.size" (uint32\_t)

23 Number of processes in this job on this node.

24 **PMIX\_NODE\_SIZE** "pmix.node.size" (uint32\_t)

25 Number of processes across all jobs on this node.

26 **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)

27 Maximum number of processes for this job.

28 **PMIX\_NUM\_NODES** "pmix.num.nodes" (uint32\_t)

29 Number of nodes in this namespace.

## 30 3.4.11 Memory information attributes

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

32 **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (uint64\_t)

33 Total available physical memory on this node.

34 **PMIX\_DAEMON\_MEMORY** "pmix.dmn.mem" (float)

35 Megabytes of memory currently used by the RM daemon.

36 **PMIX\_CLIENT\_AVG\_MEMORY** "pmix.cl.mem.avg" (float)

37 Average Megabytes of memory used by client processes.

## 1 3.4.12 Topology information attributes

2 These attributes are used to describe topology information in the PMIx universe.

3 **PMIX\_NET\_TOPO** "pmix.ntopo" (char\*)

4 eXtensible Markup Language (XML) representation of the network topology.

5 **PMIX\_LOCAL\_TOPO** "pmix.ltopo" (char\*)

6 XML representation of local node topology.

7 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

8 Comma-delimited list of nodes running processes for this job.

9 **PMIX\_TOPOLOGY** "pmix.topo" (hwloc\_topology\_t)

10 Pointer to the PMIx client's internal hwloc topology object.

11 **PMIX\_TOPOLOGY\_SIGNATURE** "pmix.toposig" (char\*)

12 Topology signature string.

13 **PMIX\_LOCALITY\_STRING** "pmix.locstr" (char\*)

14 String describing a process's bound location. The string is of the form:

15 **NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s**

16 Where the %s is replaced with an integer index or inclusive range for hwloc. **NM** identifies  
17 the numa node(s). **SK** identifies the socket(s). **L3** identifies the L3 cache(s). **L2** identifies the  
18 L2 cache(s). **L1** identifies the L1 cache(s). **CR** identifies the cores(s). **HT** identifies the  
19 hardware thread(s). If your architecture does not have the specified hardware designation  
20 then it can be omitted from the signature.

21 For example: **NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39**.

22 This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches  
23 0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.

24 **PMIX\_HWLOC\_SHMEM\_ADDR** "pmix.hwlocaddr" (size\_t)

25 Address of the hwloc shared memory segment.

26 **PMIX\_HWLOC\_SHMEM\_SIZE** "pmix.hwlocsize" (size\_t)

27 Size of the hwloc shared memory segment.

28 **PMIX\_HWLOC\_SHMEM\_FILE** "pmix.hwlocfile" (char\*)

29 Path to the hwloc shared memory file.

30 **PMIX\_HWLOC\_XML\_V1** "pmix.hwlocxml1" (char\*)

31 XML representation of local topology using hwloc's v1.x format.

32 **PMIX\_HWLOC\_XML\_V2** "pmix.hwlocxml2" (char\*)

33 XML representation of local topology using hwloc's v2.x format.

## 34 3.4.13 Request-related attributes

35 These attributes are used to influence the behavior of various PMIx operations.

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

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

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

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

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

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

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

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

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

11 Comma-delimited list of algorithms to use for the collective operation.

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

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

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

15 Notify the parent process upon termination of child job.

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

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

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

19 Value for calls to [PMIx\\_Publish](#) .

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

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

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

23 Look only in the client’s local data store for the requested value - do not request data from  
24 the PMIx server if not found.

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

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

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

30 Status to be returned upon job termination.

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

32 Process state

### 33 3.4.14 Server-to-PMIx library attributes

34 Attributes used by the host environment to pass data to its PMIx server library. The data will then  
35 be parsed and provided to the local PMIx clients.

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

37 Registration is for this namespace only, do not copy job data.

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

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

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

1 Regular expression of nodes containing processes for this job.  
 2 **PMIX\_PROC\_MAP** "pmix.pmap" (char\*)  
 3 Regular expression describing processes on each node within this job.  
 4 **PMIX\_ANL\_MAP** "pmix.anlmap" (char\*)  
 5 Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.  
 6 **PMIX\_APP\_MAP\_TYPE** "pmix.apmap.type" (char\*)  
 7 Type of mapping used to layout the application (e.g., *cyclic*).  
 8 **PMIX\_APP\_MAP\_REGEX** "pmix.apmap.regex" (char\*)  
 9 Regular expression describing the result of the process mapping.

### 10 3.4.15 Server-to-Client attributes

11 Attributes used internally to communicate data from the PMIx server to the PMIx client.  
 12 **PMIX\_PROC\_BLOB** "pmix.pblob" (pmix\_byte\_object\_t)  
 13 Packed blob of process data.  
 14 **PMIX\_MAP\_BLOB** "pmix.mblob" (pmix\_byte\_object\_t)  
 15 Packed blob of process location.

### 16 3.4.16 Event handler registration and notification attributes

17 Attributes to support event registration and notification.

▼ Advice to users ▲

18 The event handler subsystem defined in the PMIx *ad hoc* version 1 Standard was completely  
 19 overhauled in version 2 to resolve design flaws. Deprecated attributes shown below were therefore  
 20 removed in the version 2 Standard.

▲

21 **PMIX\_ERROR\_NAME** "pmix.errname" (pmix\_status\_t)  
 22 Specific error to be notified  
 23 **PMIX\_ERROR\_GROUP\_COMM** "pmix.errgroup.comm" (bool)  
 24 Set true to get comm errors notification  
 25 **PMIX\_ERROR\_GROUP\_ABORT** "pmix.errgroup.abort" (bool)  
 26 Set true to get abort errors notification  
 27 **PMIX\_ERROR\_GROUP\_MIGRATE** "pmix.errgroup.migrate" (bool)  
 28 Set true to get migrate errors notification  
 29 **PMIX\_ERROR\_GROUP\_RESOURCE** "pmix.errgroup.resource" (bool)  
 30 Set true to get resource errors notification  
 31 **PMIX\_ERROR\_GROUP\_SPAWN** "pmix.errgroup.spawn" (bool)  
 32 Set true to get spawn errors notification  
 33 **PMIX\_ERROR\_GROUP\_NODE** "pmix.errgroup.node" (bool)

1           Set true to get node status notification

2   **PMIX\_ERROR\_GROUP\_LOCAL** "pmix.errgroup.local" (bool)

3           Set true to get local errors notification

4   **PMIX\_ERROR\_GROUP\_GENERAL** "pmix.errgroup.gen" (bool)

5           Set true to get notified of generic errors

6   **PMIX\_ERROR\_HANDLER\_ID** "pmix.errhandler.id" (int)

7           Errhandler reference id of notification being reported

8   **PMIX\_EVENT\_HDLR\_NAME** "pmix.evname" (char\*)

9           String name identifying this handler.

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

11           Invoke this event handler before any other handlers.

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

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

14   **PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY** "pmix.evfirstcat" (bool)

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

16   **PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY** "pmix.evlastcat" (bool)

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

18   **PMIX\_EVENT\_HDLR\_BEFORE** "pmix.evbefore" (char\*)

19           Put this event handler immediately before the one specified in the (char\*) value.

20   **PMIX\_EVENT\_HDLR\_AFTER** "pmix.evafter" (char\*)

21           Put this event handler immediately after the one specified in the (char\*) value.

22   **PMIX\_EVENT\_HDLR\_PREPEND** "pmix.evprepend" (bool)

23           Prepend this handler to the precedence list within its category.

24   **PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (bool)

25           Append this handler to the precedence list within its category.

26   **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrange" (pmix\_data\_array\_t\*)

27           Array of **pmix\_proc\_t** defining range of event notification.

28   **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (pmix\_proc\_t)

29           The single process that was affected.

30   **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaffected" (pmix\_data\_array\_t\*)

31           Array of **pmix\_proc\_t** defining affected processes.

32   **PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnondef" (bool)

33           Event is not to be delivered to default event handlers.

34   **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (void \*)

35           Object to be returned whenever the registered callback function **cbfunc** is invoked. The

36           object will *only* be returned to the process that registered it.

37   **PMIX\_EVENT\_DO\_NOT\_CACHE** "pmix.evnocache" (bool)

38           Instruct the PMIx server not to cache the event.

39   **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.avsilentterm" (bool)

40           Do not generate an event when this job normally terminates.

## 1 3.4.17 Fault tolerance attributes

2 Attributes to support fault tolerance behaviors.

3 **PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (bool)

4 The RM intends to terminate this session.

5 **PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (bool)

6 The RM intends to terminate this job.

7 **PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (bool)

8 The RM intends to terminate all processes on this node.

9 **PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (bool)

10 The RM intends to terminate just this process.

11 **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evtimeout" (int)

12 The time in seconds before the RM will execute error response.

13 **PMIX\_EVENT\_NO\_TERMINATION** "pmix.evnoterm" (bool)

14 Indicates that the handler has satisfactorily handled the event and believes termination of the  
15 application is not required.

16 **PMIX\_EVENT\_WANT\_TERMINATION** "pmix.evterm" (bool)

17 Indicates that the handler has determined that the application should be terminated

## 18 3.4.18 Spawn attributes

19 Attributes used to describe **PMIx\_Spawn** behavior.

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

21 Name of personality to use.

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

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

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

25 Hostfile to use for spawned processes.

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

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

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

29 Hostfile listing hosts to add to existing allocation.

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

31 Prefix to use for starting spawned processes.

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

33 Working directory for spawned processes.

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

35 Mapping mechanism to use for placing spawned processes.

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

37 Display process mapping upon spawn.

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

1           Number of processes to spawn on each identified resource.

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

3           Process mapping policy.

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

5           Process ranking policy.

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

7           Process binding policy.

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

9           Preload binaries onto nodes.

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

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

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

13           Spawned processes will not call **PMIx\_Init** .

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

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

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

17           Forward this process's **stdin** to the designated process.

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

19           Forward **stdout** from spawned processes to this process.

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

21           Forward **stderr** from spawned processes to this process.

22    **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

23           Spawned application consists of debugger daemons.

24    **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)

25           Designated application is to be spawned as a disconnected job. Meaning that it is not part of

26           the "comm\_world" of the parent process.

27    **PMIX\_SET\_SESSION\_CWD** "pmix.ssncwd" (bool)

28           Set the application's current working directory to the session working directory assigned by

29           the RM.

30    **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)

31           Tag application output with the identity of the source process.

32    **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)

33           Timestamp output from applications.

34    **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)

35           Merge **stdout** and **stderr** streams from application processes.

36    **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)

37           Output application output to the specified file.

38    **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)

39           Mark the **argv** with the rank of the process.

40    **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)

41           Number of cpus to assign to each rank.

42    **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)

43           Do not place processes on the head node.

1 **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
 2 Do not oversubscribe the cpus.  
 3 **PMIX\_REPORT\_BINDINGS** "pmix.repbinding" (bool)  
 4 Report bindings of the individual processes.  
 5 **PMIX\_CPU\_LIST** "pmix.cpulist" (char\*)  
 6 List of cpus to use for this job.  
 7 **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)  
 8 Application supports recoverable operations.  
 9 **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)  
 10 Application is continuous, all failed processes should be immediately restarted.  
 11 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)  
 12 Maximum number of times to restart a job.

### 13 3.4.19 Query attributes

14 Attributes used to describe `PMIx_Query_info_nb` behavior.

15 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (char\*)  
 16 Request a comma-delimited list of active namespaces.  
 17 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (pmix\_status\_t)  
 18 Status of a specified, currently executing job.  
 19 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (char\*)  
 20 Request a comma-delimited list of scheduler queues.  
 21 **PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (TBD)  
 22 Status of a specified scheduler queue.  
 23 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (char\*)  
 24 Input namespace of the job whose information is being requested returns (  
 25 `pmix_data_array_t`) an array of `pmix_proc_info_t`.  
 26 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)  
 27 Input namespace of the job whose information is being requested returns (  
 28 `pmix_data_array_t`) an array of `pmix_proc_info_t` for processes in job on same  
 29 node.  
 30 **PMIX\_QUERY\_AUTHORIZATIONS** "pmix.qry.auths" (bool)  
 31 Return operations the PMIx tool is authorized to perform.  
 32 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)  
 33 Return a comma-delimited list of supported spawn attributes.  
 34 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)  
 35 Return a comma-delimited list of supported debug attributes.  
 36 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)  
 37 Return information on memory usage for the processes indicated in the qualifiers.  
 38 **PMIX\_QUERY\_LOCAL\_ONLY** "pmix.qry.local" (bool)  
 39 Constrain the query to local information only.  
 40 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)



1 Report average values.  
2 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)  
3 Report minimum and maximum values.  
4 **PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)  
5 String identifier of the allocation whose status is being requested.  
6 **PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)  
7 Query number of seconds (`uint32_t`) remaining in allocation for the specified namespace.

## 8 3.4.20 Log attributes

9 Attributes used to describe `PMIx_Log_nb` behavior.  
10 **PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)  
11 Log string to `stderr`.  
12 **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)  
13 Log string to `stdout`.  
14 **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (char\*)  
15 Log data to syslog. Defaults to `ERROR` priority.  
16 **PMIX\_LOG\_MSG** "pmix.log.msg" (`pmix_byte_object_t`)  
17 Message blob to be sent somewhere.  
18 **PMIX\_LOG\_EMAIL** "pmix.log.email" (`pmix_data_array_t`)  
19 Log via email based on `pmix_info_t` containing directives.  
20 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (char\*)  
21 Comma-delimited list of email addresses that are to receive the message.  
22 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (char\*)  
23 Subject line for email.  
24 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)  
25 Message to be included in email.

## 26 3.4.21 Debugger attributes

27 Attributes used to assist debuggers.  
28 **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (bool)  
29 Job is being spawned under debugger. The processes are instructed to pause on start.  
30 **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (bool)  
31 Instruct job to stop processes during `PMIx_Init`.  
32 **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** "pmix.dbg.notify" (bool)  
33 Block at desired point until receiving debugger release notification.  
34 **PMIX\_DEBUG\_JOB** "pmix.dbg.job" (char\*)  
35 Namespace of the job to be debugged.  
36 **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** "pmix.dbg.waiting" (bool)  
37 Job to be debugged is waiting for a release.

## 1 3.4.22 Resource manager attributes

2 Attributes used to describe the RM.

3 **PMIX\_RM\_NAME** "pmix.rm.name" (char\*)

4 String name of the RM.

5 **PMIX\_RM\_VERSION** "pmix.rm.version" (char\*)

6 RM version string.

## 7 3.4.23 Environment variable attributes

8 Attributes used to adjust environment variables.

9 **PMIX\_SET\_ENVAR** "pmix.set.envar" (char\*)

10 String "key=value" value shall be put into the environment.

11 **PMIX\_UNSET\_ENVAR** "pmix.unset.envar" (char\*)

12 Unset the environment variable specified in the string.

## 13 3.4.24 Job Allocation attributes

14 Attributes used to describe the job allocation.

15 **PMIX\_ALLOC\_ID** "pmix.alloc.id" (char\*)

16 Provide a string identifier for this allocation request which can later be used to query status  
17 of the request.

18 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (uint64\_t)

19 The number of nodes.

20 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)

21 Regular expression of the specific nodes.

22 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

23 Number of cpus.

24 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)

25 Regular expression of the number of cpus for each node.

26 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)

27 Regular expression of the specific cpus indicating the cpus involved.

28 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)

29 Number of Megabytes.

30 **PMIX\_ALLOC\_NETWORK** "pmix.alloc.net" (array)

31 Array of `pmix_info_t` describing requested network resources. If not given as part of an  
32 `pmix_info_t` struct that identifies the involved nodes, then the description will be  
33 applied across all nodes in the requestor's allocation.

34 **PMIX\_ALLOC\_NETWORK\_ID** "pmix.alloc.netid" (char\*)

35 Name of the network.

36 **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)

1           Mbits/sec.  
2       **PMIX\_ALLOC\_NETWORK\_QOS** "pmix.alloc.netqos" (char\*)  
3           Quality of service level.  
4       **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)  
5           Time in seconds.

## 6 3.4.25 Job control attributes

7       Attributes used to request control operations on an executing application.

8       **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (char\*)  
9           Provide a string identifier for this request.

10       **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (bool)  
11           Pause the specified processes.

12       **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (bool)  
13           Resume ("un-pause") the specified processes.

14       **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)  
15           Cancel the specified request (**NULL** implies cancel all requests from this requestor).

16       **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (bool)  
17           Forcibly terminate the specified processes and cleanup.

18       **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)  
19           Restart the specified processes using the given checkpoint ID.

20       **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)  
21           Checkpoint the specified processes and assign the given ID to it.

22       **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)  
23           Use event notification to trigger a process checkpoint.

24       **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)  
25           Use the given signal to trigger a process checkpoint.

26       **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)  
27           Time in seconds to wait for a checkpoint to complete.

28       **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
29       "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)  
30           Array of **pmix\_info\_t** declaring each method and value supported by this application.

31       **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)  
32           Send given signal to specified processes.

33       **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)  
34           Regular expression identifying nodes that are to be provisioned.

35       **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvning" (char\*)  
36           Name of the image that is to be provisioned.

37       **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)  
38           Indicate that the job can be pre-empted.

39       **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)  
40           Politely terminate the specified processes.

## 1 3.4.26 Monitoring attributes

2 Attributes used to control monitoring of an executing application.

3 **PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)

4 Provide a string identifier for this request.

5 **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)

6 Identifier to be canceled (NULL means cancel all monitoring for this process).

7 **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)

8 The application desires to control the response to a monitoring event.

9 **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)

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

11 **PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (void)

12 Send heartbeat to local PMIx server.

13 **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)

14 Time in seconds before declaring heartbeat missed.

15 **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)

16 Number of heartbeats that can be missed before generating the event.

17 **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)

18 Register to monitor file for signs of life.

19 **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)

20 Monitor size of given file is growing to determine if the application is running.

21 **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (char\*)

22 Monitor time since last access of given file to determine if the application is running.

23 **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)

24 Monitor time since last modified of given file to determine if the application is running.

25 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)

26 Time in seconds between checking the file.

27 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)

28 Number of file checks that can be missed before generating the event.

## 29 3.5 Callback Functions

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

## 1 3.5.1 Release Callback Function

### 2 Summary

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

### 6 Format

*PMIx v1.0*

```
▼ C ▼  
7 typedef void (*pmix_release_cbfunc_t)  
8     (void *cbdata)  
▲ C ▲
```

### 9 INOUT `cbdata`

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

### 11 Description

12 Since the data is “owned” by the host server, provide a callback function to notify the host server  
13 that we are done with the data so it can be released.

## 14 3.5.2 Modex Callback Function

### 15 Summary

16 The `pmix_modex_cbfunc_t` is used by the `pmix_server_fence_fn_t` and  
17 `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex BCX data.

*PMIx v1.0*

```
▼ C ▼  
18 typedef void (*pmix_modex_cbfunc_t)  
19     (pmix_status_t status,  
20      const char *data, size_t ndata,  
21      void *cbdata,  
22      pmix_release_cbfunc_t release_fn,  
23      void *release_cbdata)  
▲ C ▲
```

### 24 IN `status`

25 Status associated with the operation (handle)

### 26 IN `data`

27 Data to be passed (pointer)

1       **IN**   **ndata**  
 2           size of the data (**size\_t**)  
 3       **IN**   **cbdata**  
 4           Callback data passed to original API call (memory reference)  
 5       **IN**   **release\_fn**  
 6           Callback for releasing *data* (function pointer)  
 7       **IN**   **release\_cbdata**  
 8           Pointer to be passed to *release\_fn* (memory reference)

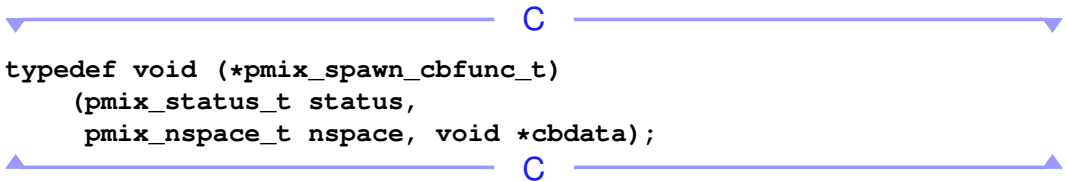
9       **Description**

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

13   **3.5.3 Spawn Callback Function**

14       **Summary**

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

17       *PMIx v1.0*     **C**

```

17       typedef void (*pmix_spawn_cbfunc_t)
18           (pmix_status_t status,
19            pmix_nspace_t nspace, void *cbdata);

```

20       **IN**   **status**  
 21           Status associated with the operation (handle)  
 22       **IN**   **nspace**  
 23           Namespace string ( `pmix_nspace_t` )  
 24       **IN**   **cbdata**  
 25           Callback data passed to original API call (memory reference)

26       **Description**

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

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

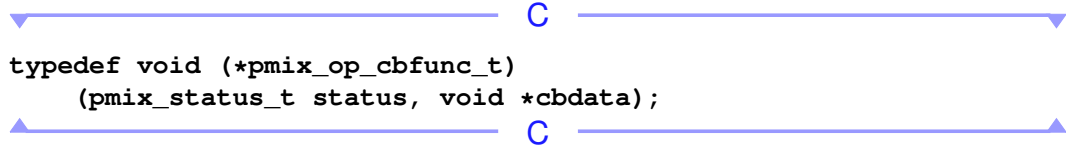
## 1 3.5.4 Op Callback Function

### 2 Summary

3 The [pmix\\_op\\_cbfunc\\_t](#) is used by operations that simply return a status.

*PMIx v1.0*

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



6 **IN status**

7 Status associated with the operation (handle)

8 **IN cbdata**

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

### 10 Description

11 Used by a wide range of PMIx API's including [PMIx\\_Fence\\_nb](#),  
12 [pmix\\_server\\_client\\_connected\\_fn\\_t](#), [PMIx\\_server\\_register\\_namespace](#). This  
13 callback function is used to return a status to an often nonblocking operation.

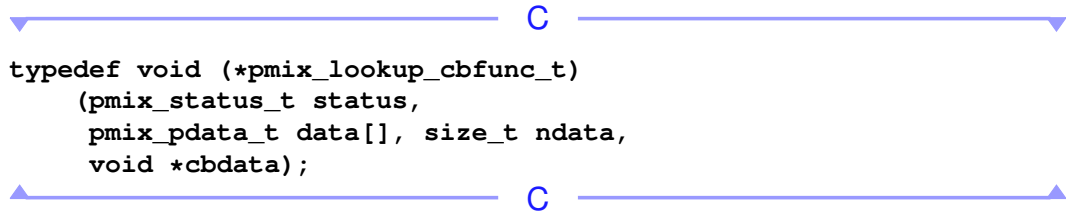
## 14 3.5.5 Lookup Callback Function

### 15 Summary

16 The [pmix\\_lookup\\_cbfunc\\_t](#) is used by [PMIx\\_Lookup\\_nb](#) to return data.

*PMIx v1.0*

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



21 **IN status**

22 Status associated with the operation (handle)

23 **IN data**

24 Array of data returned ([pmix\\_pdata\\_t](#))

25 **IN ndata**

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

27 **IN cbdata**

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

## Description

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

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

## 3.5.6 Value Callback Function

### Summary

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

*PMIx v1.0*

```
typedef void (*pmix_value_cbfunc_t)
(pmix_status_t status,
 pmix_value_t *kv, void *cbdata);
```

#### IN `status`

Status associated with the operation (handle)

#### IN `kv`

Key/value pair representing the data (`pmix_value_t`)

#### IN `cbdata`

Callback data passed to original API call (memory reference)

### Description

A callback function for calls to `PMIx_Get_nb`. The *status* indicates if the requested data was found or not. A pointer to the `pmix_value_t` structure containing the found data is returned. The pointer will be `NULL` if the requested data was not found.

## 3.5.7 Info Callback Function

### Summary

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

*PMIx v2.0*



```

1 typedef void (*pmix_info_cbfunc_t)
2     (pmix_status_t status,
3      pmix_info_t info[], size_t ninfo,
4      void *cbdata,
5      pmix_release_cbfunc_t release_fn,
6      void *release_cbdata);

```

**IN status**  
Status associated with the operation (`pmix_status_t`)

**IN info**  
Array of `pmix_info_t` returned by the operation (pointer)

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

**IN cbdata**  
Callback data passed to original API call (memory reference)

**IN release\_fn**  
Function to be called when done with the *info* data (function pointer)

**IN release\_cbdata**  
Callback data to be passed to *release\_fn* (memory reference)

### Description

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

## 3.5.8 Event Handler Registration Callback Function

The `pmix_evhdlr_reg_cbfunc_t` callback function.

