



Process Management Interface for Exascale (PMIx) Standard

Version 2.2

February 2019

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 2.2.

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

Copyright © 2018-2019 PMIx Standard Review Board.

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

This page intentionally left blank

Contents

1. Introduction	1
1.1. Charter	2
1.2. PMIx Standard Overview	2
1.2.1. Who should use the standard?	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
1.7. Version 2.1: Dec. 2018	10
1.8. Version 2.2: Jan 2019	10
2. PMIx Terms and Conventions	12
2.1. Notational Conventions	14
2.2. Semantics	15
2.3. Naming Conventions	15
2.4. Procedure Conventions	16
2.5. Standard vs Reference Implementation	16
3. Data Structures and Types	17
3.1. Constants	17
3.1.1. Error Constants	18
3.1.2. Macros for use with PMIx constants	21
3.2. Data Types	22
3.2.1. Key Structure	22

3.2.2.	Namespace Structure	23
3.2.3.	Rank Structure	23
3.2.4.	Process Structure	24
3.2.5.	Process structure support macros	24
3.2.6.	Process State Structure	26
3.2.7.	Process Information Structure	27
3.2.8.	Process Information Structure support macros	27
3.2.9.	Scope of Put Data	29
3.2.10.	Range of Published Data	29
3.2.11.	Data Persistence Structure	30
3.2.12.	Data Array Structure	30
3.2.13.	Data array structure support macros	30
3.2.14.	Value Structure	32
3.2.15.	Value structure support macros	33
3.2.16.	Info Structure	36
3.2.17.	Info structure support macros	36
3.2.18.	Info Type Directives	39
3.2.19.	Info Directive support macros	40
3.2.20.	Job Allocation Directives	42
3.2.21.	Lookup Returned Data Structure	42
3.2.22.	Lookup data structure support macros	43
3.2.23.	Application Structure	45
3.2.24.	App structure support macros	46
3.2.25.	Query Structure	47
3.2.26.	Query structure support macros	47
3.2.27.	Modex Structure	49
3.2.28.	Modex data structure support macros	49
3.3.	Data Packing/Unpacking Types and Structures	51
3.3.1.	Byte Object Type	51
3.3.2.	Byte object support macros	51
3.3.3.	Data Buffer Type	53
3.3.4.	Data buffer support macros	53
3.3.5.	Data Array Structure	55

3.3.6.	Data array support macros	55
3.3.7.	Generalized Data Types Used for Packing/Unpacking	57
3.4.	Reserved attributes	58
3.4.1.	Initialization attributes	59
3.4.2.	Tool-related attributes	60
3.4.3.	Identification attributes	60
3.4.4.	UNIX socket rendezvous socket attributes	61
3.4.5.	TCP connection attributes	61
3.4.6.	Global Data Storage (GDS) attributes	62
3.4.7.	General process-level attributes	62
3.4.8.	Scratch directory attributes	62
3.4.9.	Relative Rank Descriptive Attributes	63
3.4.10.	Information retrieval attributes	64
3.4.11.	Information storage attributes	65
3.4.12.	Size information attributes	66
3.4.13.	Memory information attributes	67
3.4.14.	Topology information attributes	67
3.4.15.	Request-related attributes	68
3.4.16.	Server-to-PMIx library attributes	69
3.4.17.	Server-to-Client attributes	69
3.4.18.	Event handler registration and notification attributes	70
3.4.19.	Fault tolerance attributes	71
3.4.20.	Spawn attributes	72
3.4.21.	Query attributes	74
3.4.22.	Log attributes	75
3.4.23.	Debugger attributes	76
3.4.24.	Resource manager attributes	76
3.4.25.	Environment variable attributes	76
3.4.26.	Job Allocation attributes	77
3.4.27.	Job control attributes	77
3.4.28.	Monitoring attributes	78
3.5.	Callback Functions	79
3.5.1.	Release Callback Function	79