### Advice to users

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

PMIx v2.0

```

1 typedef void (*pmix_evhdlr_reg_cbfunc_t)
2     (pmix_status_t status,
3      size_t evhdlr_ref,
4      void *cbdata)

```

```

5 IN   status
6       Status indicates if the request was successful or not (pmix_status_t)
7 IN   evhdlr_ref
8       Reference assigned to the event handler by PMIx — this reference * must be used to
9       deregister the err handler (size_t)
10 IN   cbdata
11       Callback data passed to original API call (memory reference)

```

## Description

Define a callback function for calls to `PMIx_Register_event_handler`

## 3.5.9 Notification Handler Completion Callback Function

### Summary

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

*PMIx v2.0*

```

18 typedef void (*pmix_event_notification_cbfunc_fn_t)
19     (pmix_status_t status,
20      pmix_info_t *results, size_t nresults,
21      pmix_op_cbfunc_t cbfunc, void *thiscbdata,
22      void *notification_cbdata);

```

```

23 IN   status
24       Status returned by the event handler's operation (pmix_status_t)
25 IN   results
26       Results from this event handler's operation on the event (pmix_info_t)
27 IN   nresults
28       Number of elements in the results array (size_t)
29 IN   cbfunc
30       pmix_op_cbfunc_t function to be executed when PMIx completes processing the
31       callback (function reference)

```

1       **IN**   **thiscbdata**  
2            Callback data that was passed in to the handler (memory reference)  
3       **IN**   **cbdata**  
4            Callback data to be returned when PMIx executes cbfunc (memory reference)

## 5       **Description**

6       Define a callback by which an event handler can notify the PMIx library that it has completed its  
7       response to the notification. The handler is *required* to execute this callback so the library can  
8       determine if additional handlers need to be called. The handler shall return  
9       **PMIX\_ERR\_EVENT\_COMPLETE** if no further action is required. The return status of each event  
10       handler and any returned **pmix\_info\_t** structures will be added to the *results* array of  
11       **pmix\_info\_t** passed to any subsequent event handlers to help guide their operation.  
12       If non-NULL, the provided callback function will be called to allow the event handler to release the  
13       provided info array and execute any other required cleanup operations.

## 14    **3.5.10 Notification Function**

### 15    **Summary**

16    The **pmix\_notification\_fn\_t** is called by PMIx to deliver notification of an event.

#### ▼ Advice to users ▼

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



PMIx v2.0

C

```
20    typedef void (*pmix_notification_fn_t)  
21           (size_t evhdlr_registration_id,  
22           pmix_status_t status,  
23           const pmix_proc_t *source,  
24           pmix_info_t info[], size_t ninfo,  
25           pmix_info_t results[], size_t nresults,  
26           pmix_event_notification_cbfunc_fn_t cbfunc,  
27           void *cbdata);
```

1 **IN** `evhdlr_registration_id`  
 2 Registration number of the handler being called (`size_t`)  
 3 **IN** `status`  
 4 Status associated with the operation (`pmix_status_t`)  
 5 **IN** `source`  
 6 Identifier of the process that generated the event (`pmix_proc_t`). If the source is the  
 7 SMS, then the namespace will be empty and the rank will be `PMIX_RANK_UNDEF`  
 8 **IN** `info`  
 9 Information describing the event (`pmix_info_t`). This argument will be `NULL` if no  
 10 additional information was provided by the event generator.  
 11 **IN** `ninfo`  
 12 Number of elements in the info array (`size_t`)  
 13 **IN** `results`  
 14 Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This  
 15 argument will be `NULL` if this is the first handler servicing the event, or if no prior handlers  
 16 provided results.  
 17 **IN** `nresults`  
 18 Number of elements in the results array (`size_t`)  
 19 **IN** `cbfunc`  
 20 `pmix_event_notification_cbfunc_fn_t` callback function to be executed upon  
 21 completion of the handler's operation and prior to handler return (function reference).  
 22 **IN** `cbdata`  
 23 Callback data to be passed to `cbfunc` (memory reference)

## 24 Description

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

## Advice to users

29 Possible uses of the `info` array include:

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

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

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

---

▼ — Advice to PMIx server hosts — ▼

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

### 12 3.5.11 Server Setup Application Callback Function

13 The [PMIx\\_server\\_setup\\_application](#) callback function.

#### 14 Summary

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

1 **Format**

PMIx v2.0

C

```
2 typedef void (*pmix_setup_application_cbfunc_t) (  
3     pmix_status_t status,  
4     pmix_info_t info[], size_t ninfo,  
5     void *provided_cbdata,  
6     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

- 7 **IN status**  
8 returned status of the request ([pmix\\_status\\_t](#))
- 9 **IN info**  
10 Array of info structures (array of handles)
- 11 **IN ninfo**  
12 Number of elements in the *info* array (integer)
- 13 **IN provided\_cbdata**  
14 Data originally passed to call to [PMIx\\_server\\_setup\\_application](#) (memory  
15 reference)
- 16 **IN cbfunc**  
17 [pmix\\_op\\_cbfunc\\_t](#) function to be called when processing completed (function  
18 reference)
- 19 **IN cbdata**  
20 Data to be passed to the *cbfunc* callback function (memory reference)

21 **Description**

22 Define a function to be called by the PMIx server library for return of application-specific setup  
23 data in response to a request from the host RM. The returned *info* array is owned by the PMIx  
24 server library and will be free'd when the provided *cbfunc* is called.

25 **3.5.12 Server Direct Modex Response Callback Function**

26 The [PMIx\\_server\\_dmodex\\_request](#) callback function.

27 **Summary**

28 Provide a function by which the local PMIx server library can return connection and other data  
29 posted by local application processes to the host resource manager.

1 **Format**

*PMIx v1.0*

```

▼ C ▼
2 typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
3     char *data, size_t sz,
4     void *cbdata);
▲ C ▲

```

- 5 **IN status**  
Returned status of the request ([pmix\\_status\\_t](#))
- 6 **IN data**  
Pointer to a data "blob" containing the requested information (handle)
- 7 **IN sz**  
Number of bytes in the *data* blob (integer)
- 8 **IN cbdata**  
Data passed into the initial call to [PMIx\\_server\\_dmodex\\_request](#) (memory reference)

14 **Description**

15 Define a function to be called by the PMIx server library for return of information posted by a local  
 16 application process (via [PMIx\\_Put](#) with subsequent [PMIx\\_Commit](#) ) in response to a request  
 17 from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd  
 18 upon return from the function.

19 **3.5.13 pmix\_connection\_cbfunc\_t**

20 **Summary**

21 Callback function for incoming connection request from a local client

22 **Format**

*PMIx v1.0*

```

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

```

- 25 **IN incoming\_sd**  
(integer)
- 26 **IN cbdata**  
(memory reference)

1 **Description**

2 Callback function for incoming connection requests from local clients - only used by host  
3 environments that wish to directly handle socket connection requests.

4 **3.5.14 pmix\_tool\_connection\_cbfunc\_t**

5 **Summary**

6 Callback function for incoming tool connections.

7 **Format**

8 *PMIx v2.0*

```
▼────────────────────────────────────────── C ───────────────────────────────────────────▼  
8 typedef void (*pmix_tool_connection_cbfunc_t) (  
9             pmix_status_t status,  
10            pmix_proc_t *proc, void *cbdata)  
▲────────────────────────────────────────── C ───────────────────────────────────────────▲
```

- 11 **IN status**  
12 `pmix_status_t` value (handle)
- 13 **IN proc**  
14 `pmix_proc_t` structure containing the identifier assigned to the tool (handle)
- 15 **IN cbdata**  
16 Data to be passed (memory reference)

17 **Description**

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

▼── Advice to PMIx server hosts ───▼

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

▲──▲





22 **3.5.15 Constant String Functions**

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







1 **Summary**  
2 String representation of a [pmix\\_status\\_t](#) .

*PMIx v1.0*

▼  C   
3 **const char\***  
4 **PMIx\_Error\_string**(pmix\_status\_t status);  
▲  C 





5 **Summary**  
6 String representation of a [pmix\\_proc\\_state\\_t](#) .

*PMIx v2.0*

▼  C   
7 **const char\***  
8 **PMIx\_Proc\_state\_string**(pmix\_proc\_state\_t state);  
▲  C 





9 **Summary**  
10 String representation of a [pmix\\_scope\\_t](#) .

*PMIx v2.0*

▼  C   
11 **const char\***  
12 **PMIx\_Scope\_string**(pmix\_scope\_t scope);  
▲  C 





13 **Summary**  
14 String representation of a [pmix\\_persistence\\_t](#) .

*PMIx v2.0*

▼  C   
15 **const char\***  
16 **PMIx\_Persistence\_string**(pmix\_persistence\_t persist);  
▲  C 

17 **Summary**  
18 String representation of a [pmix\\_data\\_range\\_t](#) .

*PMIx v2.0*

▼  C   
19 **const char\***  
20 **PMIx\_Data\_range\_string**(pmix\_data\_range\_t range);  
▲  C 

1       **Summary**

2       String representation of a `pmix_info_directives_t` .

*PMIx v2.0*   ▼————— C —————▼

3       **const char\***

4       **PMIx\_Info\_directives\_string**(`pmix_info_directives_t directives`);

▲————— C —————▲

5       **Summary**

6       String representation of a `pmix_data_type_t` .

*PMIx v2.0*   ▼————— C —————▼

7       **const char\***

8       **PMIx\_Data\_type\_string**(`pmix_data_type_t type`);

▲————— C —————▲

9       **Summary**

10       String representation of a `pmix_alloc_directive_t` .

*PMIx v2.0*   ▼————— C —————▼

11       **const char\***

12       **PMIx\_Alloc\_directive\_string**(`pmix_alloc_directive_t directive`);

▲————— C —————▲

## 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. Note that  
6 a process can only call *one* of the init/finalize functional pairs - e.g., a process that calls the client  
7 initialization function cannot also call the tool or server initialization functions, and must call the  
8 corresponding client finalize.

### Advice to users

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

## 4.1 Query

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

### 4.1.1 PMIx\_Initialized

#### Format

PMIx v1.0

```
int PMIx_Initialized(void)
```

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

#### Rationale

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

1        **Description**

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

4        **4.1.2 PMIx\_Get\_version**

5        **Summary**

6        Get the PMIx version information.

7        **Format**

8        *PMIx v1.0*    ▼ \_\_\_\_\_ C \_\_\_\_\_ ▼  
9        `const char* PMIx_Get_version(void)`  
10        ▲ \_\_\_\_\_ C \_\_\_\_\_ ▲

11       **Description**

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

14       **4.2 Client Initialization and Finalization**

15       Initialization and finalization routines for PMIx clients.

16       ▼ \_\_\_\_\_ **Advice to users** \_\_\_\_\_ ▼

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

20       **4.2.1 PMIx\_Init**

21       **Summary**

22       Initialize the PMIx client library

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34

## Format

PMIx v1.2

```

pmix_status_t
PMIx_Init (pmix_proc_t *proc,
           pmix_info_t info[], size_t ninfo)

```

### INOUT `proc`

proc structure (handle)

### IN `info`

Array of `pmix_info_t` structures (array of handles)

### IN `ninfo`

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

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

## Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

### `PMIX_USOCK_DISABLE` "`pmix.usock.disable`" (`bool`)

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

### `PMIX_SOCKET_MODE` "`pmix.sockmode`" (`uint32_t`)

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

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

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

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

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

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

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

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

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

1 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)  
 2 The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be  
 3 supported for specifying the port to be used.

4 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)  
 5 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be  
 6 supported for specifying the port to be used.

7 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
 8 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
 9 this attribute may be supported for disabling it.

10 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
 11 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
 12 this attribute may be supported for disabling it.

13 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)  
 14 Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

15 **PMIX\_GDS\_MODULE** "pmix.gds.mod" (char\*)  
 16 Comma-delimited string of desired modules. This attribute is specific to the PRI and  
 17 controls only the selection of GDS module for internal use by the process. Module selection  
 18 for interacting with the server is performed dynamically during the connection process.

▲-----▲

## 19 Description

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

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

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

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

---

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

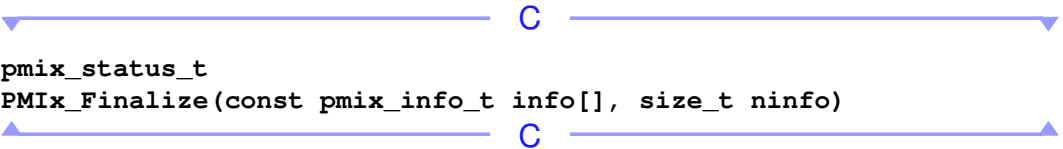

1 Each call to `PMIx_Init` may contain an array of `pmix_info_t` structures passing directives to  
2 the PMIx client library as per the above attributes.  
3 Multiple calls to `PMIx_Init` shall not include conflicting directives. The `PMIx_Init` function  
4 will return an error when directives that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

## 5 4.2.2 `PMIx_Finalize`

### 6 Summary

7 Finalize the PMIx client library.

### 8 Format

*PMIx v1.0*  `C`  
9 `pmix_status_t`  
10 `PMIx_Finalize(const pmix_info_t info[], size_t ninfo)`  
 `C`

11 **IN** `info`  
12     Array of `pmix_info_t` structures (array of handles)  
13 **IN** `ninfo`  
14     Number of element in the `info` array (`size_t`)

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

### Optional Attributes

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

17 **PMIX\_EMBED\_BARRIER** "`pmix.embed.barrier`" (`bool`)  
18     Execute a blocking fence operation before executing the specified operation. By default,  
19     `PMIx_Finalize` does not include an internal barrier operation. This attribute directs  
20     `PMIx_Finalize` to execute a barrier as part of the finalize operation.

### 21 Description

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

## 1 4.3 Tool Initialization and Finalization



2 Initialization and finalization routines for PMIx tools.

### 3 4.3.1 PMIx\_tool\_init

#### 4 Summary

5 Initialize the PMIx library for operating as a tool.

#### 6 Format

7 *PMIx v2.0*  

```
8 pmix_status_t  
9 PMIx_tool_init(pmix_proc_t *proc,  
10                pmix_info_t info[], size_t ninfo)
```

#### 10 INOUT proc

11 `pmix_proc_t` structure (handle)

#### 12 IN info

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

#### 14 IN ninfo

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

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

#### Required Attributes

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

18 **PMIX\_TOOL\_NAMESPACE** "pmix.tool.namespace" (`char*`)

19 Name of the namespace to use for this tool.

20 **PMIX\_TOOL\_RANK** "pmix.tool.rank" (`uint32_t`)

21 Rank of this tool.

22 **PMIX\_TOOL\_DO\_NOT\_CONNECT** "pmix.tool.nocon" (`bool`)

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

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

25 URI of the PMIx server to be contacted.



## Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

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

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

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

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

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

PID of the target PMIx server for a tool.

**PMIX\_TCP\_URI** "pmix.tcp.uri" (char\*)

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

**PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)

Time in seconds between connection attempts to a PMIx server.

**PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)

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

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

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

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

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

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

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

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

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

**PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)

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

**PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)

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

1 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
 2 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
 3 this attribute may be supported for disabling it.

4 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
 5 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
 6 this attribute may be supported for disabling it.

7 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)  
 8 Pointer to libevent<sup>2</sup> **event\_base** to use in place of the internal progress thread.

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

▲-----▲

## 13 Description

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

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

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

---

<sup>2</sup><http://libevent.org/>

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

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

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

## 4.3.2 `PMIx_tool_finalize`

### Summary

Finalize the PMIx library for a tool connection.

### Format

*PMIx v2.0*

```

pmix_status_t
PMIx_tool_finalize(void)

```

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

### Description

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

## 1 4.4 Server Initialization and Finalization

2 Initialization and finalization routines for PMIx servers.

### 3 4.4.1 PMIx\_server\_init

#### 4 Summary

5 Initialize the PMIx server.

#### 6 Format

PMIx v1.0

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

10 **INOUT** module  
11     pmix\_server\_module\_t structure (handle)  
12 **IN** info  
13     Array of pmix\_info\_t structures (array of handles)  
14 **IN** ninfo  
15     Number of elements in the info array (**size\_t**)

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

#### Required Attributes

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

18 **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (**char\***)

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

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

21     Rank of this PMIx server

22 **PMIX\_SERVER\_TMPDIR** "pmix.srv.tmpdir" (**char\***)

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

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

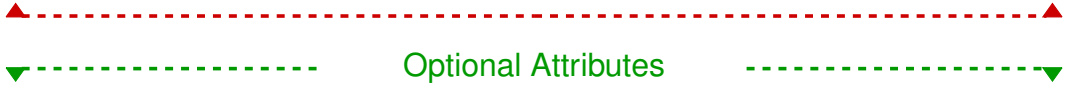
26     Temporary directory for this system, and where a PMIx server that declares itself to be a  
27     system-level server will place a *tool* rendezvous point and contact information.

28 **PMIX\_SERVER\_TOOL\_SUPPORT** "pmix.srv.tool" (**bool**)

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

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

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



### Optional Attributes

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

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

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

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

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

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

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

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

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

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

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

24 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)

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

27 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)

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

30 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)

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

33 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)

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

1 **PMIX\_SERVER\_REMOTE\_CONNECTIONS** "pmix.srvr.remote" (bool)  
 2 Allow connections from remote tools. Forces the PMIx server to not exclusively use  
 3 loopback device. If the library supports connections from remote tools, this attribute may  
 4 be supported for enabling or disabling it.

5 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)  
 6 Pointer to libevent<sup>3</sup> event\_base to use in place of the internal progress thread.

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

## 11 Description

12 Initialize the PMIx server support library, and provide a pointer to a `pmix_server_module_t`  
 13 structure containing the caller's callback functions. The array of `pmix_info_t` structs is used to  
 14 pass additional info that may be required by the server when initializing. For example, it may  
 15 include the `PMIX_SERVER_TOOL_SUPPORT` key, thereby indicating that the daemon is willing  
 16 to accept connection requests from tools.

### Advice to PMIx server hosts

17 Providing a value of `NULL` for the *module* argument is permitted, as is passing an empty *module*  
 18 structure. Doing so indicates that the host environment will not provide support for multi-node  
 19 operations such as `PMIx_Fence`, but does intend to support local clients access to information.

## 20 4.4.2 `PMIx_server_finalize`

### 21 Summary

22 Finalize the PMIx server library.

### 23 Format

PMIx v1.0

24 `pmix_status_t`  
 25 `PMIx_server_finalize(void)`

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

<sup>3</sup><http://libevent.org/>

1  
2  
3

## **Description**

Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.

## CHAPTER 5

# Key/Value Management

---

1 Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of  
2 the PMIx community is to eliminate collective operations, it is recognized that the traditional  
3 method of publishing/exchanging data must be supported until that objective can be met. This  
4 method relies on processes to discover and publish their local information which is collected by the  
5 local PMIx server library. Global exchange of the published information is then executed via a  
6 collective operation performed by the host SMS servers.

## 7 5.1 Setting and Accessing Key/Value Pairs

### 8 5.1.1 PMIx\_Put

#### 9 Summary

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

#### 11 Format

*PMIx v1.0*

```
▼ C ▼  
12 pmix_status_t  
13 PMIx_Put (pmix_scope_t scope,  
14           const pmix_key_t key,  
15           pmix_value_t *val)  
▲ C ▲
```

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

18 **IN key**  
19 key ( `pmix_key_t` )

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

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



## Description

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

The provided *scope* is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The `pmix_scope_t` values are defined in Section 3.2.9 on page 26. Specific implementations may support different scope values, but all implementations must support at least `PMIX_GLOBAL`.

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

### Advice to PMIx library implementers

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

### Advice to users

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

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

## 5.1.2 PMIx\_Get

### Summary

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

1 **Format**

PMIx v1.0

C

```

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

```

C

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

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

Required Attributes

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

- 18 **PMIX\_OPTIONAL** "pmix.optional" (`bool`)  
19 Look only in the client's local data store for the requested value - do not request data from  
20 the PMIx server if not found.
- 21 **PMIX\_IMMEDIATE** "pmix.immediate" (`bool`)  
22 Specified operation should immediately return an error from the PMIx server if the requested  
23 data cannot be found - do not request it from the host RM.
- 24 **PMIX\_DATA\_SCOPE** "pmix.scope" (`pmix_scope_t`)  
25 Scope of the data to be found in a `PMIx_Get` call.

Optional Attributes

26 The following attributes are optional for host environments:

- 27 **PMIX\_TIMEOUT** "pmix.timeout" (`int`)  
28 Time in seconds before the specified operation should time out (0 indicating infinite) in  
29 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent  
30 the target process from ever exposing its data.

## Advice to PMIx library implementers

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

### Description

Retrieve information for the specified *key* as published by the process identified in the given `pmix_proc_t`, returning a pointer to the value in the given address.

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

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

## Advice to users

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

In general, only data posted by a process via `PMIx_Put` needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of `PMIX_RANK_WILDCARD`.

### 5.1.3 PMIx\_Get\_nb

#### Summary

Nonblocking `PMIx_Get` operation.

1 **Format**

PMIx v1.0

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**  
process reference (handle)
- 7
- 8 **IN key**  
key to retrieve (string)
- 9
- 10 **IN info**  
Array of info structures (array of handles)
- 11
- 12 **IN ninfo**  
Number of elements in the *info* array (integer)
- 13
- 14 **IN cbfunc**  
Callback function (function reference)
- 15
- 16 **IN cbdata**  
Data to be passed to the callback function (memory reference)
- 17

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

Required Attributes

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

- 20 **PMIX\_OPTIONAL** "pmix.optional" (bool)  
21 Look only in the client's local data store for the requested value - do not request data from  
22 the PMIx server if not found.
- 23 **PMIX\_IMMEDIATE** "pmix.immediate" (bool)  
24 Specified operation should immediately return an error from the PMIx server if the requested  
25 data cannot be found - do not request it from the host RM.
- 26 **PMIX\_DATA\_SCOPE** "pmix.scope" (pmix\_scope\_t)  
27 Scope of the data to be found in a **PMIx\_Get** call.

Optional Attributes

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

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

## Advice to PMIx library implementers

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

### Description

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

## Advice to users

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

In general, only data posted by a process via `PMIx_Put` needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of `PMIX_RANK_WILDCARD`.

## 5.1.4 `PMIx_Store_internal`

### Summary

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

1 **Format**

*PMIx v1.0*

C

```

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

```

C

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

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

19 **5.2.1 PMIx\_Commit**

20 **Summary**

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

22 **Format**

*PMIx v1.0*

C

```

23 pmix_status_t PMIx_Commit(void)

```

C

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

## Description

This is an asynchronous operation. The PRI 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 - i.e., 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, collecting information posted via `PMIx_Put` as directed.

### Format

*PMIx v1.0*

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.

### Required Attributes

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

**PMIX\_COLLECT\_DATA** "`pmix.collect`" (`bool`)  
Collect data and return it at the end of the operation.

## Optional Attributes

The following attributes are optional for host environments:

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

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

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

Comma-delimited list of algorithms to use for the collective operation.

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

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

## Advice to PMIx library implementers

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

## Description

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

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

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

## 5.2.3 PMIx\_Fence\_nb

### Summary

Execute a nonblocking **PMIx\_Fence** across the processes identified in the specified array of processes, collecting information posted via **PMIx\_Put** as directed.



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30

## Format

PMIx v1.0

```

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

```

- IN procs**  
Array of **pmix\_proc\_t** structures (array of handles)
- IN nprocs**  
Number of 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)
- IN cbfunc**  
Callback function (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.

### Required Attributes

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

**PMIX\_COLLECT\_DATA** "pmix.collect" (**bool**)  
Collect data and return it at the end of the operation.

### Optional Attributes

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

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

**PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (**char\***)  
Comma-delimited list of algorithms to use for the collective operation.

**PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (**bool**)  
If **true**, indicates that the requested choice of algorithm is mandatory.

## Advice to PMIx library implementers

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

### Description

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

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

## 5.3 Publish and Lookup Data

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

### Advice to PMIx library implementers

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

### Advice to PMIx server hosts

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

### 5.3.1 `PMIx_Publish`

#### Summary

Publish data for later access via `PMIx_Lookup`.

1

## Format

PMIx v1.0

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.

### Required Attributes

9

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

10

11

12

### Optional Attributes

13

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

14

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

15

Time in seconds before the specified operation should time out (0 indicating infinite) in

16

error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

17

18

**PMIX\_RANGE** "`pmix.range`" (`pmix_data_range_t`)

19

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

20

**PMIX\_PERSISTENCE** "`pmix.persist`" (`pmix_persistence_t`)

21

Value for calls to `PMIx_Publish`.

### Advice to PMIx library implementers

22

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

23

24

25

26

27

## Description

Publish the data in the *info* array for subsequent 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. Attempts to access the data by processes outside of the provided data range will be rejected. The persistence parameter instructs the server as to how long the data is to be retained.

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

### Advice to users

Duplicate keys within the specified data range may lead to unexpected behavior depending on host RM implementation of the backing key-value store.

### Advice to PMIx library implementers

Implementations should, to the best of their ability, detect duplicate keys and protect the user from unexpected behavior - preferably returning an error. This version of the standard does not define a specific error code to be returned, so the implementation must make it clear to the user what to expect in this scenario. One suggestion is to define an RM specific error code beyond the `PMIX_EXTERNAL_ERR_BASE` boundary. Future versions of the standard will clarify that a specific PMIx error be returned when conflicting values are published for a given key, and will provide attributes to allow modified behaviors such as overwrite.

## 5.3.2 `PMIx_Publish_nb`

### Summary

Nonblocking `PMIx_Publish` routine.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26

## Format

PMIx v1.0

```

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

```

- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of element 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.

### Required Attributes

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

### Optional Attributes

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

- PMIX\_TIMEOUT** "pmix.timeout" (`int`)  
Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.
- PMIX\_RANGE** "pmix.range" (`pmix_data_range_t`)  
Value for calls to publish/lookup/unpublish or for monitoring event notifications.
- PMIX\_PERSISTENCE** "pmix.persist" (`pmix_persistence_t`)  
Value for calls to `PMIx_Publish`.

## Advice to PMIx library implementers

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

### Description

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

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

## 5.3.3 PMIx\_Lookup

### Summary

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

### Format

*PMIx v1.0*

C

```
pmix_status_t
PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
            const pmix_info_t info[], size_t ninfo)
```

C

### INOUT data

Array of publishable data structures (array of handles)

#### IN `ndata`

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

#### IN `info`

Array of info structures (array of handles)

#### IN `ninfo`

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

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

## Required Attributes

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

## Optional Attributes

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

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

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

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

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

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

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

## Advice to PMIx library implementers

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

## Description

Lookup information published by this or another process. By default, the search will be conducted across the `PMIX_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 `value` struct. Any key that cannot be found will return with a data type of `PMIX_UNDEF`. The function will return `PMIX_SUCCESS` if *any* values can be found, so the caller must check each data element to ensure it was returned.

The `proc` field in each `pmix_pdata_t` struct will contain the namespace/rank of the process that published the data.

### Advice to users

Although this is a blocking function, it will *not* wait by default for the requested data to be published. Instead, it will block for the time required by the server to lookup its current data and return any found items. Thus, the caller is responsible for ensuring that data is published prior to executing a lookup, using `PMIX_WAIT` to instruct the server to wait for the data to be published, or for retrying until the requested data is found.

## 5.3.4 `PMIx_Lookup_nb`

### Summary

Nonblocking version of `PMIx_Lookup`.



1

## Format

PMIx v1.0

C

2

**pmix\_status\_t**

3

**PMIx\_Lookup\_nb**(char \*\*keys,

4

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

5

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

C

6

**IN keys**

7

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

8

**IN info**

9

Array of info structures (array of handles)

10

**IN ninfo**

11

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

12

**IN cbfunc**

13

Callback function (handle)

14

**IN cbdata**

15

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

16

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

### Required Attributes

17

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

20

### Optional Attributes

21

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

22

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

23

Time in seconds before the specified operation should time out (0 indicating infinite) in

24

error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

25

26

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

27

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

28

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

29

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

30

## Advice to PMIx library implementers

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

### Description

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

## 5.3.5 PMIx\_Unpublish

### Summary

Unpublish data posted by this process using the given keys.

### Format

*PMIx v1.0*

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

#### 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.

### Required Attributes

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

## Optional Attributes

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

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

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

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

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

## Advice to PMIx library implementers

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

## Description

Unpublish data posted by this process using the given *keys*. The function will block until the data has been removed by the server (i.e., it is safe to publish that key again). A value of **NULL** for the *keys* parameter instructs the server to remove *all* data published by this process.

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

## 5.3.6 PMIx\_Unpublish\_nb

### Summary

Nonblocking version of **PMIx\_Unpublish**.

1 **Format**

PMIx v1.0

C

```

2 pmix_status_t
3 PMIx_Unpublish_nb(char **keys,
4                   const pmix_info_t info[], size_t ninfo,
5                   pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN keys**  
(array of strings)
- 8 **IN info**  
Array of info structures (array of handles)
- 10 **IN ninfo**  
Number of element in the *info* array (integer)
- 12 **IN cbfunc**  
Callback function `pmix_op_cbfunc_t` (function reference)
- 14 **IN cbdata**  
Data to be passed to the callback function (memory reference)

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

Required Attributes

17 PMIx libraries are not required to directly support any attributes for this function. However, any  
18 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
19 *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
20 requesting the operation.

Optional Attributes

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

- 22 **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
23 Time in seconds before the specified operation should time out (0 indicating infinite) in  
24 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent  
25 the target process from ever exposing its data.
- 26 **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
27 Value for calls to publish/lookup/unpublish or for monitoring event notifications.

## Advice to PMIx library implementers

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

### Description

7 Non-blocking form of the `PMIx_Unpublish` function. The callback function will be executed  
8 once the server confirms removal of the specified data. The *info* array must be maintained until the  
9 callback is provided.  
10

## CHAPTER 6

# Process Management

---

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

## 3 6.1 Abort

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

### 6 6.1.1 PMIx\_Abort

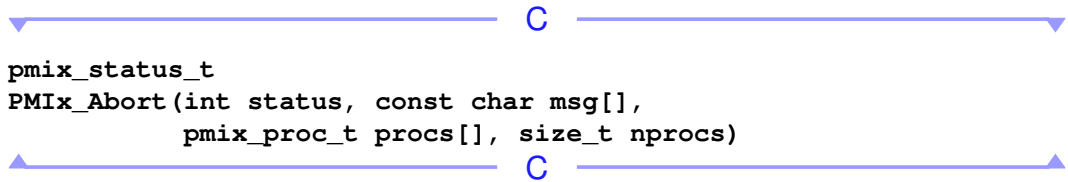
#### 7 Summary

8 Abort the specified processes

#### 9 Format

10 *PMIx v1.0*

```
10 pmix_status_t  
11 PMIx_Abort(int status, const char msg[],  
12             pmix_proc_t procs[], size_t nprocs)  
13
```



13 **IN** **status**  
14 Error code to return to invoking environment (integer)  
15 **IN** **msg**  
16 String message to be returned to user (string)  
17 **IN** **procs**  
18 Array of [pmix\\_proc\\_t](#) structures (array of handles)  
19 **IN** **nprocs**  
20 Number of elements in the *procs* array (integer)

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

## Description

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

### Advice to users

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

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

## 6.2 Process Creation

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

### 6.2.1 PMIx\_Spawn

#### Summary

Spawn a new job.

1 **Format**

PMIx v1.0

C

```

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

```

C

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

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

▼----- Required Attributes -----▼

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

- 20 **PMIX\_SPAWNED** "`pmix.spawned`" (`bool`)  
21 `true` if this process resulted from a call to `PMIx_Spawn`.
- 22 **PMIX\_PARENT\_ID** "`pmix.parent`" (`pmix_proc_t`)  
23 Process identifier of the parent process of the calling process.
- 24 **PMIX\_REQUESTOR\_IS\_CLIENT** "`pmix.req.client`" (`bool`)  
25 The requesting process is a PMIx client.
- 26 **PMIX\_REQUESTOR\_IS\_TOOL** "`pmix.req.tool`" (`bool`)  
27 The requesting process is a PMIx tool.

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

- 33 **PMIX\_WDIR** "`pmix.wdir`" (`char*`)  
34 Working directory for spawned processes.



1 **PMIX\_SET\_SESSION\_CWD** "pmix.ssn cwd" (bool)  
2 Set the application's current working directory to the session working directory assigned by  
3 the RM.  
4 **PMIX\_PREFIX** "pmix.prefix" (char\*)  
5 Prefix to use for starting spawned processes.  
6 **PMIX\_HOST** "pmix.host" (char\*)  
7 Comma-delimited list of hosts to use for spawned processes.  
8 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)  
9 Hostfile to use for spawned processes.

▲-----▲  
▼-----▼ **Optional Attributes** -----▼

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

11 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)  
12 Hostfile listing hosts to add to existing allocation.  
13 **PMIX\_ADD\_HOST** "pmix.addhost" (char\*)  
14 Comma-delimited list of hosts to add to the allocation.  
15 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)  
16 Preload binaries onto nodes.  
17 **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (char\*)  
18 Comma-delimited list of files to pre-position on nodes.  
19 **PMIX\_PERSONALITY** "pmix.pers" (char\*)  
20 Name of personality to use.  
21 **PMIX\_MAPPER** "pmix.mapper" (char\*)  
22 Mapping mechanism to use for placing spawned processes.  
23 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)  
24 Display process mapping upon spawn.  
25 **PMIX\_PPR** "pmix.ppr" (char\*)  
26 Number of processes to spawn on each identified resource.  
27 **PMIX\_MAPBY** "pmix.mapby" (char\*)  
28 Process mapping policy.  
29 **PMIX\_RANKBY** "pmix.rankby" (char\*)  
30 Process ranking policy.  
31 **PMIX\_BINDTO** "pmix.bindto" (char\*)  
32 Process binding policy.  
33 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)

1           Spawned processes will not call `PMIx_Init` .

2     **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)  
3       Spawned process rank that is to receive `stdin`.

4     **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (bool)  
5       Forward this process's `stdin` to the designated process.

6     **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
7       Forward `stdout` from spawned processes to this process.

8     **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
9       Forward `stderr` from spawned processes to this process.

10    **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)  
11      Spawned application consists of debugger daemons.

12    **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)  
13      Tag application output with the identity of the source process.

14    **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)  
15      Timestamp output from applications.

16    **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)  
17      Merge `stdout` and `stderr` streams from application processes.

18    **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)  
19      Output application output to the specified file.

20    **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)  
21      Mark the `argv` with the rank of the process.

22    **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)  
23      Number of cpus to assign to each rank.

24    **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)  
25      Do not place processes on the head node.

26    **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
27      Do not oversubscribe the cpus.

28    **PMIX\_REPORT\_BINDINGS** "pmix.repbinding" (bool)  
29      Report bindings of the individual processes.

30    **PMIX\_CPU\_LIST** "pmix.cpulists" (char\*)  
31      List of cpus to use for this job.

32    **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)  
33      Application supports recoverable operations.

34    **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)  
35      Application is continuous, all failed processes should be immediately restarted.

1 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)

2 Maximum number of times to restart a job.



### 3 **Description**

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

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

#### ▼ **Advice to users** ▼

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



## 16 **6.2.2 PMIx\_Spawn\_nb**

### 17 **Summary**

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

1

## Format

PMIx v1.0

C

2

`pmix_status_t`

3

`PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,`

4

`const pmix_app_t apps[], size_t napps,`

5

`pmix_spawn_cbfunc_t cbfunc, void *cbdata)`

C

6

**IN** `job_info`

7

Array of info structures (array of handles)

8

**IN** `ninfo`

9

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

10

**IN** `apps`

11

Array of `pmix_app_t` structures (array of handles)

12

**IN** `cbfunc`

13

Callback function `pmix_spawn_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.

### Required Attributes

17

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

20

**PMIX\_SPAWNED** "`pmix.spawned`" (bool)

21

true if this process resulted from a call to `PMIx_Spawn`.

22

**PMIX\_PARENT\_ID** "`pmix.parent`" (`pmix_proc_t`)

23

Process identifier of the parent process of the calling process.

24

**PMIX\_REQUESTOR\_IS\_CLIENT** "`pmix.req.client`" (bool)

25

The requesting process is a PMIx client.

26

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

27

The requesting process is a PMIx tool.

28

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

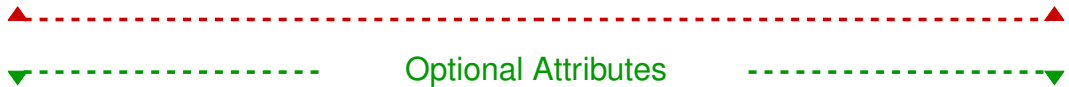
33

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

34

Working directory for spawned processes.

1 **PMIX\_SET\_SESSION\_CWD** "pmix.ssn cwd" (bool)  
2 Set the application's current working directory to the session working directory assigned by  
3 the RM.  
4 **PMIX\_PREFIX** "pmix.prefix" (char\*)  
5 Prefix to use for starting spawned processes.  
6 **PMIX\_HOST** "pmix.host" (char\*)  
7 Comma-delimited list of hosts to use for spawned processes.  
8 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)  
9 Hostfile to use for spawned processes.



### Optional Attributes

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

11 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)  
12 Hostfile listing hosts to add to existing allocation.  
13 **PMIX\_ADD\_HOST** "pmix.addhost" (char\*)  
14 Comma-delimited list of hosts to add to the allocation.  
15 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)  
16 Preload binaries onto nodes.  
17 **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (char\*)  
18 Comma-delimited list of files to pre-position on nodes.  
19 **PMIX\_PERSONALITY** "pmix.pers" (char\*)  
20 Name of personality to use.  
21 **PMIX\_MAPPER** "pmix.mapper" (char\*)  
22 Mapping mechanism to use for placing spawned processes.  
23 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)  
24 Display process mapping upon spawn.  
25 **PMIX\_PPR** "pmix.ppr" (char\*)  
26 Number of processes to spawn on each identified resource.  
27 **PMIX\_MAPBY** "pmix.mapby" (char\*)  
28 Process mapping policy.  
29 **PMIX\_RANKBY** "pmix.rankby" (char\*)  
30 Process ranking policy.  
31 **PMIX\_BINDTO** "pmix.bindto" (char\*)  
32 Process binding policy.  
33 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)

1           Spawned processes will not call `PMIx_Init` .

2     **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)  
3           Spawned process rank that is to receive `stdin`.

4     **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (bool)  
5           Forward this process's `stdin` to the designated process.

6     **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
7           Forward `stdout` from spawned processes to this process.

8     **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
9           Forward `stderr` from spawned processes to this process.

10    **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)  
11           Spawned application consists of debugger daemons.

12    **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)  
13           Tag application output with the identity of the source process.

14    **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)  
15           Timestamp output from applications.

16    **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)  
17           Merge `stdout` and `stderr` streams from application processes.

18    **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)  
19           Output application output to the specified file.

20    **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)  
21           Mark the `argv` with the rank of the process.

22    **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)  
23           Number of cpus to assign to each rank.

24    **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)  
25           Do not place processes on the head node.

26    **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
27           Do not oversubscribe the cpus.

28    **PMIX\_REPORT\_BINDINGS** "pmix.repbinding" (bool)  
29           Report bindings of the individual processes.

30    **PMIX\_CPU\_LIST** "pmix.cpulists" (char\*)  
31           List of cpus to use for this job.

32    **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)  
33           Application supports recoverable operations.

34    **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)  
35           Application is continuous, all failed processes should be immediately restarted.

1 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)  
2 Maximum number of times to restart a job.



3 **Description**

4 Nonblocking version of the **PMIx\_Spawn** routine. The provided callback function will be  
5 executed upon successful start of *all* specified application processes.

▼ **Advice to users** ▼

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



9 **6.3 Connecting and Disconnecting Processes**

10 This section defines functions to connect and disconnect processes in two or more separate PMIx  
11 namespaces. The PMIx definition of *connected* solely implies the following:

- 12 • job-level information for each namespace is to be made available to all processes in the  
13 connected assemblage
- 14 • any data posted by a process in the connected assemblage via calls to **PMIx\_Put** and  
15 committed via **PMIx\_Commit** is to be made accessible to all processes in the assemblage
- 16 • the host environment should treat the failure of any process in the assemblage as a reportable  
17 event, taking action on the assemblage as if it were a single application. For example, if the  
18 environment defaults (in the absence of any application directives) to terminating an application  
19 upon failure of any process in that application, then the environment should terminate all  
20 processes in the connected assemblage upon failure of any member.

▼ **Advice to PMIx server hosts** ▼

21 The host environment is not required to assign a new namespace to the connected assemblage, nor  
22 to assign new ranks for its members. However, it is required to generate a  
23 **PMIX\_ERR\_INVALID\_TERMINATION** event should any process in the assemblage terminate or  
24 call **PMIx\_Finalize** without first *disconnecting* from the assemblage.



## Advice to users

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

4 The PMIx implementation is not required to provide any tracking support for the assemblage. Thus,  
5 the application is responsible for maintaining the membership list of the assemblage.

### 6.3.1 PMIx\_Connect

#### Summary

Connect namespaces.

#### Format

PMIx v1.0

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

#### IN **procs**

Array of proc structures (array of handles)

#### IN **nprocs**

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

#### IN **info**

Array of info structures (array of handles)

#### IN **ninfo**

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

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

#### Required Attributes

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



## Optional Attributes

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

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

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

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

Comma-delimited list of algorithms to use for the collective operation.

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

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

## Advice to PMIx library implementers

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

## Description

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

## Advice to users

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

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

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

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

## 10 6.3.2 PMIx\_Connect\_nb

### 11 Summary

12 Nonblocking **PMIx\_Connect\_nb** routine.

### 13 Format

PMIx v1.0

C

```
14 pmix_status_t  
15 PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,  
16                 const pmix_info_t info[], size_t ninfo,  
17                 pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

18 **IN procs**  
19 Array of proc structures (array of handles)  
20 **IN nprocs**  
21 Number of elements in the *procs* array (integer)  
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.

## Required Attributes

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

## Optional Attributes

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

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

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

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

Comma-delimited list of algorithms to use for the collective operation.

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

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

## Advice to PMIx library implementers

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

## Description

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

### 6.3.3 PMIx\_Disconnect

#### Summary

Disconnect a previously connected set of processes.

1 **Format**

PMIx v1.0

C

```

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

```

C

- 5 **IN procs**  
6 Array of proc structures (array of handles)
- 7 **IN nprocs**  
8 Number of elements in the *procs* array (integer)
- 9 **IN info**  
10 Array of info structures (array of handles)
- 11 **IN ninfo**  
12 Number of element in the *info* array (integer)

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

Required Attributes

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

Optional Attributes

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

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

Advice to PMIx library implementers

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

## Description

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

### Advice to users

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

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

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

## 6.3.4 `PMIx_Disconnect_nb`

### Summary

Nonblocking `PMIx_Disconnect` routine.

### Format

*PMIx v1.0*

C

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

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.

### Required Attributes

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

### Optional Attributes

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

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

### Advice to PMIx library implementers

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

## Description

27  
 28 Nonblocking `PMIx_Disconnect` routine. The callback function is called once all processes  
 29 identified in *procs* have called either `PMIx_Disconnect_nb` or its blocking version, *and* the  
 30 host environment has completed any required supporting operations.

## CHAPTER 7

# Job Allocation Management and Reporting

---

1 The job management APIs provide an application with the ability to orchestrate its operation in  
2 partnership with the SMS. Members of this category include the  
3 [PMIx\\_Allocation\\_request\\_nb](#), [PMIx\\_Job\\_control\\_nb](#), and  
4 [PMIx\\_Process\\_monitor\\_nb](#) APIs.

## 5 7.1 Query

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

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

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

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

## 1 7.1.1 PMIx\_Resolve\_peers

### 2 Summary

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

### 4 Format

PMIx v1.0

C

```
5 pmix_status_t  
6 PMIx_Resolve_peers(const char *nodename,  
7                   const pmix_namespace_t nspace,  
8                   pmix_proc_t **procs, size_t *nprocs)
```

C

9 **IN** **nodename**

10 Name of the node to query (string)

11 **IN** **nspace**

12 namespace (string)

13 **OUT** **procs**

14 Array of process structures (array of handles)

15 **OUT** **nprocs**

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

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

### 18 Description

19 Given a *nodename*, return the array of processes within the specified *nspace* that are executing on  
20 that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified  
21 node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will  
22 be 0. The caller is responsible for releasing the *procs* array when done with it. The  
23 **PMIX\_PROC\_FREE** macro is provided for this purpose.

## 24 7.1.2 PMIx\_Resolve\_nodes

### 25 Summary

26 Return a list of nodes hosting processes within the given namespace.



1 **Format**

*PMIx v1.0*

C

2 `pmix_status_t`

3 `PMIx_Resolve_nodes(const char *nspace, char **nodelist)`

C

4 **IN** `nspace`

Namespace (string)

6 **OUT** `nodelist`

Comma-delimited list of nodenames (string)

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

9 **Description**

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

13 **7.1.3 PMIx\_Query\_info\_nb**

14 **Summary**

15 Query information about the system in general.

16 **Format**

*PMIx v2.0*

C

17 `pmix_status_t`

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

C

20 **IN** `queries`

Array of query structures (array of handles)

22 **IN** `nqueries`

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

24 **IN** `cbfunc`

Callback function `pmix_info_cbfunc_t` (function reference)

26 **IN** `cbdata`

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

28 Returns one of the following constants:

1 **PMIX\_SUCCESS** All data has been returned  
2 **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available  
3 **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the data has been returned  
4 **PMIX\_ERR\_NOT\_SUPPORTED** The host RM does not support this function

▼----- Required Attributes -----▼

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



▼----- Optional Attributes -----▼

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

10 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (char\*)  
11 Request a comma-delimited list of active namespaces.

12 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (pmix\_status\_t)  
13 Status of a specified, currently executing job.

14 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (char\*)  
15 Request a comma-delimited list of scheduler queues.

16 **PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (TBD)  
17 Status of a specified scheduler queue.

18 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (char\*)  
19 Input namespace of the job whose information is being requested returns (  
20 **pmix\_data\_array\_t**) an array of **pmix\_proc\_info\_t**.

21 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)  
22 Input namespace of the job whose information is being requested returns (  
23 **pmix\_data\_array\_t**) an array of **pmix\_proc\_info\_t** for processes in job on same  
24 node.

25 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)  
26 Return a comma-delimited list of supported spawn attributes.

27 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)  
28 Return a comma-delimited list of supported debug attributes.

29 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)  
30 Return information on memory usage for the processes indicated in the qualifiers.

31 **PMIX\_QUERY\_LOCAL\_ONLY** "pmix.qry.local" (bool)  
32 Constrain the query to local information only.

33 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)

1 Report average values.  
2 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)  
3 Report minimum and maximum values.  
4 **PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)  
5 String identifier of the allocation whose status is being requested.  
6 **PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)  
7 Query number of seconds (uint32\_t) remaining in allocation for the specified namespace.  
8



9 **Description**

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

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

16 The *status* argument to the callback function indicates if requested data was found or not. An array  
17 of **pmix\_info\_t** will contain each key that was provided and the corresponding value that was  
18 found. Requests for keys that are not found will return the key paired with a value of type  
19 **PMIX\_UNDEF** .

▼ **Advice to users** ▼

20 The desire to query a list of attributes supported by the implementation and/or the host environment  
21 has been expressed and noted. The PMIx community is exploring the possibility and it will likely  
22 become available in a future release



## 1 7.2 Allocation Requests

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

- 4 • *Evolving* applications that dynamically request and return resources as they execute
- 5 • *Malleable* environments where the scheduler redirects resources away from executing  
6 applications for higher priority jobs or load balancing
- 7 • *Resilient* applications that need to request replacement resources in the face of failures
- 8 • *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time,  
9 but realizes that they underestimated their required time while executing

10 PMIx attempts to address this range of use-cases with a single, flexible API.

### 11 7.2.1 PMIx\_Allocation\_request\_nb

#### 12 Summary

13 Request an allocation operation from the host resource manager.

#### 14 Format

PMIx v2.0

C

```
15 pmix_status_t  
16 PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,  
17 pmix_info_t info[], size_t ninfo,  
18 pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

19 **IN directive**

Allocation directive (handle)

20 **IN info**

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

21 **IN ninfo**

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

22 **IN cbfunc**

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

23 **IN cbdata**

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

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

## Required Attributes

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

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

**PMIX\_ALLOC\_ID** "pmix.alloc.id" (char\*)

Provide a string identifier for this allocation request which can later be used to query status of the request.

**PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (uint64\_t)

The number of nodes.

**PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

Number of cpus.

**PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)

Time in seconds.

## Optional Attributes

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

**PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)

Regular expression of the specific nodes.

**PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)

Regular expression of the number of cpus for each node.

**PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)

Regular expression of the specific cpus indicating the cpus involved.

**PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)

Number of Megabytes.

**PMIX\_ALLOC\_NETWORK** "pmix.alloc.net" (array)

Array of **pmix\_info\_t** describing requested network resources. If not given as part of an **pmix\_info\_t** struct that identifies the involved nodes, then the description will be applied across all nodes in the requestor's allocation.

**PMIX\_ALLOC\_NETWORK\_ID** "pmix.alloc.netid" (char\*)

Name of the network.

**PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)

Mbits/sec.

1 **PMIX\_ALLOC\_NETWORK\_QOS** "pmix.alloc.netqos" (char\*)

2 Quality of service level.



### 3 **Description**

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

- 6 • Request allocation of additional resources, including memory, bandwidth, and compute. This  
7 should be accomplished in a non-blocking manner so that the application can continue to  
8 progress while waiting for resources to become available. Note that the new allocation will be  
9 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one  
10 allocation will not impact the other.
- 11 • Extend the reservation on currently allocated resources, subject to scheduling availability and  
12 priorities. This includes extending the time limit on current resources, and/or requesting  
13 additional resources be allocated to the requesting job. Any additional allocated resources will be  
14 considered as part of the current allocation, and thus will be released at the same time.
- 15 • Return no-longer-required resources to the scheduler. This includes the “loan” of resources back  
16 to the scheduler with a promise to return them upon subsequent request.

## 17 **7.2.2 PMIx\_Job\_control\_nb**

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

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

## Summary

Request a job control action.

## Format

PMIx v2.0

C

```
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)
```

C

**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)

**IN ndirs**

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

**IN cbfunc**

Callback function `pmix_info_cbfunc_t` (function reference)

**IN cbdata**

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

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

### Required Attributes

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

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

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

Provide a string identifier for this request.

`PMIX_JOB_CTRL_PAUSE` "pmix.jctrl.pause" (bool)

Pause the specified processes.