3.5.2.	Modex Callback Function	80
3.5.3.	Spawn Callback Function	81
3.5.4.	Op Callback Function	81
3.5.5.	Lookup Callback Function	82
3.5.6.	Value Callback Function	83
3.5.7.	Info Callback Function	83
3.5.8.	Event Handler Registration Callback Function	84
3.5.9.	Notification Handler Completion Callback Function	85
3.5.10.	Notification Function	86
3.5.11.	Server Setup Application Callback Function	87
3.5.12.	Server Direct Modex Response Callback Function	88
3.5.13.	PMIx Client Connection Callback Function	89
3.5.14.	PMIx Tool Connection Callback Function	90
3.5.15.	Constant String Functions	90
4.	Initialization and Finalization	93
4.1.	Query	93
4.1.1.	PMIx_Initialized	93
4.1.2.	PMIx_Get_version	94
4.2.	Client Initialization and Finalization	94
4.2.1.	PMIx_Init	94
4.2.2.	PMIx_Finalize	97
4.3.	Tool Initialization and Finalization	98
4.3.1.	PMIx_tool_init	98
4.3.2.	PMIx_tool_finalize	101
4.4.	Server Initialization and Finalization	102
4.4.1.	PMIx_server_init	102
4.4.2.	PMIx_server_finalize	104
5.	Key/Value Management	106
5.1.	Setting and Accessing Key/Value Pairs	106
5.1.1.	PMIx_Put	106
5.1.2.	PMIx_Get	107
5.1.3.	PMIx_Get_nb	110

5.1.4.	PMIx_Store_internal	113
5.1.5.	Accessing information: examples	114
5.2.	Exchanging Key/Value Pairs	119
5.2.1.	PMIx_Commit	119
5.2.2.	PMIx_Fence	119
5.2.3.	PMIx_Fence_nb	121
5.3.	Publish and Lookup Data	124
5.3.1.	PMIx_Publish	124
5.3.2.	PMIx_Publish_nb	126
5.3.3.	PMIx_Lookup	128
5.3.4.	PMIx_Lookup_nb	130
5.3.5.	PMIx_Unpublish	132
5.3.6.	PMIx_Unpublish_nb	134
6.	Process Management	136
6.1.	Abort	136
6.1.1.	PMIx_Abort	136
6.2.	Process Creation	137
6.2.1.	PMIx_Spawn	137
6.2.2.	PMIx_Spawn_nb	142
6.3.	Connecting and Disconnecting Processes	146
6.3.1.	PMIx_Connect	147
6.3.2.	PMIx_Connect_nb	149
6.3.3.	PMIx_Disconnect	151
6.3.4.	PMIx_Disconnect_nb	153
7.	Job Allocation Management and Reporting	156
7.1.	Query	156
7.1.1.	PMIx_Resolve_peers	157
7.1.2.	PMIx_Resolve_nodes	157
7.1.3.	PMIx_Query_info_nb	158
7.2.	Allocation Requests	163
7.2.1.	PMIx_Allocation_request_nb	164
7.2.2.	PMIx_Job_control_nb	166

7.3.	Process and Job Monitoring	169
7.3.1.	PMIx_Process_monitor_nb	170
7.3.2.	PMIx_Heartbeat	172
7.4.	Logging	172
7.4.1.	PMIx_Log_nb	173
8.	Event Notification	176
8.1.	Notification and Management	176
8.1.1.	PMIx_Register_event_handler	178
8.1.2.	PMIx_Deregister_event_handler	181
8.1.3.	PMIx_Notify_event	182
9.	Data Packing and Unpacking	185
9.1.	Support Macros	185
9.1.1.	PMIX_DATA_BUFFER_CREATE	185
9.1.2.	PMIX_DATA_BUFFER_RELEASE	186
9.1.3.	PMIX_DATA_BUFFER_CONSTRUCT	186
9.1.4.	PMIX_DATA_BUFFER_DESTRUCT	186
9.1.5.	PMIX_DATA_BUFFER_LOAD	187
9.1.6.	PMIX_DATA_BUFFER_UNLOAD	187
9.2.	General Routines	188
9.2.1.	PMIx_Data_pack	188
9.2.2.	PMIx_Data_unpack	190
9.2.3.	PMIx_Data_copy	192
9.2.4.	PMIx_Data_print	192
9.2.5.	PMIx_Data_copy_payload	193
10.	Server-Specific Interfaces	195
10.1.	Server Support Functions	195
10.1.1.	PMIx_generate_regex	195
10.1.2.	PMIx_generate_ppn	196
10.1.3.	PMIx_server_register_nspace	197
10.1.4.	PMIx_server_deregister_nspace	210
10.1.5.	PMIx_server_register_client	211
10.1.6.	PMIx_server_deregister_client	212