`PMIX_JOB_CTRL_RESUME` "pmix.jctrl.resume" (bool)

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

`PMIX_JOB_CTRL_KILL` "pmix.jctrl.kill" (bool)

1           Forcibly terminate the specified processes and cleanup.

2     **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)

3           Send given signal to specified processes.

4     **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)

5           Politely terminate the specified processes.



▼----- Optional Attributes -----▼

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

7     **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)

8           Cancel the specified request (**NULL** implies cancel all requests from this requestor).

9     **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)

10           Restart the specified processes using the given checkpoint ID.

11     **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)

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

13     **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)

14           Use event notification to trigger a process checkpoint.

15     **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)

16           Use the given signal to trigger a process checkpoint.

17     **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)

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

19     **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**

20     "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)

21           Array of **pmix\_info\_t** declaring each method and value supported by this application.

22     **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)

23           Regular expression identifying nodes that are to be provisioned.

24     **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvning" (char\*)

25           Name of the image that is to be provisioned.

26     **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

27           Indicate that the job can be pre-empted.





## Description

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

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

## 7.3 Process and Job Monitoring

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

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

### 7.3.1 PMIx\_Process\_monitor\_nb

#### Summary

Request that application processes be monitored.

#### Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
                        const pmix_info_t directives[], size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```

```

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.

### Optional Attributes

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

```

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

```

1 Monitor time since last access of given file to determine if the application is running.

2 **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)

3 Monitor time since last modified of given file to determine if the application is running.

4 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)

5 Time in seconds between checking the file.

6 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)

7 Number of file checks that can be missed before generating the event.



8 **Description**

9 Request that application processes be monitored via several possible methods. For example, that  
 10 the server monitor this process for periodic heartbeats as an indication that the process has not  
 11 become “wedged”. When a monitor detects the specified alarm condition, it will generate an event  
 12 notification using the provided error code and passing along any available relevant information. It  
 13 is up to the caller to register a corresponding event handler.

14 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,  
 15 **PMIX\_MONITOR\_FILE** to indicate that the requestor is asking that a file be monitored.

16 The *error* argument is the status code to be used when generating an event notification alerting that  
 17 the monitor has been triggered. The range of the notification defaults to  
 18 **PMIX\_RANGE\_NAMESPACE**. This can be changed by providing a **PMIX\_RANGE** directive.

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

21 The *cbfunc* function provides a *status* to indicate whether or not the request was granted, and to  
 22 provide some information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
 23 **pmix\_info\_t** structures.

24 **7.3.2 PMIx\_Heartbeat**

25 **Summary**

26 Send a heartbeat to the PMIx server library

27 **Format**

PMIx v2.0

28 **void PMIx\_Heartbeat (void)**

1       **Description**

2       A simplified macro wrapping `PMIx_Process_monitor_nb` that sends a heartbeat to the  
3       PMIx server library.

4       **7.4 Logging**

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

9       **7.4.1 PMIx\_Log\_nb**

10      **Summary**

11      Log data to a data service.

12      **Format**

13      *PMIx v2.0*

```
13      pmix_status_t  
14      PMIx_Log_nb(const pmix_info_t data[], size_t ndata,  
15                  const pmix_info_t directives[], size_t ndirs,  
16                  pmix_op_cbfunc_t cbfunc, void *cbdata)
```

- 17      **IN data**  
18          Array of info structures (array of handles)
- 19      **IN ndata**  
20          Number of elements in the *data* array (**size\_t**)
- 21      **IN directives**  
22          Array of info structures (array of handles)
- 23      **IN ndirs**  
24          Number of elements in the *directives* array (**size\_t**)
- 25      **IN cbfunc**  
26          Callback function `pmix_op_cbfunc_t` (function reference)
- 27      **IN cbdata**  
28          Data to be passed to the callback function (memory reference)

29      Return codes are one of the following:

1 **PMIX\_SUCCESS** The logging request is valid and is being processed. The resulting status from  
2 the operation will be provided in the callback function.  
3 **PMIX\_ERR\_BAD\_PARAM** The logging request contains at least one incorrect entry that prevents  
4 it from being processed. The callback function will *not* be called.  
5 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function. The  
6 callback function will *not* be called.

▼----- Required Attributes -----▼

7 If the PMIx library does not itself perform this operation, then it is required to pass any attributes  
8 provided by the client to the host environment for processing. In addition, it must include the  
9 following attributes in the passed *info* array:

10 **PMIX\_USERID** "pmix.euid" (uint32\_t)  
11 Effective user id.

12 **PMIX\_GRPID** "pmix.egid" (uint32\_t)  
13 Effective group id.

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

16 **PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)  
17 Log string to **stderr**.

18 **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)  
19 Log string to **stdout**.

20 **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (char\*)  
21 Log data to syslog. Defaults to **ERROR** priority.



▼----- Optional Attributes -----▼

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

23 **PMIX\_LOG\_MSG** "pmix.log.msg" (pmix\_byte\_object\_t)  
24 Message blob to be sent somewhere.

25 **PMIX\_LOG\_EMAIL** "pmix.log.email" (pmix\_data\_array\_t)  
26 Log via email based on **pmix\_info\_t** containing directives.

27 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (char\*)  
28 Comma-delimited list of email addresses that are to receive the message.

29 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (char\*)  
30 Subject line for email.

31 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)  
32 Message to be included in email.



1  
2  
3  
4  
5  
6  
7  
8

## Description

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

▼ Advice to users ▼

It is strongly recommended that the [PMIx\\_Log\\_nb](#) API not be used by applications for streaming data as it is not a “performant” transport and can perturb the application since it involves the local PMIx server and host SMS daemon.

## CHAPTER 8

# Event Notification

---

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

## 4 8.1 Notification and Management

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

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

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

### ▼ Advice to PMIx library implementers ▼

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

---

## Advice to users

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

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

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

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

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

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



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

### Advice to users

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

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

## 17 8.1.1 PMIx\_Register\_event\_handler

### 18 Summary

19 Register an event handler

### 20 Format

PMIx v2.0

C

```
21 void  
22 PMIx_Register_event_handler(pmixon_status_t codes[], size_t ncodes,  
23                             pmixon_info_t info[], size_t ninfo,  
24                             pmixon_notification_fn_t evhdlr,  
25                             pmixon_evhdlr_reg_cbfunc_t cbfunc,  
26                             void *cbdata);
```

```

1  IN codes
2      Array of status codes (array of pmix_status_t)
3  IN ncodes
4      Number of elements in the codes array (size_t)
5  IN info
6      Array of info structures (array of handles)
7  IN ninfo
8      Number of elements in the info array (size_t)
9  IN evhdlr
10     Event handler to be called pmix_notification_fn_t (function reference)
11 IN cbfunc
12     Callback function pmix_evhdlr_reg_cbfunc_t (function reference)
13 IN cbdata
14     Data to be passed to the cbfunc callback function (memory reference)

```

### Required Attributes

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

```

16 PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)
17     String name identifying this handler.
18 PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool)
19     Invoke this event handler before any other handlers.
20 PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool)
21     Invoke this event handler after all other handlers have been called.
22 PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool)
23     Invoke this event handler before any other handlers in this category.
24 PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
25     Invoke this event handler after all other handlers in this category have been called.
26 PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)
27     Put this event handler immediately before the one specified in the (char*) value.
28 PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)
29     Put this event handler immediately after the one specified in the (char*) value.
30 PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)
31     Prepend this handler to the precedence list within its category.
32 PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)
33     Append this handler to the precedence list within its category.
34 PMIX_EVENT_CUSTOM_RANGE "pmix.evrage" (pmix_data_array_t*)

```

1           Array of `pmix_proc_t` defining range of event notification.

2     **PMIX\_RANGE** "pmix.range" (`pmix_data_range_t`)

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

4     **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (`void *`)

5           Object to be returned whenever the registered callback function `cbfunc` is invoked. The

6           object will *only* be returned to the process that registered it.

7     Host environments that implement support for PMIx event notification are required to support the

8     following attributes:

9     **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (`pmix_proc_t`)

10           The single process that was affected.

11     **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaffected" (`pmix_data_array_t*`)

12           Array of `pmix_proc_t` defining affected processes.



▼----- Optional Attributes -----▼

13     Host environments that support PMIx event notification *may* offer notifications for environmental

14     events impacting the job and for SMS events relating to the job. The following attributes are

15     optional for host environments that support this operation:

16     **PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (`bool`)

17           The RM intends to terminate this session.

18     **PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (`bool`)

19           The RM intends to terminate this job.

20     **PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (`bool`)

21           The RM intends to terminate all processes on this node.

22     **PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (`bool`)

23           The RM intends to terminate just this process.

24     **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evtimeout" (`int`)

25           The time in seconds before the RM will execute error response.

26     **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.evsilentterm" (`bool`)

27           Do not generate an event when this job normally terminates.



## Description

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

### Advice to users

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

Upon completion, the callback will receive a status based on the following table:

`PMIX_SUCCESS` The event handler was successfully registered - the event handler identifier is returned in the callback.

`PMIX_ERR_BAD_PARAM` One or more of the directives provided in the *info* array was unrecognized.

`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support event notification, or the host SMS does not support notification of the specified event code.

### Advice to users

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

## 8.1.2 `PMIx_Deregister_event_handler`

### Summary

Deregister an event handler.

1 **Format**

PMIx v2.0

C

```

2 void
3 PMIx_Deregister_event_handler(size_t evhdlr_ref,
4                               pmix_op_cbfunc_t cbfunc,
5                               void *cbdata);

```

C

- 6 **IN** **evhdlr\_ref**  
Event handler ID returned by registration (**size\_t**)
- 7
- 8 **IN** **cbfunc**  
Callback function to be executed upon completion of operation **pmix\_op\_cbfunc\_t**  
(function reference)
- 9
- 10
- 11 **IN** **cbdata**  
Data to be passed to the cbfunc callback function (memory reference)
- 12

13 **Description**

14 Deregister an event handler. If non-NULL, the provided cbfunc will be called to confirm removal  
15 of the designated handler, including a status code as per the following:

- 16 **PMIX\_SUCCESS** The event handler was successfully deregistered.
- 17 **PMIX\_ERR\_BAD\_PARAM** The provided *evhdlr\_ref* was unrecognized.
- 18 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification.

19 **8.1.3 PMIx\_Notify\_event**

20 **Summary**

21 Report an event for notification via any registered event handler.

22 **Format**

PMIx v2.0

C

```

23 pmix_status_t
24 PMIx_Notify_event(pmix_status_t status,
25                  const pmix_proc_t *source,
26                  pmix_data_range_t range,
27                  pmix_info_t info[], size_t ninfo,
28                  pmix_op_cbfunc_t cbfunc, void *cbdata);

```

1 **IN status**  
 2 Status code of the event (`pmix_status_t`)  
 3 **IN source**  
 4 Pointer to a `pmix_proc_t` identifying the original reporter of the event (handle)  
 5 **IN range**  
 6 Range across which this notification shall be delivered (`pmix_data_range_t`)  
 7 **IN info**  
 8 Array of `pmix_info_t` structures containing any further info provided by the originator  
 9 of the event (array of handles)  
 10 **IN ninfo**  
 11 Number of elements in the *info* array (`size_t`)  
 12 **IN cbfunc**  
 13 Callback function to be executed upon completion of operation `pmix_op_cbfunc_t`  
 14 (function reference)  
 15 **IN cbdata**  
 16 Data to be passed to the `cbfunc` callback function (memory reference)

17 **PMIX\_SUCCESS** The notification request is valid and is being processed. The callback function  
 18 will be called when the process-local operation is complete and will provide the resulting  
 19 status of that operation. Note that this does *not* reflect the success or failure of delivering the  
 20 event to any recipients.

21 **PMIX\_ERR\_BAD\_PARAM** The request contains at least one incorrect entry that prevents it from  
 22 being processed. The callback function will *not* be called.

23 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification,  
 24 or in the case of a PMIx server calling the API, the range extended beyond the local node and  
 25 the host SMS environment does not support event notification. The callback function will  
 26 *not* be called.

### Required Attributes

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

28 **PMIX\_EVENT\_NON\_DEFAULT** "`pmix.evnondef`" (`bool`)

29 Event is not to be delivered to default event handlers.

30 **PMIX\_EVENT\_CUSTOM\_RANGE** "`pmix.evrangle`" (`pmix_data_array_t*`)

31 Array of `pmix_proc_t` defining range of event notification.

32 Host environments that implement support for PMIx event notification are required to provide the  
 33 following attributes for all events generated by the environment:

34 **PMIX\_EVENT\_AFFECTED\_PROC** "`pmix.evproc`" (`pmix_proc_t`)

35 The single process that was affected.

36 **PMIX\_EVENT\_AFFECTED\_PROCS** "`pmix.evaffected`" (`pmix_data_array_t*`)

37 Array of `pmix_proc_t` defining affected processes.



1

## Description

2

Report an event for notification via any registered event handler. This function can be called by any PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server calls this API to report events it detected itself so that the host SMS daemon distribute and handle them, and to pass events given to it by its host down to any attached client processes for processing. Examples might include notification of the failure of another process, detection of an impending node failure due to rising temperatures, or an intent to preempt the application. Events may be locally generated or come from anywhere in the system.

3

4

5

6

7

8

9

Host SMS daemons call the API to pass events down to its embedded PMIx server both for transmittal to local client processes and for the server's own internal processing.

10

11

Client application processes can call this function to notify the SMS and/or other application processes of an event it encountered. Note that processes are not constrained to report status values defined in the official PMIx standard — any integer value can be used. Thus, applications are free to define their own internal events and use the notification system for their own internal purposes.

12

13

14

### Advice to users

15

The callback function will be called upon completion of the **notify\_event** function's actions.

16

At that time, any messages required for executing the operation (e.g., to send the notification to the local PMIx server) will have been queued, but may not yet have been transmitted. The caller is required to maintain the input data until the callback function has been executed — the sole purpose of the callback function is to indicate when the input data is no longer required.

17

18

19



## CHAPTER 9

# Data Packing and Unpacking

---

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

## 8    9.1   Support Macros

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

### 10  9.1.1  PMIX\_DATA\_BUFFER\_CREATE

#### 11       Summary

12       Allocate memory for a `pmix_data_buffer_t` object and initialize it

#### 13       Format

14       *PMIx v2.0*   ▼ `PMIX_DATA_BUFFER_CREATE(buffer);`   C   ▶

#### 15       OUT `buffer`

16       Variable to be assigned the pointer to the allocated `pmix_data_buffer_t` (handle)

#### 17       Description

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



## 1 9.1.2 PMIX\_DATA\_BUFFER\_RELEASE

### 2 Summary

3 Free a `pmix_data_buffer_t` object and the data it contains

### 4 Format

*PMIx v2.0*

```
▼ _____ C _____ ▼  
PMIX_DATA_BUFFER_RELEASE (buffer);  
▲ _____ C _____ ▲
```

6 **IN** `buffer`

7 Pointer to the `pmix_data_buffer_t` to be released (handle)

### 8 Description

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

## 10 9.1.3 PMIX\_DATA\_BUFFER\_CONSTRUCT

### 11 Summary

12 Initialize a statically declared `pmix_data_buffer_t` object

### 13 Format

*PMIx v2.0*

```
▼ _____ C _____ ▼  
PMIX_DATA_BUFFER_CONSTRUCT (buffer);  
▲ _____ C _____ ▲
```

15 **IN** `buffer`

16 Pointer to the allocated `pmix_data_buffer_t` that is to be initialized (handle)

### 17 Description

18 Initialize a pre-allocated buffer object

## 19 9.1.4 PMIX\_DATA\_BUFFER\_DESTRUCT

### 20 Summary

21 Release the data contained in a `pmix_data_buffer_t` object

1 **Format**

*PMIx v2.0*

```

▼ _____ C _____ ▼
2 PMIX_DATA_BUFFER_DESTRUCT(buffer);
▲ _____ C _____ ▲

```

3 **IN buffer**

4 Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)

5 **Description**

6 Free's the data contained in a `pmix_data_buffer_t` object

7 **9.1.5 PMIX\_DATA\_BUFFER\_LOAD**

8 **Summary**

9 Load a blob into a `pmix_data_buffer_t` object

10 **Format**

*PMIx v2.0*

```

▼ _____ C _____ ▼
11 PMIX_DATA_BUFFER_LOAD(buffer, data, size);
▲ _____ C _____ ▲

```

12 **IN buffer**

13 Pointer to a pre-allocated `pmix_data_buffer_t` (handle)

14 **IN data**

15 Pointer to a blob (`char*`)

16 **IN size**

17 Number of bytes in the blob `size_t`

18 **Description**

19 Load the given data into the provided `pmix_data_buffer_t` object, usually done in  
20 preparation for unpacking the provided data. Note that the data is *not* copied into the buffer - thus,  
21 the blob must not be released until after operations on the buffer have completed.

22 **9.1.6 PMIX\_DATA\_BUFFER\_UNLOAD**

23 **Summary**

24 Unload the data from a `pmix_data_buffer_t` object

1

## Format

*PMIx v2.0*

C

2

```
PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

3

**IN** `buffer`

4

Pointer to the `pmix_data_buffer_t` whose data is to be extracted (handle)

5

**OUT** `data`

6

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

7

**OUT** `size`

8

Variable to be assigned the number of bytes in the blob `size_t`

9

## Description

10

Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release - thus, the user is responsible for releasing the blob when done with it.

11

12

13

## 14 9.2 General Routines

15

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

### 16 9.2.1 `PMIx_Data_pack`

17

#### Summary

18

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

19

#### Format

*PMIx v2.0*

C

20

`pmix_status_t`

21

```
PMIx_Data_pack(const pmix_proc_t *target,  
               pmix_data_buffer_t *buffer,  
               void *src, int32_t num_vals,  
               pmix_data_type_t type);
```

22

23

24

- 1 **IN target**  
 2 Pointer to a `pmix_proc_t` containing the nspace/rank of the process that will be  
 3 unpacking the final buffer. A NULL value may be used to indicate that the target is based on  
 4 the same PMIx version as the caller. Note that only the target's nspace is relevant. (handle)
- 5 **IN buffer**  
 6 Pointer to a `pmix_data_buffer_t` where the packed data is to be stored (handle)
- 7 **IN src**  
 8 Pointer to a location where the data resides. Strings are to be passed as (char \*\*) — i.e., the  
 9 caller must pass the address of the pointer to the string as the (void\*). This allows the caller  
 10 to pass multiple strings in a single call. (memory reference)
- 11 **IN num\_vals**  
 12 Number of elements pointed to by the *src* pointer. A string value is counted as a single value  
 13 regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,  
 14 string arrays) should be contiguous, although the data pointed to need not be contiguous  
 15 across array entries.(`int32_t`)
- 16 **IN type**  
 17 The type of the data to be packed ( `pmix_data_type_t` )
- 18 **PMIX\_SUCCESS** The data has been packed as requested
- 19 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.
- 20 **PMIX\_ERR\_BAD\_PARAM** The provided buffer or src is **NULL**
- 21 **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
 22 implementation
- 23 **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation
- 24 **PMIX\_ERROR** General error

## 25 Description

26 The pack function packs one or more values of a specified type into the specified buffer. The buffer  
 27 must have already been initialized via the `PMIX_DATA_BUFFER_CREATE` or  
 28 `PMIX_DATA_BUFFER_CONSTRUCT` macros — otherwise, `PMIx_Data_pack` will return an  
 29 error. Providing an unsupported type flag will likewise be reported as an error.

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

35 The namespace of the intended recipient of the packed buffer (i.e., the process that will be  
 36 unpacking it) is used solely to resolve any data type differences between PMIx versions. The  
 37 recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx  
 38 library is aware of the version the recipient is using. Note that all processes in a given namespace

1 are *required* to use the same PMIx version — thus, the caller must only know at least one process  
2 from the target’s namespace.

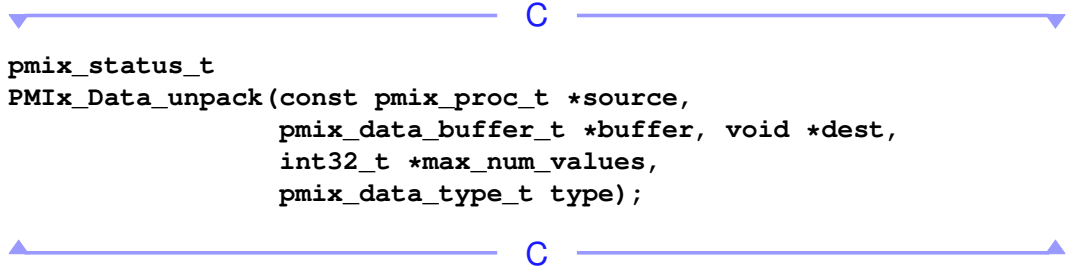
### 3 **9.2.2 PMIx\_Data\_unpack**

#### 4 **Summary**

5 Unpack values from a `pmix_data_buffer_t`

#### 6 **Format**

PMIx v2.0



```
7 pmix_status_t  
8 PMIx_Data_unpack(const pmix_proc_t *source,  
9                 pmix_data_buffer_t *buffer, void *dest,  
10                int32_t *max_num_values,  
11                pmix_data_type_t type);  
12
```

13 **IN source**  
14 Pointer to a `pmix_proc_t` structure containing the nspace/rank of the process that packed  
15 the provided buffer. A NULL value may be used to indicate that the source is based on the  
16 same PMIx version as the caller. Note that only the source’s nspace is relevant. (handle)

17 **IN buffer**  
18 A pointer to the buffer from which the value will be extracted. (handle)

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

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

31 **IN type**  
32 The type of the data to be unpacked — must be one of the PMIx defined data types (  
33 `pmix_data_type_t`)

34 **PMIX\_SUCCESS** The data has been unpacked as requested

1 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.  
2 **PMIX\_ERR\_BAD\_PARAM** The provided buffer or dest is **NULL**  
3 **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
4 implementation  
5 **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation  
6 **PMIX\_ERROR** General error

## 7 **Description**

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

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

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

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

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

32 The namespace of the process that packed the buffer is used solely to resolve any data type  
33 differences between PMIx versions. The packer must, therefore, be known to the user prior to  
34 calling the `pack` function so that the PMIx library is aware of the version the packer is using. Note  
35 that all processes in a given namespace are *required* to use the same PMIx version — thus, the  
36 caller must only know at least one process from the packer's namespace.

## 1 9.2.3 PMIx\_Data\_copy

### 2 Summary

3 Copy a data value from one location to another.

### 4 Format

*PMIx v2.0*

```
▼────────────────────────────────────────── C ───────────────────────────────────▼  
5 pmix_status_t  
6 PMIx_Data_copy(void **dest, void *src,  
7                 pmix_data_type_t type);  
▲────────────────────────────────────────── C ───────────────────────────────────▲
```

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

11 **IN src**  
12 A pointer to the memory location from which the data is to be copied (handle)

13 **IN type**  
14 The type of the data to be copied — must be one of the PMIx defined data types. (  
15 [pmix\\_data\\_type\\_t](#))

16 [PMIX\\_SUCCESS](#) The data has been copied as requested

17 [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) The PMIx implementation does not support this function.

18 [PMIX\\_ERR\\_BAD\\_PARAM](#) The provided src or dest is **NULL**

19 [PMIX\\_ERR\\_UNKNOWN\\_DATA\\_TYPE](#) The specified data type is not known to this  
20 implementation

21 [PMIX\\_ERR\\_OUT\\_OF\\_RESOURCE](#) Not enough memory to support the operation

22 [PMIX\\_ERROR](#) General error

### 23 Description

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

## 28 9.2.4 PMIx\_Data\_print

### 29 Summary

30 Pretty-print a data value.

1 **Format**

*PMIx v2.0*

C

```

2 pmix_status_t
3 PMIx_Data_print(char **output, char *prefix,
4                 void *src, pmix_data_type_t type);

```

5 **IN output**

6 The address of a pointer into which the address of the resulting output is to be stored.  
7 (char\*\*)

8 **IN prefix**

9 String to be prepended to the resulting output (char\*)

10 **IN src**

11 A pointer to the memory location of the data value to be printed (handle)

12 **IN type**

13 The type of the data value to be printed — must be one of the PMIx defined data types. (  
14 pmix\_data\_type\_t)

15 **PMIX\_SUCCESS** The data has been printed as requested

16 **PMIX\_ERR\_BAD\_PARAM** The provided data type is not recognized.

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

18 **Description**

19 Since registered data types can be complex structures, the system needs some way to know how to  
20 print them (i.e., convert them to a string representation). Primarily for debug purposes.