10.1.7.	<code>PMIx_server_setup_fork</code>	213
10.1.8.	<code>PMIx_server_dmodex_request</code>	214
10.1.9.	<code>PMIx_server_setup_application</code>	215
10.1.10.	<code>PMIx_server_setup_local_support</code>	217
10.2.	Server Function Pointers	218
10.2.1.	<code>pmix_server_module_t</code> Module	218
10.2.2.	<code>pmix_server_client_connected_fn_t</code>	219
10.2.3.	<code>pmix_server_client_finalized_fn_t</code>	221
10.2.4.	<code>pmix_server_abort_fn_t</code>	222
10.2.5.	<code>pmix_server_fence_nb_fn_t</code>	223
10.2.6.	<code>pmix_server_dmodex_req_fn_t</code>	226
10.2.7.	<code>pmix_server_publish_fn_t</code>	228
10.2.8.	<code>pmix_server_lookup_fn_t</code>	230
10.2.9.	<code>pmix_server_unpublish_fn_t</code>	232
10.2.10.	<code>pmix_server_spawn_fn_t</code>	234
10.2.11.	<code>pmix_server_connect_fn_t</code>	239
10.2.12.	<code>pmix_server_disconnect_fn_t</code>	241
10.2.13.	<code>pmix_server_register_events_fn_t</code>	243
10.2.14.	<code>pmix_server_deregister_events_fn_t</code>	245
10.2.15.	<code>pmix_server_notify_event_fn_t</code>	247
10.2.16.	<code>pmix_server_listener_fn_t</code>	248
10.2.17.	<code>pmix_server_query_fn_t</code>	249
10.2.18.	<code>pmix_server_tool_connection_fn_t</code>	252
10.2.19.	<code>pmix_server_log_fn_t</code>	253
10.2.20.	<code>pmix_server_alloc_fn_t</code>	255
10.2.21.	<code>pmix_server_job_control_fn_t</code>	257
10.2.22.	<code>pmix_server_monitor_fn_t</code>	260
A.	Acknowledgements	263
A.1.	Version 2.0	263
A.2.	Version 1.0	264
	Bibliography	266

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.

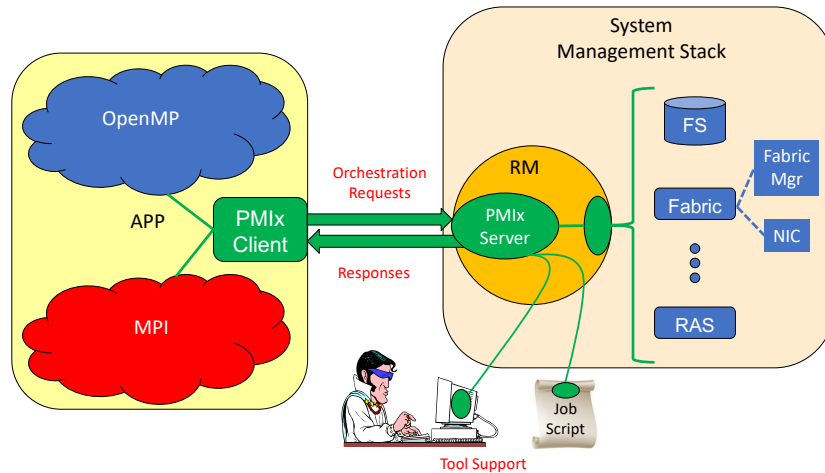


Figure 1.1.: PMIx-SMS Interactions

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 12
- Data Structures and Types in Chapter 3 on page 17

- 1 • PMIx Initialization and Finalization in Chapter 4 on page 93
- 2 • Key/Value Management in Chapter 5 on page 106
- 3 • Process Management in Chapter 6 on page 136
- 4 • Job Management in Chapter 7 on page 156
- 5 • Event Notification in Chapter 8 on page 176
- 6 • Data Packing and Unpacking in Chapter 9 on page 185
- 7 • PMIx Server Specific Interfaces in Chapter 10 on page 195

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.