21 **9.2.5 PMIx\_Data\_copy\_payload**

22 **Summary**

23 Copy a payload from one buffer to another

24 **Format**

*PMIx v2.0*

C

```

25 pmix_status_t
26 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
27                       pmix_data_buffer_t *src);

```



1 **IN dest**  
2     Pointer to the destination `pmix_data_buffer_t` (handle)

3 **IN src**  
4     Pointer to the source `pmix_data_buffer_t` (handle)

5 **PMIX\_SUCCESS** The data has been copied as requested

6 **PMIX\_ERR\_BAD\_PARAM** The src and dest `pmix_data_buffer_t` types do not match

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

## 8 **Description**

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

## CHAPTER 10

# Server-Specific Interfaces

---

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

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

## 10.1 Server Support Functions

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

### 10.1.1 PMIx\_generate\_regex

#### Summary

16 Generate a regular expression representation of the input string.

#### Format

*PMIx v1.0*

```
18 pmix_status_t  
19 PMIx_generate_regex(const char *input, char **regex)
```

20 **IN** **input**  
21 String to process (string)

22 **OUT** **regex**  
23 Regular expression representation of *input* (string)

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

1       **Description**

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

4       If values have leading zero's, then that is preserved, as are prefix and suffix strings. For example, an  
5       input string of

6       “**odin009.org,odin010.org,odin011.org,odin012.org,odin[102-107].org**”  
7       will return a regular expression of “**pmix:odin[009-012,102-107].org**”

▼ **Advice to users** ▼

8       The returned regular expression will have a “**pmix:**” at the beginning of the string. This informs  
9       the PMIx parser that the string was produced using the PRI's regular expression generator, and thus  
10       that same plugin should be used for parsing the string



11 **10.1.2 PMIx\_generate\_ppn**

12       **Summary**

13       Generate a regular expression representation of the input string.

14       **Format**

PMIx v1.0   ▼ **C** ▼

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



16       **IN input**  
17       String to process (string)

18       **OUT regex**  
19       Regular expression representation of *input* (string)

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

## Description

The input is expected to consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job. Thus, an input of "1-4;2-5;8,10,11,12;6,7,9" would generate a regex of "pmix:2x(3);8,10-12;6-7,9"

### Advice to users

The returned regular expression will have a "pmix:" at the beginning of the string. This informs the PMIx parser that the string was produced using the PRI's regular expression generator, and thus that same plugin should be used for parsing the string

## 10.1.3 PMIx\_server\_register\_namespace

### Summary

Setup the data about a particular namespace.

### Format

*PMIx v1.0*

```
pmix_status_t
PMIx_server_register_namespace(const pmix_namespace_t nspace,
                               int nlocalprocs,
                               pmix_info_t info[], size_t ninfo,
                               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

- IN nspace**  
namespace (string)
- IN nlocalprocs**  
number of local processes (integer)
- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of elements in the *info* array (integer)
- IN cbfunc**  
Callback function `pmix_op_cbfunc_t` (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.

## Required Attributes

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

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

Registration is for this namespace only, do not copy job data.

Host environments are *required* to provide the following attributes:

- for the given namespace:

- **PMIX\_NAMESPACE** "pmix.namespace" (char\*)

Namespace of the job.

- **PMIX\_JOBID** "pmix.jobid" (char\*)

Job identifier assigned by the scheduler.

- **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

Comma-delimited list of nodes running processes for this job.

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

Number of processes in this namespace.

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

Number of processes in this job.

- **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)

Maximum number of processes for this job.

- **PMIX\_NUM\_NODES** "pmix.num.nodes" (uint32\_t)

Number of nodes in this namespace.

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

Regular expression of nodes containing processes for this job.

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

Regular expression describing processes on each node within this job.

- for its own node:

- **PMIX\_LOCAL\_SIZE** "pmix.local.size" (uint32\_t)

Number of processes in this job on this node.

- **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)

Comma-delimited list of ranks on this node within the specified namespace.

- **PMIX\_LOCAL\_CPUSSETS** "pmix.lcpus" (char\*)

Colon-delimited cpusets of local peers within the specified namespace.

- for each process in the given namespace:

- **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)



- 1       - **PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (char\*)  
2            Comma-delimited list of all nodes in this allocation regardless of whether or not they  
3            currently host processes.
- 4       - **PMIX\_JOB\_NUM\_APPS** "pmix.job.napps" (uint32\_t)  
5            Number of applications in this job.
- 6       - **PMIX\_MAPBY** "pmix.mapby" (char\*)  
7            Process mapping policy.
- 8       - **PMIX\_RANKBY** "pmix.rankby" (char\*)  
9            Process ranking policy.
- 10       - **PMIX\_BINDTO** "pmix.bindto" (char\*)  
11            Process binding policy.
- 12       • for each application in the given namespace:
- 13       - **PMIX\_APP\_SIZE** "pmix.app.size" (uint32\_t)  
14            Number of processes in this application.
- 15       • for its own node:
- 16       - **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (uint64\_t)  
17            Total available physical memory on this node.
- 18       - **PMIX\_HWLOC\_XML\_V1** "pmix.hwlocxml1" (char\*)  
19            XML representation of local topology using hwloc's v1.x format.
- 20       - **PMIX\_HWLOC\_XML\_V2** "pmix.hwlocxml2" (char\*)  
21            XML representation of local topology using hwloc's v2.x format.
- 22       - **PMIX\_LOCALLDR** "pmix.lldr" (pmix\_rank\_t)  
23            Lowest rank on this node within this job.
- 24       - **PMIX\_NODE\_SIZE** "pmix.node.size" (uint32\_t)  
25            Number of processes across all jobs on this node.
- 26       - **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (pmix\_proc\_t array)  
27            Array of **pmix\_proc\_t** of processes on the specified node.
- 28       • for each process in the given namespace:
- 29       - **PMIX\_PROCID** "pmix.procid" (pmix\_proc\_t)  
30            Process identifier
- 31       - **PMIX\_GLOBAL\_RANK** "pmix.grank" (pmix\_rank\_t)  
32            Process rank spanning across all jobs in this session.
- 33       - **PMIX\_HOSTNAME** "pmix.hname" (char\*)  
34            Name of the host where the specified process is running.

1 Attributes not directly provided by the host environment *may* be derived by the PMIx server library  
2 from other required information and included in the data made available to the server library's  
3 clients.



## 4 Description

5 Pass job-related information to the PMIx server library for distribution to local client processes.

### ▼ Advice to PMIx server hosts ▼

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

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

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



### ▼ Advice to users ▼

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



## 24 10.1.4 PMIx\_server\_deregister\_nspace

### 25 Summary

26 Deregister a namespace.



1 **Format**

*PMIx v1.0*

```

2 void PMIx_server_deregister_nspace(const pmix_namespace_t nspace,
3                                   pmix_op_cbfunc_t cbfunc, void *cbdata)

```

- 4 **IN nspace**  
Namespace (string)
- 5
- 6 **IN cbfunc**  
Callback function `pmix_op_cbfunc_t` (function reference)
- 7
- 8 **IN cbdata**  
Data to be passed to the callback function (memory reference)
- 9

10 **Description**

11 Deregister the specified *nspace* and purge all objects relating to it, including any client information  
 12 from that namespace. This is intended to support persistent PMIx servers by providing an  
 13 opportunity for the host RM to tell the PMIx server library to release all memory for a completed  
 14 job.

15 **10.1.5 PMIx\_server\_register\_client**

16 **Summary**

17 Register a client process with the PMIx server library.

18 **Format**

*PMIx v1.0*

```

19 pmix_status_t
20 PMIx_server_register_client(const pmix_proc_t *proc,
21                             uid_t uid, gid_t gid,
22                             void *server_object,
23                             pmix_op_cbfunc_t cbfunc, void *cbdata)

```

- 24 **IN proc**  
`pmix_proc_t` structure (handle)
- 25
- 26 **IN uid**  
user id (integer)
- 27
- 28 **IN gid**  
group id (integer)
- 29

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

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

## 8 **Description**

9 Register a client process with the PMIx server library.

### ▼ **Advice to PMIx server hosts** ▼

10 Host environments are *required* to execute this operation prior to starting the client process. The  
11 expected user ID and group ID of the child process helps the server library to properly authenticate  
12 clients as they connect by requiring the two values to match.

13 The host server can also, if it desires, provide an object it wishes to be returned when a server  
14 function is called that relates to a specific process. For example, the host server may have an object  
15 that tracks the specific client. Passing the object to the library allows the library to provide that  
16 object to the host server during subsequent calls related to that client, such as a  
17 `pmix_server_client_connected_fn_t` function. This allows the host server to access  
18 the object without performing a lookup based on the client's namespace and rank.

## 19 **10.1.6 PMIx\_server\_deregister\_client**

### 20 **Summary**

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

### 22 **Format**

*PMIx v1.0*

▼ **C** ▼

```
23 void  
24 PMIx_server_deregister_client(const pmix_proc_t *proc,  
25                               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

▲ **C** ▲

26 **IN** `proc`  
27 `pmix_proc_t` structure (handle)  
28 **IN** `cbfunc`  
29 Callback function `pmix_op_cbfunc_t` (function reference)  
30 **IN** `cbdata`  
31 Data to be passed to the callback function (memory reference)

## Description

The `PMIx_server_deregister_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired.

## 10.1.7 `PMIx_server_setup_fork`

### Summary

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

### Format

*PMIx v1.0*

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

**IN** `proc`  
    `pmix_proc_t` structure (handle)  
**IN** `env`  
    Environment array (array of strings)

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

### Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

#### Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via `PMIx_Spawn`) plus other values (e.g., variables required to properly initialize the client's fabric library).

# 1 10.1.8 PMIx\_server\_dmodex\_request

## 2 Summary

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

## 4 Format

PMIx v1.0

```

5 pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,
6                                           pmix_dmodex_response_fn_t cbfunc,
7                                           void *cbdata)

```

- 8 **IN proc**  
9     **pmix\_proc\_t** structure (handle)
- 10 **IN cbfunc**  
11     Callback function **pmix\_dmodex\_response\_fn\_t** (function reference)
- 12 **IN cbdata**  
13     Data to be passed to the callback function (memory reference)

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

## 15 Description

16 Define a function by which the host server can request modex data from the local PMIx server.  
 17 Traditional wireup procedures revolve around the per-process posting of data (e.g., location and  
 18 endpoint information) via the **PMIx\_Put** and **PMIx\_Commit** functions followed by a  
 19 **PMIx\_Fence** barrier that globally exchanges the posted information. However, the barrier  
 20 operation represents a significant time impact at large scale.

21 PMIx supports an alternative wireup method known as *Direct Modex* that replaces the  
 22 barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In  
 23 place of the barrier operation, data posted by each process is cached on the local PMIx server.  
 24 When a process requests the information posted by a particular peer, it first checks the local cache  
 25 to see if the data is already available. If not, then the request is passed to the local PMIx server,  
 26 which subsequently requests that its RM host request the data from the RM daemon on the node  
 27 where the specified peer process is located. Upon receiving the request, the RM daemon passes the  
 28 request into its PMIx server library using the **PMIx\_server\_dmodex\_request** function,  
 29 receiving the response in the provided **cbfunc** once the indicated process has posted its information.  
 30 The RM daemon then returns the data to the requesting daemon, who subsequently passes the data  
 31 to its PMIx server library for transfer to the requesting client.

## Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

### 10.1.9 PMIx\_server\_setup\_application

#### Summary

Provide a function by which the resource manager can request application-specific setup data prior to launch of an application.

#### Format

PMIx v2.0

C

```
pmix_status_t
PMIx_server_setup_application(const pmix_namespace_t nspace,
                             pmix_info_t info[], size_t ninfo,
                             pmix_setup_application_cbfunc_t cbfunc,
                             void *cbdata)
```

C

**IN nspace**  
namespace (string)

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

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

**IN cbfunc**  
Callback function [pmix\\_setup\\_application\\_cbfunc\\_t](#) (function reference)

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

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

1           **Description**

2           Provide a function by which the RM can request application-specific setup data (e.g., environmental  
3           variables, fabric configuration and security credentials) from supporting PMIx server library  
4           subsystems prior to initiating launch of an application.

**Advice to PMIx server hosts**

5           Host environments are *required* to execute this operation prior to launching an application.

6           This is defined as a non-blocking operation in case contributing subsystems need to perform some  
7           potentially time consuming action (e.g., query a remote service) before responding. The returned  
8           data must be distributed by the RM and subsequently delivered to the local PMIx server on each  
9           node where application processes will execute prior to initiating execution of those processes.

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

**Advice to PMIx library implementers**

12          Support for harvesting of environmental variables and providing of local configuration information  
13          by the PMIx implementation is optional.

14   **10.1.10 PMIx\_server\_setup\_local\_support**

15           **Summary**

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

1

## Format

PMIx v2.0

C

2

`pmix_status_t`

3

```
PMIx_server_setup_local_support(const pmix_namespace_t nspace,
```

4

```
    pmix_info_t info[], size_t ninfo,
```

5

```
    pmix_op_cbfunc_t cbfunc,
```

6

```
    void *cbdata);
```

C

7

**IN** `nnamespace`

8

Namespace (string)

9

**IN** `info`

10

Array of info structures (array of handles)

11

**IN** `ninfo`

12

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

13

**IN** `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

Provide a function by which the local PMIx server can perform any application-specific operations

20

prior to spawning local clients of a given application. For example, a network library might need to

21

setup the local driver for “instant on” addressing. The data provided in the *info* array is the data

22

provided to there host RM from the a call to `PMIx_server_setup_application`.

### Advice to PMIx server hosts

23

Host environments are *required* to execute this operation prior to starting any local application

24

processes from the specified namespace.

## 1 10.2 Server Function Pointers

2 PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the  
3 protocol. This method allows RMs to implement the server without being burdened with PMIx  
4 internal details. When a request is received from the client, the corresponding server function will  
5 be called with the information.

6 Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. Client  
7 calls to such functions will return a **PMIX\_ERR\_NOT\_SUPPORTED** status.

8 The host RM will provide the function pointers in a **pmix\_server\_module\_t** structure passed  
9 to **PMIx\_server\_init** . That module structure and associated function references are defined  
10 in this section.

### ▼ Advice to PMIx server hosts ▼

11 For performance purposes, the host server is required to return as quickly as possible from all  
12 functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx  
13 server support library to handle multiple client requests as quickly and scalably as possible.

14 *All* data passed to the host server functions is "owned" by the PMIX server support library and  
15 *MUST NOT* be free'd. Data returned by the host server via callback function is owned by the host  
16 server, which is free to release it upon return from the callback



### 17 10.2.1 pmix\_server\_module\_t Module

#### 18 Summary

19 List of function pointers that a PMIx server passes to **PMIx\_server\_init** during startup.

#### 20 Format



## C

```

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

```

## C

## 26 10.2.2 pmix\_server\_client\_connected\_fn\_t

### 27 Summary

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

### 29 Format

*PMIx v1.0*

## C

```

30 typedef pmix_status_t (*pmix_server_client_connected_fn_t) (
31     const pmix_proc_t *proc,
32     void* server_object,
33     pmix_op_cbfunc_t cbfunc,
34     void *cbdata)

```

```

1  IN  proc
2      pmix_proc_t structure (handle)
3  IN  server_object
4      object reference (memory reference)
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 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

## 10 Description

11 Notify the host environment that a client has called **PMIx\_Init** . Note that the client will be in a  
12 blocked state until the host server executes the callback function, thus allowing the PMIx server  
13 support library to release the client. The **server\_object** parameter will be the value of the  
14 **server\_object** parameter passed to **PMIx\_server\_register\_client** by the host server  
15 when registering the connecting client. If provided, an implementation of  
16 **pmix\_server\_client\_connected\_fn\_t** is only required to call the callback function  
17 designated. A host server can choose to not be notified when clients connect by setting  
18 **client\_connected** to **NULL**.

19 It is possible that only a subset of the clients in a namespace call **PMIx\_init** . The server's  
20 **pmix\_server\_client\_connected\_fn\_t** implementation should not depend on being  
21 called once per rank in a namespace or delay calling the callback function until all ranks have  
22 connected. However, if a rank makes any PMIx calls, it must first call **PMIx\_Init** and therefore  
23 the server's **pmix\_server\_client\_connected\_fn\_t** will be called before any other  
24 server functions specific to the rank.

### Advice to PMIx server hosts

25 This operation is an opportunity for a host environment to update the status of the ranks it manages.  
26 It is also a convenient and well defined time to perform initialization necessary to support further  
27 calls into the server related to that rank.

## 28 10.2.3 **pmix\_server\_client\_finalized\_fn\_t**

### 29 Summary

30 Notify the host environment that a client called **PMIx\_Finalize** .

1 **Format**

*PMIx v1.0*

C

```
2 typedef pmix_status_t (*pmix_server_client_finalized_fn_t) (  
3     const pmix_proc_t *proc,  
4     void* server_object,  
5     pmix_op_cbfunc_t cbfunc,  
6     void *cbdata)
```

C

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

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

16 **Description**

17 Notify the host environment that a client called **PMIx\_Finalize**. Note that the client will be in  
18 a blocked state until the host server executes the callback function, thus allowing the PMIx server  
19 support library to release the client. The `server_object` parameter will be the value of the  
20 `server_object` parameter passed to **PMIx\_server\_register\_client** by the host server  
21 when registering the connecting client. If provided, an implementation of  
22 `pmix_server_client_finalized_fn_t` is only required to call the callback function  
23 designated. A host server can choose to not be notified when clients finalize by setting  
24 `client_finalized` to **NULL**.

25 Note that the host server is only being informed that the client has called **PMIx\_Finalize**. The  
26 client might not have exited. If a client exits without calling **PMIx\_Finalize**, the server support  
27 library will not call the `pmix_server_client_finalized_fn_t` implementation.

Advice to PMIx server hosts

28 This operation is an opportunity for a host server to update the status of the tasks it manages. It is  
29 also a convenient and well defined time to release resources used to support that client.

## 1 10.2.4 pmix\_server\_abort\_fn\_t

### 2 Summary

3 Notify the host environment that a local client called `PMIx_Abort` .

### 4 Format

*PMIx v1.0*

C

```
5 typedef pmix_status_t (*pmix_server_abort_fn_t) (  
6     const pmix_proc_t *proc,  
7     void *server_object,  
8     int status,  
9     const char msg[],  
10    pmix_proc_t procs[],  
11    size_t nprocs,  
12    pmix_op_cbfunc_t cbfunc,  
13    void *cbdata)
```

C

14 **IN** `proc`  
15 `pmix_proc_t` structure identifying the process requesting the abort (handle)  
16 **IN** `server_object`  
17 object reference (memory reference)  
18 **IN** `status`  
19 exit status (integer)  
20 **IN** `msg`  
21 exit status message (string)  
22 **IN** `procs`  
23 Array of `pmix_proc_t` structures identifying the processes to be terminated (array of  
24 handles)  
25 **IN** `nprocs`  
26 Number of elements in the `procs` array (integer)  
27 **IN** `cbfunc`  
28 Callback function `pmix_op_cbfunc_t` (function reference)  
29 **IN** `cbdata`  
30 Data to be passed to the callback function (memory reference)

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

## Description

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

## 10.2.5 pmix\_server\_fenceb\_fn\_t

### Summary

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

### Format

PMIx v1.0

C

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

C

**IN procs**  
Array of **pmix\_proc\_t** structures identifying operation participants(array of handles)

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

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

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

**IN data**  
(string)

**IN ndata**  
(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.

## Required Attributes

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.

2 The following attributes are required to be supported by all host environments:

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

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

## Optional Attributes

5 The following attributes are optional for host environments:

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

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

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

11 Comma-delimited list of algorithms to use for the collective operation.

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

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

## Advice to PMIx server hosts

14 Host environment are *required* to return **PMIX\_ERR\_NOT\_SUPPORTED** if passed an attributed  
15 marked as **PMIX\_INFO\_REQD** that they do not support, even if support for that attribute is  
16 optional.

## Description

All local clients in the provided array of *procs* called either `PMIx_Fence` or `PMIx_Fence_nb`. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the `PMIx_Fence` / `PMIx_Fence_nb` operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_fence_nb_fn_t` function.

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

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

## 10.2.6 pmix\_server\_dmodex\_req\_fn\_t

### Summary

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

### Format

*PMIx v1.0*

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

**IN** `proc`  
`pmix_proc_t` structure identifying the process whose data is being requested (handle)

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

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

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

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

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

▼----- Required Attributes -----▼

4 PMIx libraries are required to pass any provided attributes to the host environment for processing.



▼----- Optional Attributes -----▼

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

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

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



10 **Description**

11 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
12 hosts the specified proc to obtain and return any information that process posted via calls to  
13 `PMIx_Put` and `PMIx_Commit`.

14 The array of *info* structs is used to pass user-requested options to the server. This can include a  
15 timeout to preclude an indefinite wait for data that may never become available. The directives are  
16 optional *unless* the *mandatory* flag has been set - in such cases, the host RM is required to return an  
17 error if the directive cannot be met.

18 **10.2.7 pmix\_server\_publish\_fn\_t**

19 **Summary**

20 Publish data per the PMIx API specification.



1

## Format

PMIx v1.0

C

2

```

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)

```

3

4

5

6

7

C

8

**IN** `proc`

9

`pmix_proc_t` structure of the process publishing the data (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

**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.

### Required Attributes

19

PMIx libraries are required to pass any provided attributes to the host environment for processing.

20

In addition, the following attributes are required to be included in the passed `info` array:

21

**PMIX\_USERID** "`pmix.euid`" (`uint32_t`)

22

Effective user id.

23

**PMIX\_GRPID** "`pmix.egid`" (`uint32_t`)

24

Effective group id.

25

Host environments that implement this entry point are required to support the following attributes:

26

**PMIX\_RANGE** "`pmix.range`" (`pmix_data_range_t`)

27

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

28

**PMIX\_PERSISTENCE** "`pmix.persist`" (`pmix_persistence_t`)

29

Value for calls to `PMIx_Publish`.

## Optional Attributes

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

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

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

### Description

Publish data per the **PMIx\_Publish** specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be **PMIX\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION** or their equivalent. These values can be specified by including the respective attributed in the *info* array.

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

### Advice to PMIx server hosts

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

## 10.2.8 pmix\_server\_lookup\_fn\_t

### Summary

Lookup published data.

1

## Format

PMIx v1.0

C

```

2 typedef pmix_status_t (*pmix_server_lookup_fn_t) (
3     const pmix_proc_t *proc,
4     char **keys,
5     const pmix_info_t info[],
6     size_t ninfo,
7     pmix_lookup_cbfunc_t cbfunc,
8     void *cbdata)

```

C

9 **IN proc**  
 10 `pmix_proc_t` structure of the process seeking the data (handle)

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

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

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

17 **IN cbfunc**  
 18 Callback function `pmix_lookup_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.

### Required Attributes

22 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
 23 In addition, the following attributes are required to be included in the passed *info* array:

24 **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
 25 Effective user id.

26 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)  
 27 Effective group id.

28 Host environments that implement this entry point are required to support the following attributes:

29 **PMIX\_RANGE** "pmix.range" (`pmix_data_range_t`)  
 30 Value for calls to publish/lookup/unpublish or for monitoring event notifications.

31 **PMIX\_WAIT** "pmix.wait" (`int`)  
 32 Caller requests that the PMIx server wait until at least the specified number of values are  
 33 found (0 indicates all and is the default).

## Optional Attributes

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

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

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

### Description

Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested.

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

### Advice to PMIx server hosts

The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

## 10.2.9 pmix\_server\_unpublish\_fn\_t

### Summary

Delete data from the data store.

1

## Format

PMIx v1.0

C

```

2 typedef pmix_status_t (*pmix_server_unpublish_fn_t) (
3     const pmix_proc_t *proc,
4     char **keys,
5     const pmix_info_t info[],
6     size_t ninfo,
7     pmix_op_cbfunc_t cbfunc,
8     void *cbdata)

```

C

9 **IN** **proc**  
10 `pmix_proc_t` structure identifying the process making the request (handle)

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

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

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

17 **IN** **cbfunc**  
18 Callback function `pmix_op_cbfunc_t` (function reference)

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

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

### Required Attributes

22 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
23 In addition, the following attributes are required to be included in the passed *info* array:

24 **PMIX\_USERID** "`pmix.euid`" (`uint32_t`)  
25 Effective user id.

26 **PMIX\_GRPID** "`pmix.egid`" (`uint32_t`)  
27 Effective group id.

28 Host environments that implement this entry point are required to support the following attributes:

29 **PMIX\_RANGE** "`pmix.range`" (`pmix_data_range_t`)  
30 Value for calls to publish/lookup/unpublish or for monitoring event notifications.

## Optional Attributes

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

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

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

### Description

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

### Advice to PMIx server hosts

The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

## 10.2.10 pmix\_server\_spawn\_fn\_t

### Summary

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

1

## Format

PMIx v1.0

C

2