1 1.7 Version 2.1: Dec. 2018

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

- 4 • Clarify description of `PMIx_Connect` and `PMIx_Disconnect` APIs.
- 5 • Explain that values for the `PMIX_COLLECTIVE_ALGO` are environment-dependent
- 6 • Identify the namespace/rank values required for retrieving attribute-associated information using
7 the `PMIx_Get` API
- 8 • Provide definitions for `session`, `job`, `application`, and other terms used throughout the
9 document
- 10 • Clarify definitions of `PMIX_UNIV_SIZE` versus `PMIX_JOB_SIZE`
- 11 • Clarify server module function return values
- 12 • Provide examples of the use of `PMIx_Get` for retrieval of information
- 13 • Clarify the use of `PMIx_Get` versus `PMIx_Query_info_nb`
- 14 • Clarify return values for non-blocking APIs and emphasize that callback functions must not be
15 invoked prior to return from the API
- 16 • Provide detailed example for construction of the `PMIx_server_register_namespace` input
17 information array
- 18 • Define information levels (e.g., `session` vs `job`) and associated attributes for both storing
19 and retrieving values
- 20 • Clarify roles of PMIx server library and host environment for collective operations
- 21 • Clarify definition of `PMIX_UNIV_SIZE`

22 1.8 Version 2.2: Jan 2019

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

- 24 • Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete
25 atomically as the API cannot return the requested information except via the provided callback
26 function
- 27 • Add missing `pmix_data_array_t` definition and support macros
- 28 • Add a rule divider between implementer and host environment required attributes for clarity
- 29 • Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t`
30 qualifiers

- 1 • Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- 2 • Add missing `PMIX_INFO_IS_OPTIONAL` macro
- 3 • Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a
- 4 `pmix_info_t` array
- 5 • Clarify the allowed hierarchical nesting of the `PMIX_SESSION_INFO_ARRAY` ,
- 6 `PMIX_JOB_INFO_ARRAY` , and associated attributes
- 7 • Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the
- 8 `session` level and (b) `PMIX_MAX_PROCS` when used at the `job` and `application`
- 9 levels, but leave it in for backward compatibility.

CHAPTER 2

PMIx Terms and Conventions

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

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

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

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

Advice to users

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

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

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

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

- *resource manager* is used in a generic sense to represent the system that will host the PMIx server library. This could be a vendor’s RM, a programming library’s RunTime Environment (RTE), or some other agent.
- *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.

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

2.1 Notational Conventions

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

▼ C ▼

C specific text...

```
int foo = 42;
```

▲ C ▲

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

▼

Note: General text...

▲

▼ Rationale ▼

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

▲

▼ Advice to users ▼

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

▲

Advice to PMIx library implementers

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 RTEs) providing support for the PMIx server library is set off in this section. Some
6 readers may wish to skip these sections, while readers interested in integrating PMIx servers into
7 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

2.3 Naming Conventions

14 The PMIx standard has adopted the following conventions:

- 15 • PMIx constants and attributes are prefixed with **PMIX**.
- 16 • Structures and type definitions are prefixed with **pmix**.
- 17 • Underscores are used to separate words in a function or variable name.
- 18 • Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the
19 first letter of the word following it. For example, **PMIx_Get_version** .
- 20 • PMIx server and tool APIs are all lower case letters following the prefix - e.g.,
21 **PMIx_server_register_namespace** .
- 22 • The **PMIX_** prefix is used to denote functions.
- 23 • The **pmix_** prefix is used to denote function pointer and type definitions.

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

1 2.4 Procedure Conventions

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

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

11 2.5 Standard vs Reference Implementation

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

18 Similarly, the *PMIx Reference RunTime Environment* (PRRTE) is provided by the community to
19 enable users operating in non-PMIx environments to develop and execute PMIx-enabled
20 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** . Values added to the list in
8 this version of the standard are shown in **magenta**.