```

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)

```

3

4

5

6

7

8

9

C

10

**IN** `proc`

`pmix_proc_t` structure of the process making the request (handle)

11

12

**IN** `job_info`

Array of info structures (array of handles)

13

14

**IN** `ninfo`

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

15

16

**IN** `apps`

Array of `pmix_app_t` structures (array of handles)

17

18

**IN** `napps`

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

19

20

**IN** `cbfunc`

Callback function `pmix_spawn_cbfunc_t` (function reference)

21

22

**IN** `cbdata`

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

23

24

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

### Required Attributes

25

PMIx libraries are required to pass any provided attributes to the host environment for processing.

26

In addition, the following attributes are required to be included in the passed `info` array:

27

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user id.

28

29

**PMIX\_GRPID** "pmix.egid" (`uint32_t`)

Effective group id.

30

1 Host environments that provide this module entry point are required to pass the `PMIX_SPAWNED`  
2 and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child processes so those  
3 values can be returned to clients upon connection to the PMIx server. In addition, they are required  
4 to support the following attributes when present in either the `job_info` or the `info` array of an  
5 element of the `apps` array:

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

7 Working directory for spawned processes.

8 `PMIX_SET_SESSION_CWD` "pmix.ssn cwd" (bool)

9 Set the application's current working directory to the session working directory assigned by  
10 the RM.

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

12 Prefix to use for starting spawned processes.

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

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

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

16 Hostfile to use for spawned processes.



17  **Optional Attributes** 

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

19 `PMIX_ADD_HOSTFILE` "pmix.addhostfile" (char\*)

20 Hostfile listing hosts to add to existing allocation.

21 `PMIX_ADD_HOST` "pmix.addhost" (char\*)

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

23 `PMIX_PRELOAD_BIN` "pmix.preloadbin" (bool)

24 Preload binaries onto nodes.

25 `PMIX_PRELOAD_FILES` "pmix.preloadfiles" (char\*)

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

27 `PMIX_PERSONALITY` "pmix.pers" (char\*)

28 Name of personality to use.

29 `PMIX_MAPPER` "pmix.mapper" (char\*)

30 Mapping mechanism to use for placing spawned processes.

31 `PMIX_DISPLAY_MAP` "pmix.dispmap" (bool)

32 Display process mapping upon spawn.

33 `PMIX_PPR` "pmix.ppr" (char\*)

Number of processes to spawn on each identified resource.



1 **PMIX\_MAPBY** "pmix.mapby" (char\*)  
 2 Process mapping policy.

3 **PMIX\_RANKBY** "pmix.rankby" (char\*)  
 4 Process ranking policy.

5 **PMIX\_BINDTO** "pmix.bindto" (char\*)  
 6 Process binding policy.

7 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)  
 8 Spawned processes will not call **PMIx\_Init** .

9 **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)  
 10 Spawned process rank that is to receive **stdin**.

11 **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (bool)  
 12 Forward this process's **stdin** to the designated process.

13 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
 14 Forward **stdout** from spawned processes to this process.

15 **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
 16 Forward **stderr** from spawned processes to this process.

17 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)  
 18 Spawned application consists of debugger daemons.

19 **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)  
 20 Tag application output with the identity of the source process.

21 **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)  
 22 Timestamp output from applications.

23 **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)  
 24 Merge **stdout** and **stderr** streams from application processes.

25 **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)  
 26 Output application output to the specified file.

27 **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)  
 28 Mark the **argv** with the rank of the process.

29 **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)  
 30 Number of cpus to assign to each rank.

31 **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)  
 32 Do not place processes on the head node.

33 **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
 34 Do not oversubscribe the cpus.

35 **PMIX\_REPORT\_BINDINGS** "pmix.repbinding" (bool)

1 Report bindings of the individual processes.

2 **PMIX\_CPU\_LIST** "pmix.cpulist" (char\*)

3 List of cpus to use for this job.

4 **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)

5 Application supports recoverable operations.

6 **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)

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

8 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)

9 Maximum number of times to restart a job.

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

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



## 14 Description

15 Spawn a set of applications/processes as per the **PMIx\_Spawn** API. Note that applications are not  
16 required to be MPI or any other programming model. Thus, the host server cannot make any  
17 assumptions as to their required support. The callback function is to be executed once all processes  
18 have been started. An error in starting any application or process in this request shall cause all  
19 applications and processes in the request to be terminated, and an error returned to the originating  
20 caller.

21 Note that a timeout can be specified in the job\_info array to indicate that failure to start the  
22 requested job within the given time should result in termination to avoid hangs.

## 23 10.2.11 pmix\_server\_connect\_fn\_t

### 24 Summary

25 Record the specified processes as *connected*.

1

## Format

PMIx v1.0

C

2

```

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

```

3

4

5

6

7

8

C

9

**IN procs**

Array of `pmix_proc_t` structures identifying participants (array of handles)

10

11

**IN nprocs**

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

12

13

**IN info**

Array of info structures (array of handles)

14

15

**IN ninfo**

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

16

17

**IN cbfunc**

Callback function `pmix_op_cbfunc_t` (function reference)

18

19

**IN cbdata**

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

20

21

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

### Required Attributes

22

PMIx libraries are required to pass any provided attributes to the host environment for processing.

### Optional Attributes

23

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

24

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

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

25

26

27

28

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

Comma-delimited list of algorithms to use for the collective operation.

29

30

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

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

31

## Description

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_connect_fn_t` function, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

### Advice to PMIx server hosts

The PMIx server library will call this function once all local participants have called `PMIx_Connect` or its non-blocking form with the same array of participants.

## 10.2.12 pmix\_server\_disconnect\_fn\_t

### Summary

Disconnect a previously connected set of processes.

### Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_disconnect_fn_t) (  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

C

#### IN `procs`

Array of `pmix_proc_t` structures identifying participants (array of handles)

#### IN `nprocs`

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

#### IN `info`

Array of info structures (array of handles)

#### IN `ninfo`

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

#### IN `cbfunc`

Callback function `pmix_op_cbfunc_t` (function reference)

#### IN `cbdata`

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

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

## Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

## Optional Attributes

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

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

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

## Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server’s has called the `pmix_server_disconnect_fn_t` function, *and* the host environment has completed any required supporting operations.

## Advice to PMIx server hosts

A `PMIX_ERR_INVALID_OPERATION` error must be returned if the specified set of *procs* was not previously *connected* via a call to the `pmix_server_connect_fn_t` function.

The PMIx server library will call this function once all local participants have called `PMIx_Disconnect` or its non-blocking form with the same array of participants.

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

### Summary

Register to receive notifications for the specified events.

1

## Format

PMIx v1.0

C

2

```

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

```

3

4

5

6

7

8

C

9

**IN codes**

Array of `pmix_status_t` values (array of handles)

10

11

**IN ncodes**

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

12

13

**IN info**

Array of info structures (array of handles)

14

15

**IN ninfo**

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

16

17

**IN cbfunc**

Callback function `pmix_op_cbfunc_t` (function reference)

18

19

**IN cbdata**

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

20

21

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

### Required Attributes

22

PMIx libraries are required to pass any provided attributes to the host environment for processing.

23

In addition, the following attributes are required to be included in the passed `info` array:

24

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user id.

25

26

**PMIX\_GRPID** "pmix.egid" (`uint32_t`)

Effective group id.

27

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17

## Description

Register to receive notifications for the specified status codes. The *info* array included in this API is reserved for possible future directives to further steer notification.

### Advice to PMIx library implementers

The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

- the client has requested notification of an environmental code (i.e., a PMIx code in the range between `PMIX_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER`, inclusive) or a code that lies outside the defined PMIx range of constants; and
- the PMIx server library has not previously requested notification of that code - i.e., the host environment is to be contacted only once a given unique code value

### Advice to PMIx server hosts

The host environment is *required* to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between `PMIX_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER` (inclusive).

## 10.2.14 pmix\_server\_deregister\_events\_fn\_t

### Summary

Deregister to receive notifications for the specified events.

1 **Format**

*PMIx v1.0*

C

```
2 typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (  
3     pmix_status_t *codes,  
4     size_t ncodes,  
5     pmix_op_cbfunc_t cbfunc,  
6     void *cbdata)
```

C

- 7 **IN codes**  
8 Array of `pmix_status_t` values (array of handles)
- 9 **IN ncodes**  
10 Number of elements in the `codes` array (integer)
- 11 **IN cbfunc**  
12 Callback function `pmix_op_cbfunc_t` (function reference)
- 13 **IN cbdata**  
14 Data to be passed to the callback function (memory reference)

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

16 **Description**

17 Deregister to receive notifications for the specified events to which the PMIx server has previously  
18 registered.

Advice to PMIx library implementers

19 The PMIx server library must track all client registrations. This module function shall only be  
20 called when:

- 21 • the library is deregistering environmental codes (i.e., a PMIx codes in the range between  
22 `PMIX_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER`, inclusive) or codes that lies outside  
23 the defined PMIx range of constants; and
- 24 • no client (including the server library itself) remains registered for notifications on any included  
25 code - i.e., a code should be included in this call only when no registered notifications against it  
26 remain.

27 **10.2.15 pmix\_server\_notify\_event\_fn\_t**

28 **Summary**

29 Notify the specified processes of an event.



1

## Format

PMIx v2.0

C

2

```

typedef pmix_status_t (*pmix_server_notify_event_fn_t) (pmix_status_t code,
3
4
5
6
7
8
               const pmix_proc_t *source,
               pmix_data_range_t range,
               pmix_info_t info[],
               size_t ninfo,
               pmix_op_cbfunc_t cbfunc,
               void *cbdata);

```

C

9

**IN code**

The `pmix_status_t` event code being referenced structure (handle)

10

11

**IN source**

`pmix_proc_t` of process that generated the event (handle)

12

13

**IN range**

`pmix_data_range_t` range over which the event is to be distributed (handle)

14

15

**IN info**

Optional array of `pmix_info_t` structures containing additional information on the event (array of handles)

16

17

18

**IN ninfo**

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

19

20

**IN cbfunc**

Callback function `pmix_op_cbfunc_t` (function reference)

21

22

**IN cbdata**

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

23

24

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

### Required Attributes

25

PMIx libraries are required to pass any provided attributes to the host environment for processing.

26

Host environments that provide this module entry point are required to support the following attributes:

27

28

**PMIX\_RANGE** "`pmix.range`" (`pmix_data_range_t`)

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

29

## Description

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

### Advice to PMIx server hosts

The callback function is to be executed once the host environment no longer requires that the PMIx server library maintain the provided data structures. It does *not* necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery

## 10.2.16 pmix\_server\_listener\_fn\_t

### Summary

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

### Format

PMIx v1.0

```
typedef pmix_status_t (*pmix_server_listener_fn_t) (  
    int listening_sd,  
    pmix_connection_cbfunc_t cbfunc,  
    void *cbdata)
```

**IN** `incoming_sd`  
(integer)

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

**IN** `cbdata`  
(memory reference)

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

### Description

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

## 1 10.2.17 pmix\_server\_query\_fn\_t

### 2 Summary

3 Query information from the resource manager.

### 4 Format

PMIx v2.0

```
▼────────────────────────────────────────── C ───────────────────────────────────▼  
5 typedef pmix_status_t (*pmix_server_query_fn_t) (  
6     pmix_proc_t *proct,  
7     pmix_query_t *queries, size_t nqueries,  
8     pmix_info_cbfunc_t cbfunc,  
9     void *cbdata)  
▲────────────────────────────────────────── C ───────────────────────────────────▲
```

- 10 **IN** `proct`  
11 `pmix_proc_t` structure of the requesting process (handle)
- 12 **IN** `queries`  
13 Array of `pmix_query_t` structures (array of handles)
- 14 **IN** `nqueries`  
15 Number of elements in the `queries` array (integer)
- 16 **IN** `cbfunc`  
17 Callback function `pmix_info_cbfunc_t` (function reference)
- 18 **IN** `cbdata`  
19 Data to be passed to the callback function (memory reference)

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

### ▼----- Required Attributes -----▼

21 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
22 In addition, the following attributes are required to be included in the passed `info` array:

- 23 **PMIX\_USERID** "`pmix.euid`" (`uint32_t`)  
24 Effective user id.
- 25 **PMIX\_GRPID** "`pmix.egid`" (`uint32_t`)  
26 Effective group id.

▲-----

## Optional Attributes

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

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

Request a comma-delimited list of active namespaces.

**PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (pmix\_status\_t)

Status of a specified, currently executing job.

**PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (char\*)

Request a comma-delimited list of scheduler queues.

**PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (TBD)

Status of a specified scheduler queue.

**PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (char\*)

Input namespace of the job whose information is being requested returns (**pmix\_data\_array\_t**) an array of **pmix\_proc\_info\_t**.

**PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)

Input namespace of the job whose information is being requested returns (**pmix\_data\_array\_t**) an array of **pmix\_proc\_info\_t** for processes in job on same node.

**PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)

Return a comma-delimited list of supported spawn attributes.

**PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)

Return a comma-delimited list of supported debug attributes.

**PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)

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

**PMIX\_QUERY\_LOCAL\_ONLY** "pmix.qry.local" (bool)

Constrain the query to local information only.

**PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)

Report average values.

**PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)

Report minimum and maximum values.

**PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)

String identifier of the allocation whose status is being requested.

**PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)

Query number of seconds (**uint32\_t**) remaining in allocation for the specified namespace.

## Description

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

### Advice to PMIx library implementers

The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

## 10.2.18 `pmix_server_tool_connection_fn_t`

### Summary

Register that a tool has connected to the server.

### Format

*PMIx v2.0*

```
typedef void (*pmix_server_tool_connection_fn_t) (  
    pmix_info_t info[], size_t ninfo,  
    pmix_tool_connection_cbfnc_t cbfunc,  
    void *cbdata)
```

#### IN `info`

Array of `pmix_info_t` structures (array of handles)

#### IN `ninfo`

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

#### IN `cbfunc`

Callback function `pmix_tool_connection_cbfnc_t` (function reference)

#### IN `cbdata`

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

### Required Attributes

PMIx libraries are required to pass the following attributes in the *info* array:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user id.

**PMIX\_GRPID** "pmix.egid" (`uint32_t`)

Effective group id.

## Optional Attributes

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

**PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
Forward **stdout** from spawned processes to this process.

**PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
Forward **stderr** from spawned processes to this process.

**PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (bool)  
Forward this process's **stdin** to the designated process.

## Description

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The `pmix_info_t` array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

### Advice to PMIx server hosts

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests.

## 10.2.19 pmix\_server\_log\_fn\_t

### Summary

Log data on behalf of a client.

1 **Format**

PMIx v2.0

C

```
2 typedef void (*pmix_server_log_fn_t) (  
3     const pmix_proc_t *client,  
4     const pmix_info_t data[], size_t ndata,  
5     const pmix_info_t directives[], size_t ndirs,  
6     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

- 7 **IN client**  
8 `pmix_proc_t` structure (handle)
- 9 **IN data**  
10 Array of info structures (array of handles)
- 11 **IN ndata**  
12 Number of elements in the *data* array (integer)
- 13 **IN directives**  
14 Array of info structures (array of handles)
- 15 **IN ndirs**  
16 Number of elements in the *directives* array (integer)
- 17 **IN cbfunc**  
18 Callback function `pmix_op_cbfunc_t` (function reference)
- 19 **IN cbdata**  
20 Data to be passed to the callback function (memory reference)

Required Attributes

21 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
22 In addition, the following attributes are required to be included in the passed *info* array:

23 **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
24 Effective user id.

25 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)  
26 Effective group id.

27 Host environments that provide this module entry point are required to support the following  
28 attributes:

29 **PMIX\_LOG\_STDERR** "pmix.log.stderr" (`char*`)  
30 Log string to `stderr`.

31 **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (`char*`)  
32 Log string to `stdout`.

33 **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (`char*`)  
34 Log data to syslog. Defaults to **ERROR** priority.

## Optional Attributes

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

**PMIX\_LOG\_MSG** "pmix.log.msg" (`pmix_byte_object_t`)

Message blob to be sent somewhere.

**PMIX\_LOG\_EMAIL** "pmix.log.email" (`pmix_data_array_t`)

Log via email based on `pmix_info_t` containing directives.

**PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (`char*`)

Comma-delimited list of email addresses that are to receive the message.

**PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (`char*`)

Subject line for email.

**PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (`char*`)

Message to be included in email.

### Description

Log data on behalf of a client. This function is *not* intended for output of computational results, but rather for reporting status and error messages.

## 10.2.20 pmix\_server\_alloc\_fn\_t

### Summary

Request allocation operations on behalf of a client.



1

## Format

PMIx v2.0

C

```

2 typedef pmix_status_t (*pmix_server_alloc_fn_t) (
3     const pmix_proc_t *client,
4     pmix_alloc_directive_t directive,
5     const pmix_info_t data[], size_t ndata,
6     pmix_info_cbfunc_t cbfunc, void *cbdata)

```

C

7 **IN client**  
 8 `pmix_proc_t` structure of process making request (handle)

9 **IN directive**  
 10 Specific action being requested (`pmix_alloc_directive_t`)

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

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

15 **IN cbfunc**  
 16 Callback function `pmix_info_cbfunc_t` (function reference)

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

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

### Required Attributes

20 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
 21 In addition, the following attributes are required to be included in the passed *info* array:

22 **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
 23 Effective user id.

24 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)  
 25 Effective group id.

26 Host environments that provide this module entry point are required to support the following  
 27 attributes:

28 **PMIX\_ALLOC\_ID** "pmix.alloc.id" (`char*`)  
 29 Provide a string identifier for this allocation request which can later be used to query status  
 30 of the request.

31 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (`uint64_t`)  
 32 The number of nodes.

33 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (`uint64_t`)  
 34 Number of cpus.

1 **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)  
2 Time in seconds.



▼----- Optional Attributes -----▼

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

4 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)  
5 Regular expression of the specific nodes.

6 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)  
7 Regular expression of the number of cpus for each node.

8 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)  
9 Regular expression of the specific cpus indicating the cpus involved.

10 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)  
11 Number of Megabytes.

12 **PMIX\_ALLOC\_NETWORK** "pmix.alloc.net" (array)  
13 Array of **pmix\_info\_t** describing requested network resources. If not given as part of an  
14 **pmix\_info\_t** struct that identifies the involved nodes, then the description will be  
15 applied across all nodes in the requestor's allocation.

16 **PMIX\_ALLOC\_NETWORK\_ID** "pmix.alloc.netid" (char\*)  
17 Name of the network.

18 **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)  
19 Mbits/sec.

20 **PMIX\_ALLOC\_NETWORK\_QOS** "pmix.alloc.netqos" (char\*)  
21 Quality of service level.



## Description

Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an existing allocation. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to the scheduler with a promise to return them upon subsequent request.

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

## 10.2.21 pmix\_server\_job\_control\_fn\_t

### Summary

Execute a job control action on behalf of a client.

### Format

PMIx v2.0

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

**IN requestor**  
`pmix_proc_t` structure of requesting process (handle)

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

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

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

1 **IN ndirs**  
2     Number of elements in the *info* array (integer)  
3 **IN cbfunc**  
4     Callback function `pmix_op_cbfunc_t` (function reference)  
5 **IN cbdata**  
6     Data to be passed to the callback function (memory reference)

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

▼----- Required Attributes -----▼

8 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
9 In addition, the following attributes are required to be included in the passed *info* array:

10 **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
11     Effective user id.

12 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)  
13     Effective group id.

14 Host environments that provide this module entry point are required to support the following  
15 attributes:

16 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (`char*`)  
17     Provide a string identifier for this request.

18 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (`bool`)  
19     Pause the specified processes.

20 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (`bool`)  
21     Resume ("un-pause") the specified processes.

22 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (`bool`)  
23     Forcibly terminate the specified processes and cleanup.

24 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (`int`)  
25     Send given signal to specified processes.

26 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (`bool`)  
27     Politely terminate the specified processes.



## Optional Attributes

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

- PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)  
Cancel the specified request (**NULL** implies cancel all requests from this requestor).
- PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)  
Restart the specified processes using the given checkpoint ID.
- PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)  
Checkpoint the specified processes and assign the given ID to it.
- PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)  
Use event notification to trigger a process checkpoint.
- PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)  
Use the given signal to trigger a process checkpoint.
- PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)  
Time in seconds to wait for a checkpoint to complete.
- PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
"pmix.jctrl.ckmethod" (pmix\_data\_array\_t)  
Array of **pmix\_info\_t** declaring each method and value supported by this application.
- PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)  
Regular expression identifying nodes that are to be provisioned.
- PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvning" (char\*)  
Name of the image that is to be provisioned.
- PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)  
Indicate that the job can be pre-empted.

## Description

Execute a job control action on behalf of a client. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX\_RANK\_WILDARD** can also be used to indicate that all processes in the given namespace are to be included.