Advice to PMIx library implementers

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

Advice to users

11 Other than **PMIX_SUCCESS** (which is required to be zero), the actual value of any PMIx error
12 constant is left to the PMIx library implementer. Thus, users are advised to always refer to constant
13 by name, and not a specific implementation's value, for portability between implementations and
14 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	PMIX_SUCCESS	Success
6	PMIX_ERROR	General Error
7	PMIX_ERR_SILENT	Silent error
8	PMIX_ERR_DEBUGGER_RELEASE	Error in debugger release
9	PMIX_ERR_PROC_RESTART	Fault tolerance: Error in process restart
10	PMIX_ERR_PROC_CHECKPOINT	Fault tolerance: Error in process checkpoint
11	PMIX_ERR_PROC_MIGRATE	Fault tolerance: Error in process migration
12	PMIX_ERR_PROC_ABORTED	Process was aborted
13	PMIX_ERR_PROC_REQUESTED_ABORT	Process is already requested to abort
14	PMIX_ERR_PROC_ABORTING	Process is being aborted
15	PMIX_ERR_SERVER_FAILED_REQUEST	Failed to connect to the server
16	PMIX_EXISTS	Requested operation would overwrite an existing value
17	PMIX_ERR_INVALID_CRED	Invalid security credentials
18	PMIX_ERR_HANDSHAKE_FAILED	Connection handshake failed
19	PMIX_ERR_READY_FOR_HANDSHAKE	Ready for handshake
20	PMIX_ERR_WOULD_BLOCK	Operation would block
21	PMIX_ERR_UNKNOWN_DATA_TYPE	Unknown data type
22	PMIX_ERR_PROC_ENTRY_NOT_FOUND	Process not found
23	PMIX_ERR_TYPE_MISMATCH	Invalid type
24	PMIX_ERR_UNPACK_INADEQUATE_SPACE	Inadequate space to unpack data
25	PMIX_ERR_UNPACK_FAILURE	Unpack failed
26	PMIX_ERR_PACK_FAILURE	Pack failed
27	PMIX_ERR_PACK_MISMATCH	Pack mismatch
28	PMIX_ERR_NO_PERMISSIONS	No permissions
29	PMIX_ERR_TIMEOUT	Timeout expired
30	PMIX_ERR_UNREACH	Unreachable
31	PMIX_ERR_IN_ERRNO	Error defined in errno
32	PMIX_ERR_BAD_PARAM	Bad parameter
33	PMIX_ERR_RESOURCE_BUSY	Resource busy
34	PMIX_ERR_OUT_OF_RESOURCE	Resource exhausted
35	PMIX_ERR_DATA_VALUE_NOT_FOUND	Data value not found
36	PMIX_ERR_INIT	Error during initialization
37	PMIX_ERR_NOMEM	Out of memory
38	PMIX_ERR_INVALID_ARG	Invalid argument
39	PMIX_ERR_INVALID_KEY	Invalid key
40	PMIX_ERR_INVALID_KEY_LENGTH	Invalid key length
41	PMIX_ERR_INVALID_VAL	Invalid value

1	<code>PMIX_ERR_INVALID_VAL_LENGTH</code>	Invalid value length
2	<code>PMIX_ERR_INVALID_LENGTH</code>	Invalid argument length
3	<code>PMIX_ERR_INVALID_NUM_ARGS</code>	Invalid number of arguments
4	<code>PMIX_ERR_INVALID_ARGS</code>	Invalid arguments
5	<code>PMIX_ERR_INVALID_NUM_PARSED</code>	Invalid number parsed
6	<code>PMIX_ERR_INVALID_KEYVALP</code>	Invalid key/value pair
7	<code>PMIX_ERR_INVALID_SIZE</code>	Invalid size
8	<code>PMIX_ERR_INVALID_NAMESPACE</code>	Invalid namespace
9	<code>PMIX_ERR_SERVER_NOT_AVAIL</code>	Server is not available
10	<code>PMIX_ERR_NOT_FOUND</code>	Not found
11	<code>PMIX_ERR_NOT_SUPPORTED</code>	Not supported
12	<code>PMIX_ERR_NOT_IMPLEMENTED</code>	Not implemented
13	<code>PMIX_ERR_COMM_FAILURE</code>	Communication failure
14	<code>PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER</code>	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	<code>PMIX_ERR_LOST_CONNECTION_TO_SERVER</code>	Lost connection to server
19	<code>PMIX_ERR_LOST_PEER_CONNECTION</code>	Lost connection to peer
20	<code>PMIX_ERR_LOST_CONNECTION_TO_CLIENT</code>	Lost connection to client
21	<code>PMIX_QUERY_PARTIAL_SUCCESS</code>	Query partial success (used by query system)
22	<code>PMIX_NOTIFY_ALLOC_COMPLETE</code>	Notify that allocation is complete
23	<code>PMIX_JCTRL_CHECKPOINT</code>	Job control: Monitored by PMIx client to trigger checkpoint
24		operation
25	<code>PMIX_JCTRL_CHECKPOINT_COMPLETE</code>	Job control: Sent by PMIx client and monitored
26		by PMIx server to notify that requested checkpoint operation has completed.
27	<code>PMIX_JCTRL_PREEMPT_ALERT</code>	Job control: Monitored by PMIx client to detect an RM
28		intending to preempt the job.
29	<code>PMIX_MONITOR_HEARTBEAT_ALERT</code>	Job monitoring: Heartbeat alert
30	<code>PMIX_MONITOR_FILE_ALERT</code>	Job monitoring: File alert
31	<code>PMIX_PROC_TERMINATED</code>	Process terminated - can be either normal or abnormal
32		termination
33	<code>PMIX_ERR_INVALID_TERMINATION</code>	Process terminated without calling
34	<code>PMIx_Finalize</code>	, or was a member of an assemblage formed via <code>PMIx_Connect</code> and
35		terminated or called <code>PMIx_Finalize</code> without first calling <code>PMIx_Disconnect</code> (or its
36		non-blocking form) from that assemblage.

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

38	<code>PMIX_ERR_EVENT_REGISTRATION</code>	Error in event registration
39	<code>PMIX_ERR_JOB_TERMINATED</code>	Error job terminated
40	<code>PMIX_ERR_UPDATE_ENDPOINTS</code>	Error updating endpoints

1 **PMIX_MODEL_DECLARED** Model declared
2 **PMIX_GDS_ACTION_COMPLETE** The global data storage (GDS) action has completed
3 **PMIX_OPERATION_SUCCEEDED** The requested operation was performed atomically - no
4 callback function will be executed
5 **PMIX_ERR_INVALID_OPERATION** The requested operation is supported by the
6 implementation and host environment, but fails to meet a requirement (e.g., requesting to
7 *disconnect* from processes without first *connecting* to them).

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

9 **PMIX_ERR_NODE_DOWN** Node down
10 **PMIX_ERR_NODE_OFFLINE** Node is marked as offline
11 **PMIX_ERR_SYS_OTHER** Mark the beginning of a dedicated range of constants for system
12 event reporting.

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

14 **PMIX_EVENT_NO_ACTION_TAKEN** Event handler: No action taken
15 **PMIX_EVENT_PARTIAL_ACTION_TAKEN** Event handler: Partial action taken
16 **PMIX_EVENT_ACTION_DEFERRED** Event handler: Action deferred
17 **PMIX_EVENT_ACTION_COMPLETE** Event handler: Action complete

18 **3.1.1.3 User-Defined Error Constants**

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

22 **PMIX_EXTERNAL_ERR_BASE** A starting point for user-level defined error constants.
23 Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions
24 should always be based on the **PMIX_EXTERNAL_ERR_BASE** constant and *not* a specific
25 value as the value of the constant may change.

26 **3.1.2 Macros for use with PMIx constants**

27 **3.1.2.1 Detect system event constant**

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

PMIx v2.2

1 **PMIX_SYSTEM_EVENT** (a) C

2 **IN** a C

3 Error constant to be checked (**pmix_status_t**)

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

5 reporting

6 3.2 Data Types

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

8 3.2.1 Key Structure

9 The **pmix_key_t** structure is a statically defined character array of length **PMIX_MAX_KEYLEN**

10 +1, thus supporting keys of maximum length **PMIX_MAX_KEYLEN** while preserving space for a

11 mandatory **NULL** terminator.

PMIx v2.0

12 **typedef char pmix_key_t [PMIX_MAX_KEYLEN+1];**

13 Characters in the key must be standard alphanumeric values supported by common utilities such as

14 *strcmp*.

Advice to users

15 References to keys in PMIx v1 rwere defined simply as an array of characters of size

16 **PMIX_MAX_KEYLEN+1**. The **pmix_key_t** type definition was introduced in version 2 of the

17 standard. The two definitions are code-compatible and thus do not represent a break in backward

18 compatibility.