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

## 1 10.2.22 pmix\_server\_monitor\_fn\_t

### 2 Summary

3 Request that a client be monitored for activity.

### 4 Format

PMIx v2.0

C

```
5 /* Request that a client be monitored for activity */
6 typedef pmix_status_t (*pmix_server_monitor_fn_t) (
7     const pmix_proc_t *requestor,
8     const pmix_info_t *monitor, pmix_status_t error
9     const pmix_info_t directives[], size_t ndirs,
10    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

11 **IN requestor**  
12 `pmix_proc_t` structure of requesting process (handle)

13 **IN monitor**  
14 `pmix_info_t` identifying the type of monitor being requested (handle)

15 **IN error**  
16 Status code to use in generating event if alarm triggers (integer)

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

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

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

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

25 Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant. This entry  
26 point is only called for monitoring requests that are not directly supported by the PRI.

### Required Attributes

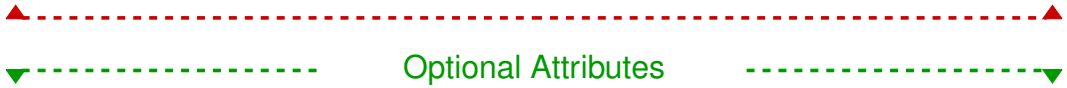
27 If supported by the PMIx server library, then the library must not pass any supported attributes to  
28 the host environment. All attributes not directly supported by the server library must be passed to  
29 the host environment if it provides this module entry. In addition, the following attributes are  
30 required to be included in the passed *info* array:

31 **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
32 Effective user id.

33 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)

1           Effective group id.

2           Host environments are not required to support any specific monitoring attributes.



3           The following attributes may be implemented by a host environment.

4           **PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)

5           Provide a string identifier for this request.

6           **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)

7           Identifier to be canceled (**NULL** means cancel all monitoring for this process).

8           **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)

9           The application desires to control the response to a monitoring event.

10          **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)

11          Register to have the PMIx server monitor the requestor for heartbeats.

12          **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)

13          Time in seconds before declaring heartbeat missed.

14          **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)

15          Number of heartbeats that can be missed before generating the event.

16          **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)

17          Register to monitor file for signs of life.

18          **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)

19          Monitor size of given file is growing to determine if the application is running.

20          **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (char\*)

21          Monitor time since last access of given file to determine if the application is running.

22          **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)

23          Monitor time since last modified of given file to determine if the application is running.

24          **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)

25          Time in seconds between checking the file.

26          **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)

27          Number of file checks that can be missed before generating the event.



1  
2  
3  
4  
5

## Description

Request that a client be monitored for activity.

▼ **Advice to PMIx server hosts** ▼

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested services or return `PMIX_ERR_NOT_SUPPORTED` to the provided *cbfunc*.

▲



## APPENDIX A

# 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 **A.1 Version 2.0**

6 The following list includes some of the active participants in the PMIx v2 standardization process.

- 7 • Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm  
8 and Terry Wilmarth
- 9 • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
- 10 • Aurelien Bouteiller and George Bosilca
- 11 • Artem Polyakov, Igor Ivanov and Boris Karasev
- 12 • Gilles Gouaillardet
- 13 • Michael A Raymond and Jim Stoffel
- 14 • Dirk Schubert
- 15 • Moe Jette
- 16 • Takahiro Kawashima and Shinji Sumimoto
- 17 • Howard Pritchard
- 18 • David Beer
- 19 • Brice Goglin
- 20 • Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
- 21 • Adam Moody and Martin Schulz
- 22 • Ryan Grant and Stephen Olivier
- 23 • Michael Karo

1 The following institutions supported this effort through time and travel support for the people listed  
2 above.

- 3 • Intel Corporation
- 4 • IBM, Inc.
- 5 • University of Tennessee, Knoxville
- 6 • The Exascale Computing Project, an initiative of the US Department of Energy
- 7 • National Science Foundation
- 8 • Mellanox, Inc.
- 9 • Research Organization for Information Science and Technology
- 10 • HPE Co.
- 11 • Allinea (ARM)
- 12 • SchedMD, Inc.
- 13 • Fujitsu Limited
- 14 • Los Alamos National Laboratory
- 15 • Adaptive Solutions, Inc.
- 16 • INRIA
- 17 • Oak Ridge National Laboratory
- 18 • Lawrence Livermore National Laboratory
- 19 • Sandia National Laboratory
- 20 • Altair

## 21 **A.2 Version 1.0**

22 The following list includes some of the active participants in the PMIx v1 standardization process.

- 23 • Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- 24 • Joshua Hursey and David Solt
- 25 • Aurelien Bouteiller and George Bosilca
- 26 • Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- 27 • Gilles Gouaillardet
- 28 • Gary Brown

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

- Moe Jette

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology
- Adaptive Solutions, Inc.
- SchedMD, Inc.

# Bibliography

---

- [1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMix: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting, EuroMPI '17*, pages 14:1–14:10, New York, NY, USA, 2017. ACM.

# Index

---

- client\_connected, 185
- client\_finalized, 186
  
- mpix\_server\_client\_connected\_fn\_t, 185
  
- PMIx\_Abort, 8, 24, 118, 187, 188
  - Defintion, 117
- PMIX\_ADD\_HOST, 120, 124, 199
  - Defintion, 61
- PMIX\_ADD\_HOSTFILE, 120, 124, 199
  - Defintion, 61
- PMIX\_ALLOC\_BANDWIDTH, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_CPU\_LIST, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_DIRECTIVE, 50
- PMIx\_Alloc\_directive\_string, 9
  - Defintion, 81
- pmix\_alloc\_directive\_t, 36, 50, 81, 216
  - Defintion, 36
- PMIX\_ALLOC\_EXTEND, 36
- PMIX\_ALLOC\_EXTERNAL, 36
- PMIX\_ALLOC\_ID, 140, 216
  - Defintion, 65
- PMIX\_ALLOC\_MEM\_SIZE, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_NETWORK, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_NETWORK\_ID, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_NETWORK\_QOS, 141, 217
  - Defintion, 66
- PMIX\_ALLOC\_NEW, 36
- PMIX\_ALLOC\_NODE\_LIST, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_NUM\_CPU\_LIST, 140, 217
  - Defintion, 65
- PMIX\_ALLOC\_NUM\_CPUS, 140, 216
  - Defintion, 65
- PMIX\_ALLOC\_NUM\_NODES, 140, 216
  - Defintion, 65
- PMIX\_ALLOC\_REAQUIRE, 36
- PMIX\_ALLOC\_RELEASE, 36
- PMIX\_ALLOC\_TIME, 140, 217
  - Defintion, 66
- PMIX\_ALLOCATED\_NODELIST, 174
  - Defintion, 55
- PMIx\_Allocation\_request\_nb, 9, 134, 141
  - Defintion, 139
- PMIX\_ANL\_MAP
  - Defintion, 59
- PMIX\_APP, 50
- PMIX\_APP\_CONSTRUCT
  - Defintion, 40
- PMIX\_APP\_CREATE
  - Defintion, 41
- PMIX\_APP\_DESTRUCT
  - Defintion, 41
- PMIX\_APP\_FREE
  - Defintion, 41
- PMIX\_APP\_MAP\_REGEX
  - Defintion, 59
- PMIX\_APP\_MAP\_TYPE
  - Defintion, 59
- PMIX\_APP\_RANK, 173
  - Defintion, 55
- PMIX\_APP\_SIZE, 173, 174
  - Defintion, 56
- pmix\_app\_t, 40, 41, 119, 123, 198
  - Defintion, 40
- PMIX\_APPLDR, 173
  - Defintion, 55

PMIX\_APPNUM, 173  
     Defintion, **55**  
 PMIX\_ARCH  
     Defintion, **54**  
 PMIX\_ATTR\_UNDEF  
     Defintion, **51**  
 PMIX\_AVAIL\_PHYS\_MEMORY, 174  
     Defintion, **56**  
 PMIX\_BINDTO, 120, 124, 174, 200  
     Defintion, **62**  
 PMIX\_BOOL, 49  
 PMIX\_BUFFER, 50  
 PMIX\_BYTE, 49  
 PMIX\_BYTE\_OBJECT, 50  
 PMIX\_BYTE\_OBJECT\_CREATE  
     Defintion, **46**  
 PMIX\_BYTE\_OBJECT\_DESTRUCT  
     Defintion, **46**  
 PMIX\_BYTE\_OBJECT\_FREE  
     Defintion, **46**  
 PMIX\_BYTE\_OBJECT\_LOAD  
     Defintion, **47**  
 pmix\_byte\_object\_t, 45–47, 50  
     Defintion, **45**  
 PMIX\_CLIENT\_AVG\_MEMORY  
     Defintion, **56**  
 PMIX\_COLLECT\_DATA, 102, 104, 189  
     Defintion, **57**  
 PMIX\_COLLECTIVE\_ALGO, 103, 104,  
     128, 130, 189, 202  
     Defintion, **58**  
 PMIX\_COLLECTIVE\_ALGO\_REQD, 103,  
     104, 128, 130, 189, 202  
     Defintion, **58**  
 PMIX\_COMMAND, 50  
 PMIx\_Commit, 8, 78, 96, 102, 126, 179, 191  
     Defintion, **101**  
 PMIX\_COMPRESSED\_STRING, 50  
 PMIx\_Connect, 8, 18, 122, 128–130, 132,  
     203  
     Defintion, **127**  
 PMIX\_CONNECT\_MAX\_RETRIES, 88  
     Defintion, **52**  
 PMIx\_Connect\_nb, 8, 129  
     Defintion, **129**  
 PMIX\_CONNECT\_RETRY\_DELAY, 88  
     Defintion, **52**  
 PMIX\_CONNECT\_SYSTEM\_FIRST, 88,  
     90  
     Defintion, **52**  
 PMIX\_CONNECT\_TO\_SYSTEM, 88, 90  
     Defintion, **52**  
 pmix\_connection\_cbfunc\_t, 209  
     Defintion, **78**  
 PMIX\_COSPAWN\_APP  
     Defintion, **62**  
 PMIX\_CPU\_LIST, 121, 125, 201  
     Defintion, **63**  
 PMIX\_CPUS\_PER\_PROC, 121, 125, 200  
     Defintion, **62**  
 PMIX\_CPUSET  
     Defintion, **54**  
 PMIX\_CREDENTIAL  
     Defintion, **54**  
 PMIX\_DAEMON\_MEMORY  
     Defintion, **56**  
 PMIX\_DATA\_ARRAY, 50  
 pmix\_data\_array\_t, 28, 49, 50, 63, 137, 211  
     Defintion, **49**  
 PMIX\_DATA\_BUFFER\_CONSTRUCT,  
     163, 165  
     Defintion, **160**  
 PMIX\_DATA\_BUFFER\_CREATE, 163,  
     165  
     Defintion, **48, 159**  
 PMIX\_DATA\_BUFFER\_DESTRUCT  
     Defintion, **48, 160**  
 PMIX\_DATA\_BUFFER\_LOAD  
     Defintion, **161**  
 PMIX\_DATA\_BUFFER\_RELEASE  
     Defintion, **48, 160**  
 pmix\_data\_buffer\_t, 47, 48, 159–164, 168  
     Defintion, **47**  
 PMIX\_DATA\_BUFFER\_UNLOAD  
     Defintion, **161**  
 PMIx\_Data\_copy, 9

Defintion, [166](#)  
 PMIx\_Data\_copy\_payload, [9](#)  
     Defintion, [167](#)  
 PMIx\_Data\_pack, [9](#), [163](#)  
     Defintion, [162](#)  
 PMIx\_Data\_print, [9](#)  
     Defintion, [166](#)  
 PMIX\_DATA\_RANGE, [50](#)  
 PMIx\_Data\_range\_string, [9](#)  
     Defintion, [80](#)  
 pmix\_data\_range\_t, [27](#), [50](#), [80](#), [157](#), [208](#)  
     Defintion, [27](#)  
 PMIX\_DATA\_SCOPE, [97](#), [99](#)  
     Defintion, [58](#)  
 PMIX\_DATA\_TYPE, [50](#)  
 PMIX\_DATA\_TYPE\_MAX, [50](#)  
 PMIx\_Data\_type\_string, [9](#)  
     Defintion, [81](#)  
 pmix\_data\_type\_t, [30](#), [33](#), [39](#), [49](#), [81](#), [163](#),  
     [164](#), [166](#), [167](#)  
     Defintion, [49](#)  
 PMIx\_Data\_unpack, [9](#)  
     Defintion, [164](#)  
 PMIX\_DEBUG\_JOB  
     Defintion, [64](#)  
 PMIX\_DEBUG\_STOP\_IN\_INIT  
     Defintion, [64](#)  
 PMIX\_DEBUG\_STOP\_ON\_EXEC  
     Defintion, [64](#)  
 PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY  
     Defintion, [64](#)  
 PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY  
     Defintion, [64](#)  
 PMIX\_DEBUGGER\_DAEMONS, [121](#), [125](#),  
     [200](#)  
     Defintion, [62](#)  
 PMIx\_Deregister\_errhandler, [9](#)  
 PMIx\_Deregister\_event\_handler, [9](#)  
     Defintion, [155](#)  
 PMIx\_Disconnect, [8](#), [18](#), [129](#), [132](#), [133](#), [204](#)  
     Defintion, [130](#)  
 PMIx\_Disconnect\_nb, [8](#), [133](#)  
     Defintion, [132](#)  
 PMIX\_DISPLAY\_MAP, [120](#), [124](#), [199](#)  
     Defintion, [61](#)  
 pmix\_dmodex\_response\_fn\_t, [179](#)  
     Defintion, [77](#)  
 PMIX\_DOUBLE, [50](#)  
 PMIX\_DSTPATH  
     Defintion, [52](#)  
 PMIX\_EMBED\_BARRIER, [86](#)  
     Defintion, [58](#)  
 PMIX\_ERR\_BAD\_PARAM, [17](#)  
 PMIX\_ERR\_COMM\_FAILURE, [18](#)  
 PMIX\_ERR\_DATA\_VALUE\_NOT\_FOUND,  
     [17](#)  
 PMIX\_ERR\_DEBUGGER\_RELEASE, [17](#)  
 PMIX\_ERR\_EVENT\_REGISTRATION, [18](#)  
 PMIX\_ERR\_HANDSHAKE\_FAILED, [17](#)  
 PMIX\_ERR\_IN\_ERRNO, [17](#)  
 PMIX\_ERR\_INIT, [17](#)  
 PMIX\_ERR\_INVALID\_ARG, [17](#)  
 PMIX\_ERR\_INVALID\_ARGS, [18](#)  
 PMIX\_ERR\_INVALID\_CRED, [17](#)  
 PMIX\_ERR\_INVALID\_KEY, [17](#)  
 PMIX\_ERR\_INVALID\_KEY\_LENGTH, [17](#)  
 PMIX\_ERR\_INVALID\_KEYVALP, [18](#)  
 PMIX\_ERR\_INVALID\_LENGTH, [18](#)  
 PMIX\_ERR\_INVALID\_NAMESPACE, [18](#)  
 PMIX\_ERR\_INVALID\_NUM\_ARGS, [18](#)  
 PMIX\_ERR\_INVALID\_NUM\_PARSED, [18](#)  
 PMIX\_ERR\_INVALID\_OPERATION, [19](#)  
 PMIX\_ERR\_INVALID\_SIZE, [18](#)  
 PMIX\_ERR\_INVALID\_TERMINATION,  
     [18](#)  
 PMIX\_ERR\_INVALID\_VAL, [17](#)  
 PMIX\_ERR\_INVALID\_VAL\_LENGTH, [18](#)  
 PMIX\_ERR\_JOB\_TERMINATED, [18](#)  
 PMIX\_ERR\_LOST\_CONNECTION\_TO\_CLIENT,  
     [18](#)  
 PMIX\_ERR\_LOST\_CONNECTION\_TO\_SERVER,  
     [18](#)  
 PMIX\_ERR\_LOST\_PEER\_CONNECTION,  
     [18](#)  
 PMIX\_ERR\_NO\_PERMISSIONS, [17](#)  
 PMIX\_ERR\_NODE\_DOWN, [19](#)

PMIX\_ERR\_NODE\_OFFLINE, 19  
 PMIX\_ERR\_NOMEM, 17  
 PMIX\_ERR\_NOT\_FOUND, 18  
 PMIX\_ERR\_NOT\_IMPLEMENTED, 18  
 PMIX\_ERR\_NOT\_SUPPORTED, 18  
 PMIX\_ERR\_OUT\_OF\_RESOURCE, 17  
 PMIX\_ERR\_PACK\_FAILURE, 17  
 PMIX\_ERR\_PACK\_MISMATCH, 17  
 PMIX\_ERR\_PROC\_ABORTED, 17  
 PMIX\_ERR\_PROC\_ABORTING, 17  
 PMIX\_ERR\_PROC\_CHECKPOINT, 17  
 PMIX\_ERR\_PROC\_ENTRY\_NOT\_FOUND, 17  
 PMIX\_ERR\_PROC\_MIGRATE, 17  
 PMIX\_ERR\_PROC\_REQUESTED\_ABORT, 17  
 PMIX\_ERR\_PROC\_RESTART, 17  
 PMIX\_ERR\_READY\_FOR\_HANDSHAKE, 17  
 PMIX\_ERR\_RESOURCE\_BUSY, 17  
 PMIX\_ERR\_SERVER\_FAILED\_REQUEST, 17  
 PMIX\_ERR\_SERVER\_NOT\_AVAIL, 18  
 PMIX\_ERR\_SILENT, 17  
 PMIX\_ERR\_TIMEOUT, 17  
 PMIX\_ERR\_TYPE\_MISMATCH, 17  
 PMIX\_ERR\_UNKNOWN\_DATA\_TYPE, 17  
 PMIX\_ERR\_UNPACK\_FAILURE, 17  
 PMIX\_ERR\_UNPACK\_INADEQUATE\_SPACE, 17  
 PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER, 18  
 PMIX\_ERR\_UNREACH, 17  
 PMIX\_ERR\_UPDATE\_ENDPOINTS, 18  
 PMIX\_ERR\_WOULD\_BLOCK, 17  
 PMIX\_ERROR, 17  
 PMIX\_ERROR\_GROUP\_ABORT  
     Defintion, 59  
 PMIX\_ERROR\_GROUP\_COMM  
     Defintion, 59  
 PMIX\_ERROR\_GROUP\_GENERAL  
     Defintion, 60  
 PMIX\_ERROR\_GROUP\_LOCAL  
     Defintion, 60  
 PMIX\_ERROR\_GROUP\_MIGRATE  
     Defintion, 59  
 PMIX\_ERROR\_GROUP\_NODE  
     Defintion, 59  
 PMIX\_ERROR\_GROUP\_RESOURCE  
     Defintion, 59  
 PMIX\_ERROR\_GROUP\_SPAWN  
     Defintion, 59  
 PMIX\_ERROR\_HANDLER\_ID  
     Defintion, 60  
 PMIX\_ERROR\_NAME  
     Defintion, 59  
 PMIx\_Error\_string, 9  
     Defintion, 80  
 PMIX\_EVENT\_ACTION\_COMPLETE, 19  
 PMIX\_EVENT\_ACTION\_DEFERRED, 19  
 PMIX\_EVENT\_ACTION\_TIMEOUT, 154  
     Defintion, 61  
 PMIX\_EVENT\_AFFECTED\_PROC, 154, 157  
     Defintion, 60  
 PMIX\_EVENT\_AFFECTED\_PROCS, 154, 157  
     Defintion, 60  
 PMIX\_EVENT\_BASE, 85, 89, 93  
     Defintion, 51  
 PMIX\_EVENT\_CUSTOM\_RANGE, 153, 157  
     Defintion, 60  
 PMIX\_EVENT\_DO\_NOT\_CACHE  
     Defintion, 60  
 PMIX\_EVENT\_HDLR\_AFTER, 153  
     Defintion, 60  
 PMIX\_EVENT\_HDLR\_APPEND, 153  
     Defintion, 60  
 PMIX\_EVENT\_HDLR\_BEFORE, 153  
     Defintion, 60  
 PMIX\_EVENT\_HDLR\_FIRST, 153  
     Defintion, 60  
 PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY, 153



Defintion, [60](#)  
 PMIX\_EVENT\_HDLR\_LAST, [153](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY,  
     [153](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_HDLR\_NAME, [153](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_HDLR\_PREPEND, [153](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_NO\_ACTION\_TAKEN, [19](#)  
 PMIX\_EVENT\_NO\_TERMINATION  
     Defintion, [61](#)  
 PMIX\_EVENT\_NON\_DEFAULT, [157](#)  
     Defintion, [60](#)  
 pmix\_event\_notification\_cbfunc\_fn\_t, [73](#), [75](#)  
     Defintion, [73](#)  
 PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN,  
     [19](#)  
 PMIX\_EVENT\_RETURN\_OBJECT, [154](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_SILENT\_TERMINATION,  
     [154](#)  
     Defintion, [60](#)  
 PMIX\_EVENT\_TERMINATE\_JOB, [154](#)  
     Defintion, [61](#)  
 PMIX\_EVENT\_TERMINATE\_NODE, [154](#)  
     Defintion, [61](#)  
 PMIX\_EVENT\_TERMINATE\_PROC, [154](#)  
     Defintion, [61](#)  
 PMIX\_EVENT\_TERMINATE\_SESSION,  
     [154](#)  
     Defintion, [61](#)  
 PMIX\_EVENT\_WANT\_TERMINATION  
     Defintion, [61](#)  
 pmix\_evhdlr\_reg\_cbfunc\_t, [72](#), [153](#)  
     Defintion, [72](#)  
 PMIX\_EXISTS, [17](#)  
 PMIX\_EXTERNAL\_ERR\_BASE, [19](#)  
 PMIx\_Fence, [3](#), [7](#), [8](#), [10](#), [93](#), [103](#), [105](#), [129](#),  
     [132](#), [179](#), [188](#), [190](#)  
     Defintion, [102](#)  
 PMIx\_Fence\_nb, [8](#), [70](#), [105](#), [188](#), [190](#)  
     Defintion, [103](#)  
 PMIx\_Finalize, [8](#), [18](#), [24](#), [58](#), [85](#), [86](#), [126](#),  
     [185](#), [186](#)  
     Defintion, [86](#)  
 PMIX\_FLOAT, [50](#)  
 PMIX\_FWD\_STDERR, [121](#), [125](#), [200](#), [213](#)  
     Defintion, [62](#)  
 PMIX\_FWD\_STDIN, [121](#), [125](#), [200](#), [213](#)  
     Defintion, [62](#)  
 PMIX\_FWD\_STDOUT, [121](#), [125](#), [200](#), [213](#)  
     Defintion, [62](#)  
 PMIX\_GDS\_ACTION\_COMPLETE, [19](#)  
 PMIX\_GDS\_MODULE, [85](#), [89](#), [93](#)  
     Defintion, [54](#)  
 PMIx\_generate\_ppn, [8](#)  
     Defintion, [170](#)  
 PMIx\_generate\_regex, [8](#)  
     Defintion, [169](#)  
 PMIx\_Get, [3](#), [8](#), [28](#), [51](#), [58](#), [85](#), [97–100](#)  
     Defintion, [96](#)  
 PMIx\_Get\_nb, [8](#), [71](#)  
     Defintion, [98](#)  
 PMIx\_Get\_version, [9](#), [13](#)  
     Defintion, [83](#)  
 PMIX\_GLOBAL, [27](#)  
 PMIX\_GLOBAL\_RANK, [174](#)  
     Defintion, [55](#)  
 PMIX\_GRPID, [106](#), [108](#), [110](#), [112](#), [113](#), [115](#),  
     [137](#), [140](#), [142](#), [145](#), [148](#), [192–198](#),  
     [205](#), [210](#), [212](#), [214](#), [216](#), [219](#), [221](#)  
     Defintion, [52](#)  
 PMIx\_Heartbeat, [9](#)  
     Defintion, [146](#)  
 PMIX\_HOST, [120](#), [124](#), [199](#)  
     Defintion, [61](#)  
 PMIX\_HOSTFILE, [120](#), [124](#), [199](#)  
     Defintion, [61](#)  
 PMIX\_HOSTNAME, [174](#)  
     Defintion, [55](#)  
 PMIX\_HWLOC\_SHMEM\_ADDR  
     Defintion, [57](#)  
 PMIX\_HWLOC\_SHMEM\_FILE  
     Defintion, [57](#)

PMIX\_HWLOC\_SHMEM\_SIZE  
     Defintion, [57](#)  
 PMIX\_HWLOC\_XML\_V1, [174](#)  
     Defintion, [57](#)  
 PMIX\_HWLOC\_XML\_V2, [174](#)  
     Defintion, [57](#)  
 PMIX\_IMMEDIATE, [97](#), [99](#)  
     Defintion, [58](#)  
 PMIX\_INDEX\_ARGV, [121](#), [125](#), [200](#)  
     Defintion, [62](#)  
 PMIX\_INFO, [50](#)  
 PMIX\_INFO\_ARRAY, [50](#)  
 pmix\_info\_array, [31](#), [32](#)  
     Defintion, [31](#)  
 pmix\_info\_cbfunc\_t, [68](#), [71](#), [136](#), [139](#), [142](#),  
     [144–146](#), [210](#), [216](#), [218](#), [220](#)  
     Defintion, [71](#), [136](#)  
 PMIX\_INFO\_CONSTRUCT  
     Defintion, [32](#)  
 PMIX\_INFO\_CREATE  
     Defintion, [33](#)  
 PMIX\_INFO\_DESTRUCT  
     Defintion, [32](#)  
 PMIX\_INFO\_DIRECTIVES, [50](#)  
 PMIx\_Info\_directives\_string, [9](#)  
     Defintion, [81](#)  
 pmix\_info\_directives\_t, [35](#), [81](#)  
     Defintion, [35](#)  
 PMIX\_INFO\_FREE  
     Defintion, [33](#)  
 PMIX\_INFO\_IS\_REQUIRED, [35](#)  
     Defintion, [36](#)  
 PMIX\_INFO\_LOAD  
     Defintion, [33](#)  
 PMIX\_INFO\_REQD, [35](#), [36](#)  
 PMIX\_INFO\_REQUIRED, [35](#)  
     Defintion, [36](#)  
 pmix\_info\_t, [3](#), [9](#), [10](#), [27](#), [31–36](#), [64–66](#),  
     [72–75](#), [84](#), [86](#), [87](#), [91](#), [93](#), [107](#),  
     [111](#), [138–140](#), [143](#), [144](#), [146](#), [148](#),  
     [157](#), [208](#), [212](#), [213](#), [215](#), [217](#), [218](#),  
     [220](#), [221](#)  
     Defintion, [31](#)  
 PMIX\_INFO\_TRUE  
     Defintion, [34](#)  
 PMIX\_INFO\_XFER  
     Defintion, [34](#)  
 PMIx\_Init, [9](#), [62](#), [64](#), [83](#), [85](#), [86](#), [121](#), [125](#),  
     [185](#), [200](#)  
     Defintion, [83](#)  
 PMIx\_init, [185](#)  
 PMIx\_Initialized, [8](#)  
     Defintion, [82](#)  
 PMIX\_INT, [49](#)  
 PMIX\_INT16, [49](#)  
 PMIX\_INT32, [49](#)  
 PMIX\_INT64, [49](#)  
 PMIX\_INT8, [49](#)  
 PMIX\_INTERNAL, [27](#)  
 PMIX\_JCTRL\_CHECKPOINT, [18](#)  
 PMIX\_JCTRL\_CHECKPOINT\_COMPLETE,  
     [18](#)  
 PMIX\_JCTRL\_PREEMPT\_ALERT, [18](#)  
 PMIX\_JOB\_CONTINUOUS, [121](#), [125](#), [201](#)  
     Defintion, [63](#)  
 PMIx\_Job\_control\_nb, [9](#), [134](#), [141](#), [175](#)  
     Defintion, [141](#)  
 PMIX\_JOB\_CTRL\_CANCEL, [143](#), [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT, [143](#),  
     [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT,  
     [143](#), [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD,  
     [143](#), [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL,  
     [143](#), [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT,  
     [143](#), [220](#)  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_ID, [142](#), [219](#)  
     Defintion, [66](#)

PMIX\_JOB\_CTRL\_KILL, 142, 219  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_PAUSE, 142, 219  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_PREEMPTIBLE, 143,  
     220  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_PROVISION, 143, 220  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_PROVISION\_IMAGE,  
     143, 220  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_RESTART, 143, 220  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_RESUME, 142, 219  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_SIGNAL, 143, 219  
     Defintion, [66](#)  
 PMIX\_JOB\_CTRL\_TERMINATE, 143, 219  
     Defintion, [66](#)  
 PMIX\_JOB\_NUM\_APPS, 174  
     Defintion, [56](#)  
 PMIX\_JOB\_RECOVERABLE, 121, 125,  
     201  
     Defintion, [63](#)  
 PMIX\_JOB\_SIZE, 98, 100, 172  
     Defintion, [56](#)  
 PMIX\_JOB\_TERM\_STATUS  
     Defintion, [58](#)  
 PMIX\_JOBID, 172  
     Defintion, [55](#)  
 pmix\_key\_t, 19, 20, 95, 97  
     Defintion, [19](#)  
 PMIX\_KVAL, 50  
 PMIX\_LOCAL, 26  
 PMIX\_LOCAL\_CPUSSETS, 172  
     Defintion, [56](#)  
 PMIX\_LOCAL\_PEERS, 172  
     Defintion, [55](#)  
 PMIX\_LOCAL\_PROCS, 174  
     Defintion, [56](#)  
 PMIX\_LOCAL\_RANK, 173  
     Defintion, [55](#)  
 PMIX\_LOCAL\_SIZE, 172  
     Defintion, [56](#)  
 PMIX\_LOCAL\_TOPO  
     Defintion, [57](#)  
 PMIX\_LOCALITY  
     Defintion, [56](#)  
 PMIX\_LOCALITY\_STRING  
     Defintion, [57](#)  
 PMIX\_LOCALLDR, 174  
     Defintion, [55](#)  
 PMIX\_LOG\_EMAIL, 148, 215  
     Defintion, [64](#)  
 PMIX\_LOG\_EMAIL\_ADDR, 148, 215  
     Defintion, [64](#)  
 PMIX\_LOG\_EMAIL\_MSG, 148, 215  
     Defintion, [64](#)  
 PMIX\_LOG\_EMAIL\_SUBJECT, 148, 215  
     Defintion, [64](#)  
 PMIX\_LOG\_MSG, 148, 215  
     Defintion, [64](#)  
 PMIx\_Log\_nb, 9, 64, 149  
     Defintion, [147](#)  
 PMIX\_LOG\_STDERR, 148, 214  
     Defintion, [64](#)  
 PMIX\_LOG\_STDOUT, 148, 214  
     Defintion, [64](#)  
 PMIX\_LOG\_SYSLOG, 148, 214  
     Defintion, [64](#)  
 PMIx\_Lookup, 8, 37, 105, 111, 113  
     Defintion, [109](#)  
 pmix\_lookup\_cbfunc\_t, 70, 194  
     Defintion, [70](#)  
 PMIx\_Lookup\_nb, 70, 71  
     Defintion, [111](#)  
 PMIX\_MAP\_BLOB  
     Defintion, [59](#)  
 PMIX\_MAPBY, 120, 124, 174, 200  
     Defintion, [62](#)  
 PMIX\_MAPPER, 120, 124, 199  
     Defintion, [61](#)  
 PMIX\_MAX\_KEYLEN, 15, 20  
 PMIX\_MAX\_NSLEN, 15, 20  
 PMIX\_MAX\_PROCS, 172

Defintion, [56](#)  
 PMIX\_MAX\_RESTARTS, [122](#), [126](#), [201](#)  
     Defintion, [63](#)  
 PMIX\_MERGE\_STDERR\_STDOUT, [121](#),  
     [125](#), [200](#)  
     Defintion, [62](#)  
 PMIX\_MODEL\_DECLARED, [19](#)  
 PMIX\_MODEL\_LIBRARY\_NAME  
     Defintion, [53](#)  
 PMIX\_MODEL\_LIBRARY\_VERSION  
     Defintion, [53](#)  
 PMIX\_MODEX, [50](#)  
 pmix\_modex\_cbfunc\_t, [68](#), [188](#), [190](#)  
     Defintion, [68](#)  
 PMIX\_MODEX\_CONSTRUCT  
     Defintion, [44](#)  
 PMIX\_MODEX\_CREATE  
     Defintion, [44](#)  
 pmix\_modex\_data\_t, [43](#)  
     Defintion, [43](#)  
 PMIX\_MODEX\_DESTRUCT  
     Defintion, [44](#)  
 PMIX\_MODEX\_FREE  
     Defintion, [45](#)  
 pmix\_modex\_t, [44](#), [45](#)  
 PMIX\_MONITOR\_APP\_CONTROL, [145](#),  
     [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_CANCEL, [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE, [145](#), [146](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE\_ACCESS, [145](#),  
     [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE\_ALERT, [18](#)  
 PMIX\_MONITOR\_FILE\_CHECK\_TIME,  
     [146](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE\_DROPS, [146](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE\_MODIFY, [146](#),  
     [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_FILE\_SIZE, [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_HEARTBEAT, [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_HEARTBEAT\_ALERT,  
     [18](#)  
 PMIX\_MONITOR\_HEARTBEAT\_DROPS,  
     [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_HEARTBEAT\_TIME,  
     [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_MONITOR\_ID, [145](#), [222](#)  
     Defintion, [67](#)  
 PMIX\_NET\_TOPO  
     Defintion, [57](#)  
 PMIX\_NO\_OVERSUBSCRIBE, [121](#), [125](#),  
     [200](#)  
     Defintion, [63](#)  
 PMIX\_NO\_PROCS\_ON\_HEAD, [121](#), [125](#),  
     [200](#)  
     Defintion, [62](#)  
 PMIX\_NODE\_LIST, [172](#)  
     Defintion, [55](#), [57](#)  
 PMIX\_NODE\_MAP, [172](#)  
     Defintion, [58](#)  
 PMIX\_NODE\_RANK, [173](#)  
     Defintion, [55](#)  
 PMIX\_NODE\_SIZE, [174](#)  
     Defintion, [56](#)  
 PMIX\_NODEID, [173](#)  
     Defintion, [55](#)  
 PMIX\_NON\_PMI, [120](#), [124](#), [200](#)  
     Defintion, [62](#)  
 pmix\_notification\_fn\_t, [74](#), [153](#)  
     Defintion, [74](#)  
 PMIX\_NOTIFY\_ALLOC\_COMPLETE, [18](#)  
 PMIX\_NOTIFY\_COMPLETION  
     Defintion, [58](#)  
 PMIx\_Notify\_error, [9](#)  
 PMIx\_Notify\_event, [9](#)  
     Defintion, [156](#)

PMIX\_NPROC\_OFFSET, 173  
     Defintion, 55  
 PMIX\_NSDIR, 54  
     Defintion, 54  
 PMIX\_NSPACE, 172  
     Defintion, 55  
 pmix\_nspace\_t, 20, 23, 69  
     Defintion, 20  
 PMIX\_NUM\_NODES, 98, 100, 172  
     Defintion, 56  
 pmix\_op\_cbfunc\_t, 70, 73, 77, 108, 115,  
     129, 133, 147, 156, 157, 171, 176,  
     177, 182, 185–187, 192, 196, 202,  
     203, 205, 207, 208, 214, 219, 221  
     Defintion, 70  
 PMIX\_OPTIONAL, 97, 99  
     Defintion, 58  
 PMIX\_OUTPUT\_TO\_FILE, 121, 125, 200  
     Defintion, 62  
 PMIX\_PARENT\_ID, 119, 123, 199  
     Defintion, 56  
 PMIX\_PDATA, 50  
 PMIX\_PDATA\_CONSTRUCT  
     Defintion, 37  
 PMIX\_PDATA\_CREATE  
     Defintion, 38  
 PMIX\_PDATA\_DESTRUCT  
     Defintion, 37  
 PMIX\_PDATA\_FREE  
     Defintion, 38  
 PMIX\_PDATA\_LOAD  
     Defintion, 38  
 pmix\_pdata\_t, 37–39, 70, 71, 111  
     Defintion, 37  
 PMIX\_PDATA\_XFER  
     Defintion, 39  
 PMIX\_PERSIST, 50  
 PMIX\_PERSIST\_APP, 27  
 PMIX\_PERSIST\_FIRST\_READ, 27  
 PMIX\_PERSIST\_INDEF, 27  
 PMIX\_PERSIST\_PROC, 27  
 PMIX\_PERSIST\_SESSION, 27  
 PMIX\_PERSISTENCE, 106, 108, 192  
     Defintion, 58  
 PMIx\_Persistence\_string, 9  
     Defintion, 80  
 pmix\_persistence\_t, 27, 50, 80  
     Defintion, 27  
 PMIX\_PERSONALITY, 120, 124, 199  
     Defintion, 61  
 PMIX\_PID, 49  
 PMIX\_POINTER, 50  
 PMIX\_PPR, 120, 124, 199  
     Defintion, 61  
 PMIX\_PREFIX, 120, 124, 199  
     Defintion, 61  
 PMIX\_PRELOAD\_BIN, 120, 124, 199  
     Defintion, 62  
 PMIX\_PRELOAD\_FILES, 120, 124, 199  
     Defintion, 62  
 PMIX\_PROC, 50  
 PMIX\_PROC\_BLOB  
     Defintion, 59  
 PMIX\_PROC\_CONSTRUCT, 22  
     Defintion, 22, 45, 48  
 PMIX\_PROC\_CREATE  
     Defintion, 22  
 PMIX\_PROC\_DATA  
     Defintion, 58  
 PMIX\_PROC\_DESTRUCT  
     Defintion, 22  
 PMIX\_PROC\_FREE, 135  
     Defintion, 23  
 PMIX\_PROC\_INFO, 50  
 PMIX\_PROC\_INFO\_CONSTRUCT  
     Defintion, 25  
 PMIX\_PROC\_INFO\_CREATE  
     Defintion, 26  
 PMIX\_PROC\_INFO\_DESTRUCT  
     Defintion, 25  
 PMIX\_PROC\_INFO\_FREE  
     Defintion, 26  
 pmix\_proc\_info\_t, 24–26, 50, 63, 137, 211  
     Defintion, 24  
 PMIX\_PROC\_LOAD  
     Defintion, 23

PMIX\_PROC\_MAP, 172  
     Defintion, **59**  
 PMIX\_PROC\_PID  
     Defintion, **55**  
 PMIX\_PROC\_RANK, 50  
 PMIX\_PROC\_STATE, 50  
 PMIX\_PROC\_STATE\_ABORTED, 24  
 PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG,  
     24  
 PMIX\_PROC\_STATE\_CALLED\_ABORT,  
     24  
 PMIX\_PROC\_STATE\_CANNOT\_RESTART,  
     24  
 PMIX\_PROC\_STATE\_COMM\_FAILED,  
     24  
 PMIX\_PROC\_STATE\_CONNECTED, 24  
 PMIX\_PROC\_STATE\_ERROR, 24  
 PMIX\_PROC\_STATE\_FAILED\_TO\_LAUNCH,  
     24  
 PMIX\_PROC\_STATE\_FAILED\_TO\_START,  
     24  
 PMIX\_PROC\_STATE\_KILLED\_BY\_CMD,  
     24  
 PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY,  
     24  
 PMIX\_PROC\_STATE\_MIGRATING, 24  
 PMIX\_PROC\_STATE\_PREPPED, 24  
 PMIX\_PROC\_STATE\_RESTART, 24  
 PMIX\_PROC\_STATE\_RUNNING, 24  
 PMIX\_PROC\_STATE\_STATUS  
     Defintion, **58**  
 PMIx\_Proc\_state\_string, 9  
     Defintion, **80**  
 pmix\_proc\_state\_t, 23, 50, 80  
     Defintion, **23**  
 PMIX\_PROC\_STATE\_TERM\_NON\_ZERO,  
     24  
 PMIX\_PROC\_STATE\_TERM\_WO\_SYNC,  
     24  
 PMIX\_PROC\_STATE\_TERMINATE, 24  
 PMIX\_PROC\_STATE\_TERMINATED, 24  
 PMIX\_PROC\_STATE\_UNDEF, 24  
 PMIX\_PROC\_STATE\_UNTERMINATED,  
     24  
 pmix\_proc\_t, 21–23, 38, 39, 50, 56, 60, 75,  
     79, 85, 87, 89, 98, 102–104, 117,  
     154, 157, 163, 164, 174, 176–179,  
     185–188, 190, 192, 194, 196, 198,  
     202, 203, 208, 210, 214, 216, 218,  
     221  
     Defintion, **21**  
 PMIX\_PROC\_TERMINATED, 18  
 PMIX\_PROC\_URI  
     Defintion, **56**  
 PMIX\_PROCDIR  
     Defintion, **54**  
 PMIx\_Process\_monitor\_nb, 9, 134, 147  
     Defintion, **144**  
 PMIX\_PROCID, 174  
     Defintion, **55**  
 PMIX\_PROGRAMMING\_MODEL  
     Defintion, **53**  
 PMIx\_Publish, 8, 27, 58, 106–109, 192, 193  
     Defintion, **105**  
 PMIx\_Publish\_nb, 8, 109  
     Defintion, **107**  
 PMIx\_Put, 8, 26–28, 78, 96, 98, 100–103,  
     126, 179, 191  
     Defintion, **95**  
 PMIX\_QUERY, 50  
 PMIX\_QUERY\_ALLOC\_STATUS, 138,  
     211  
     Defintion, **64**  
 PMIX\_QUERY\_AUTHORIZATIONS  
     Defintion, **63**  
 PMIX\_QUERY\_CONSTRUCT  
     Defintion, **42**  
 PMIX\_QUERY\_CREATE  
     Defintion, **43**  
 PMIX\_QUERY\_DEBUG\_SUPPORT, 137,  
     211  
     Defintion, **63**  
 PMIX\_QUERY\_DESTRUCT  
     Defintion, **42**  
 PMIX\_QUERY\_FREE  
     Defintion, **43**

PMIx\_Query\_info\_nb, 9, 42, 63, 134  
     Defintion, 136  
 PMIX\_QUERY\_JOB\_STATUS, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_LOCAL\_ONLY, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_LOCAL\_PROC\_TABLE,  
     137, 211  
     Defintion, 63  
 PMIX\_QUERY\_MEMORY\_USAGE, 137,  
     211  
     Defintion, 63  
 PMIX\_QUERY\_NAMESPACES, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_PARTIAL\_SUCCESS, 18  
 PMIX\_QUERY\_PROC\_TABLE, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_QUEUE\_LIST, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_QUEUE\_STATUS, 137,  
     211  
     Defintion, 63  
 PMIX\_QUERY\_REPORT\_AVG, 137, 211  
     Defintion, 63  
 PMIX\_QUERY\_REPORT\_MINMAX, 138,  
     211  
     Defintion, 64  
 PMIX\_QUERY\_SPAWN\_SUPPORT, 137,  
     211  
     Defintion, 63  
 pmix\_query\_t, 42, 43, 210, 212  
     Defintion, 42  
 PMIX\_RANGE, 106, 108, 110, 112, 114,  
     115, 154, 192, 194, 196, 208  
     Defintion, 58  
 PMIX\_RANGE\_CUSTOM, 27  
 PMIX\_RANGE\_GLOBAL, 27  
 PMIX\_RANGE\_LOCAL, 27  
 PMIX\_RANGE\_NAMESPACE, 27  
 PMIX\_RANGE\_PROC\_LOCAL, 27  
 PMIX\_RANGE\_RM, 27  
 PMIX\_RANGE\_SESSION, 27  
 PMIX\_RANGE\_UNDEF, 27  
 PMIX\_RANK, 172  
     Defintion, 55  
 PMIX\_RANK\_LOCAL\_NODE, 21  
 pmix\_rank\_t, 21, 23, 50  
     Defintion, 21  
 PMIX\_RANK\_UNDEF, 21  
 PMIX\_RANK\_WILDCARD, 21  
 PMIX\_RANKBY, 120, 124, 174, 200  
     Defintion, 62  
 PMIx\_Register\_errhandler, 9  
 PMIx\_Register\_event\_handler, 9, 73, 134  
     Defintion, 152  
 PMIX\_REGISTER\_NODATA, 172  
     Defintion, 51, 58  
 pmix\_release\_cbfunc\_t, 68  
     Defintion, 68  
 PMIX\_REMOTE, 26  
 PMIX\_REPORT\_BINDINGS, 121, 125, 200  
     Defintion, 63  
 PMIX\_REQUESTOR\_IS\_CLIENT, 119,  
     123  
     Defintion, 53  
 PMIX\_REQUESTOR\_IS\_TOOL, 119, 123  
     Defintion, 53  
 PMIx\_Resolve\_nodes, 8  
     Defintion, 135  
 PMIx\_Resolve\_peers, 8  
     Defintion, 135  
 PMIX\_RM\_NAME  
     Defintion, 65  
 PMIX\_RM\_VERSION  
     Defintion, 65  
 PMIX\_SCOPE, 50  
 PMIx\_Scope\_string, 9  
     Defintion, 80  
 pmix\_scope\_t, 26, 50, 80, 96  
     Defintion, 26  
 PMIX\_SCOPE\_UNDEF, 26  
 PMIX\_SEND\_HEARTBEAT  
     Defintion, 67  
 pmix\_server\_abort\_fn\_t  
     Defintion, 187  
 pmix\_server\_alloc\_fn\_t

Defintion, [215](#)  
 pmix\_server\_client\_connected\_fn\_t, [70](#), [177](#),  
     [185](#)  
     Defintion, [184](#)  
 PMIx\_server\_client\_finalized\_fn\_t, [186](#)  
 pmix\_server\_client\_finalized\_fn\_t, [186](#)  
     Defintion, [185](#)  
 pmix\_server\_connect\_fn\_t, [203](#), [204](#)  
     Defintion, [201](#)  
 PMIx\_server\_deregister\_client, [8](#)  
     Defintion, [177](#)  
 pmix\_server\_deregister\_events\_fn\_t  
     Defintion, [206](#)  
 PMIx\_server\_deregister\_nspace, [8](#), [178](#)  
     Defintion, [175](#)  
 pmix\_server\_disconnect\_fn\_t, [204](#)  
     Defintion, [203](#)  
 pmix\_server\_dmodex\_req\_fn\_t, [68](#)  
     Defintion, [190](#)  
 PMIx\_server\_dmodex\_request, [9](#), [77](#), [78](#),  
     [179](#)  
     Defintion, [179](#)  
 PMIX\_SERVER\_ENABLE\_MONITORING  
     Defintion, [52](#)  
 pmix\_server\_fencenb\_fn\_t, [68](#), [190](#)  
     Defintion, [188](#)  
 PMIx\_server\_finalize, [8](#)  
     Defintion, [93](#)  
 PMIX\_SERVER\_HOSTNAME  
     Defintion, [52](#)  
 PMIx\_server\_init, [8](#), [83](#), [183](#)  
     Defintion, [91](#)  
 pmix\_server\_job\_control\_fn\_t  
     Defintion, [218](#)  
 pmix\_server\_listener\_fn\_t  
     Defintion, [209](#)  
 pmix\_server\_log\_fn\_t  
     Defintion, [213](#)  
 pmix\_server\_lookup\_fn\_t  
     Defintion, [193](#)  
 pmix\_server\_module\_t, [91](#), [93](#), [183](#)  
     Defintion, [183](#)  
 pmix\_server\_monitor\_fn\_t  
     Defintion, [221](#)  
 pmix\_server\_notify\_event\_fn\_t, [76](#)  
     Defintion, [207](#)  
 PMIX\_SERVER\_NAMESPACE, [91](#), [173](#)  
     Defintion, [52](#)  
 PMIX\_SERVER\_PIDINFO, [88](#), [89](#)  
     Defintion, [52](#)  
 pmix\_server\_publish\_fn\_t  
     Defintion, [191](#)  
 pmix\_server\_query\_fn\_t  
     Defintion, [210](#)  
 PMIX\_SERVER\_RANK, [91](#), [173](#)  
     Defintion, [52](#)  
 PMIx\_server\_register\_client, [8](#), [185](#), [186](#)  
     Defintion, [176](#)  
 pmix\_server\_register\_events\_fn\_t  
     Defintion, [204](#)  
 PMIx\_server\_register\_nspace, [8](#), [13](#), [70](#)  
     Defintion, [171](#)  
 PMIX\_SERVER\_REMOTE\_CONNECTIONS,  
     [93](#)  
     Defintion, [51](#)  
 PMIx\_server\_setup\_application, [9](#), [76](#), [77](#),  
     [182](#)  
     Defintion, [180](#)  
 PMIx\_server\_setup\_fork, [9](#)  
     Defintion, [178](#)  
 PMIx\_server\_setup\_local\_support, [9](#)  
     Defintion, [181](#)  
 pmix\_server\_spawn\_fn\_t, [69](#)  
     Defintion, [197](#)  
 PMIX\_SERVER\_SYSTEM\_SUPPORT, [92](#)  
     Defintion, [51](#)  
 PMIX\_SERVER\_TMPDIR, [91](#)  
     Defintion, [51](#)  
 pmix\_server\_tool\_connection\_fn\_t  
     Defintion, [212](#)  
 PMIX\_SERVER\_TOOL\_SUPPORT, [91](#)  
     Defintion, [51](#)  
 pmix\_server\_unpublish\_fn\_t  
     Defintion, [195](#)  
 PMIX\_SERVER\_URI, [87](#), [89](#)  
     Defintion, [52](#)



PMIX\_SESSION\_ID, 173  
     Defintion, [55](#)  
 PMIX\_SET\_ENVAR  
     Defintion, [65](#)  
 PMIX\_SET\_SESSION\_CWD, 120, 124, 199  
     Defintion, [62](#)  
 pmix\_setup\_application\_cbfunc\_t, 180  
     Defintion, [76](#)  
 PMIX\_SINGLE\_LISTENER, 84  
     Defintion, [53](#)  
 PMIX\_SIZE, 49  
 PMIX\_SOCKET\_MODE, 84, 88, 92  
     Defintion, [53](#)  
 PMIx\_Spawn, 8, 40, 54, 61, 118, 119, 122,  
     123, 126, 175, 178, 197, 201  
     Defintion, [118](#)  
 pmix\_spawn\_cbfunc\_t, 69, 123, 198  
     Defintion, [69](#)  
 PMIx\_Spawn\_nb, 8, 40, 69  
     Defintion, [122](#)  
 PMIX\_SPAWNED, 119, 123, 199  
     Defintion, [54](#)  
 PMIX\_STATUS, 50  
 pmix\_status\_t, 16, 31, 50, 72, 73, 75, 77–80,  
     153, 157, 205, 207, 208  
     Defintion, [16](#)  
 PMIX\_STDIN\_TGT, 121, 125, 200  
     Defintion, [62](#)  
 PMIx\_Store\_internal, 9  
     Defintion, [100](#)  
 PMIX\_STRING, 49  
 PMIX\_SUCCESS, 17  
 PMIX\_SYSTEM\_TMPDIR, 91  
     Defintion, [51](#)  
 PMIX\_TAG\_OUTPUT, 121, 125, 200  
     Defintion, [62](#)  
 PMIX\_TCP\_DISABLE\_IPV4, 85, 89, 92  
     Defintion, [54](#)  
 PMIX\_TCP\_DISABLE\_IPV6, 85, 89, 92  
     Defintion, [54](#)  
 PMIX\_TCP\_IF\_EXCLUDE, 84, 88, 92  
     Defintion, [53](#)  
 PMIX\_TCP\_IF\_INCLUDE, 84, 88, 92  
     Defintion, [53](#)  
 PMIX\_TCP\_IPV4\_PORT, 85, 88, 92  
     Defintion, [54](#)  
 PMIX\_TCP\_IPV6\_PORT, 85, 88, 92  
     Defintion, [54](#)  
 PMIX\_TCP\_REPORT\_URI, 84, 88, 92  
     Defintion, [53](#)  
 PMIX\_TCP\_URI, 88, 89  
     Defintion, [53](#)  
 PMIX\_TDIR\_RMCLEAN  
     Defintion, [54](#)  
 PMIX\_THREADING\_MODEL  
     Defintion, [53](#)  
 PMIX\_TIME, 50  
 PMIX\_TIME\_REMAINING, 134, 138, 211  
     Defintion, [64](#)  
 PMIX\_TIMEOUT, 3, 10, 97–100, 103–106,  
     108–110, 112–116, 128, 130, 131,  
     133, 189, 191, 193, 195, 197, 201,  
     202, 204  
     Defintion, [57](#)  
 PMIX\_TIMESTAMP\_OUTPUT, 121, 125,  
     200  
     Defintion, [62](#)  
 PMIX\_TIMEVAL, 50  
 PMIX\_TMPDIR, 54  
     Defintion, [54](#)  
 pmix\_tool\_connection\_cbfunc\_t, 212  
     Defintion, [79](#)  
 PMIX\_TOOL\_DO\_NOT\_CONNECT, 87,  
     89  
     Defintion, [52](#)  
 PMIx\_tool\_finalize, 9  
     Defintion, [90](#)  
 PMIx\_tool\_init, 9, 83, 90  
     Defintion, [87](#)  
 PMIX\_TOOL\_NAMESPACE, 87  
     Defintion, [52](#)  
 PMIX\_TOOL\_RANK, 87  
     Defintion, [52](#)  
 PMIX\_TOPOLOGY  
     Defintion, [57](#)  
 PMIX\_TOPOLOGY\_SIGNATURE

Defintion, [57](#)  
 PMIX\_UINT, [49](#)  
 PMIX\_UINT16, [50](#)  
 PMIX\_UINT32, [50](#)  
 PMIX\_UINT64, [50](#)  
 PMIX\_UINT8, [49](#)  
 PMIX\_UNDEF, [49](#)  
 PMIX\_UNIV\_SIZE, [98](#), [100](#), [172](#)  
     Defintion, [56](#)  
 PMIx\_Unpublish, [8](#), [114](#), [116](#)  
     Defintion, [113](#)  
 PMIx\_Unpublish\_nb, [8](#)  
     Defintion, [114](#)  
 PMIX\_UNSET\_ENVAR  
     Defintion, [65](#)  
 PMIX\_USERID, [106](#), [108](#), [110](#), [112](#), [113](#),  
     [115](#), [137](#), [140](#), [142](#), [145](#), [148](#),  
     [192–198](#), [205](#), [210](#), [212](#), [214](#), [216](#),  
     [219](#), [221](#)  
     Defintion, [52](#)  
 PMIX\_USOCK\_DISABLE, [84](#), [92](#)  
     Defintion, [53](#)  
 PMIX\_VALUE, [50](#)  
     pmix\_value\_cbfunc\_t, [71](#)  
         Defintion, [71](#)  
 PMIX\_VALUE\_CONSTRUCT  
     Defintion, [29](#)  
 PMIX\_VALUE\_CREATE  
     Defintion, [29](#)  
 PMIX\_VALUE\_DESTRUCT  
     Defintion, [29](#)  
 PMIX\_VALUE\_FREE  
     Defintion, [30](#)  
 PMIX\_VALUE\_LOAD  
     Defintion, [30](#)  
 pmix\_value\_t, [28–31](#), [50](#), [71](#), [95](#), [96](#)  
     Defintion, [28](#)  
 PMIX\_VALUE\_XFER  
     Defintion, [31](#)  
 PMIX\_VERSION\_INFO  
     Defintion, [53](#)  
 PMIX\_WAIT, [110–112](#), [194](#)  
     Defintion, [58](#)  
 PMIX\_WDIR, [119](#), [123](#), [199](#)  
     Defintion, [61](#)