19 Passing a **pmix_key_t** value to the standard *sizeof* utility can result in compiler warnings of

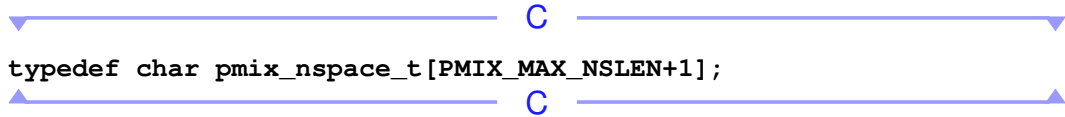
20 incorrect returned value. Users are advised to avoid using *sizeof(pmix_key_t)* and instead rely on

21 the **PMIX_MAX_KEYLEN** constant.

1 3.2.2 Namespace Structure

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

PMIx v2.0



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

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

 Advice to users 

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

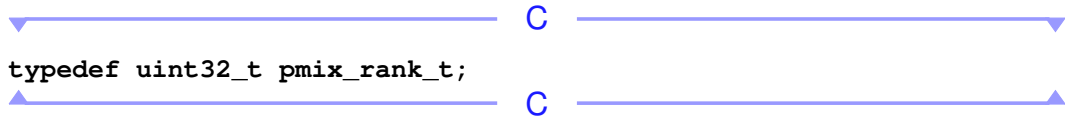
12 Passing a `pmix_namespace_t` value to the standard *sizeof* utility can result in compiler warnings of
13 incorrect returned value. Users are advised to avoid using *sizeof(pmix_namespace_t)* and instead rely
14 on the `PMIX_MAX_NSLEN` constant.



15 3.2.3 Rank Structure

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

PMIx v1.0



```
17 typedef uint32_t pmix_rank_t;
```

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

21 **PMIX_RANK_UNDEF** A value to request job-level data where the information itself is not
22 associated with any specific rank, or when passing a `pmix_proc_t` identifier to an
23 operation that only references the namespace field of that structure.

24 **PMIX_RANK_WILDCARD** A value to indicate that the user wants the data for the given key
25 from every rank that posted that key.

26 **PMIX_RANK_LOCAL_NODE** Special rank value used to define groups of ranks. This constant
27 defines the group of all ranks on a local node.

- 1 **PMIX_RANK_LOCAL_PEERS** Special rank value used to define groups of ranks. This
- 2 constant defines the group of all ranks on a local node within the same namespace as the
- 3 current process.
- 4 **PMIX_RANK_INVALID** An invalid rank value.
- 5 **PMIX_RANK_VALID** Define an upper boundary for valid rank values.

6 3.2.4 Process Structure

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

```

PMIx v1.0  ▼────────────────────────────────────────── C ───────────────────────────────────▶
9 typedef struct pmix_proc {
10     pmix_namespace_t nspace;
11     pmix_rank_t rank;
12 } pmix_proc_t;
    ▲────────────────────────────────────────── C ───────────────────────────────────▶
  
```

13 3.2.5 Process structure support macros

14 The following macros are provided to support the **pmix_proc_t** structure.

15 3.2.5.1 Initialize the **pmix_proc_t** structure

16 Initialize the **pmix_proc_t** fields

```

PMIx v1.0  ▼────────────────────────────────────────── C ───────────────────────────────────▶
17 PMIX_PROC_CONSTRUCT (m)
    ▲────────────────────────────────────────── C ───────────────────────────────────▶
  
```

18 **IN** m
 19 Pointer to the structure to be initialized (pointer to **pmix_proc_t**)

20 3.2.5.2 Destruct the **pmix_proc_t** structure

21 Clear the **pmix_proc_t** fields

```

PMIx v1.0  ▼────────────────────────────────────────── C ───────────────────────────────────▶
22 PMIX_PROC_DESTRUCT (m)
    ▲────────────────────────────────────────── C ───────────────────────────────────▶
  
```

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

25 This macro performs the identical operations as **PMIX_PROC_CONSTRUCT**, but is provided for
 26 symmetry in user code.

1 3.2.5.3 Create a `pmix_proc_t` array

2 Allocate and initialize an array of `pmix_proc_t` structures

PMIx v1.0

▼ `PMIX_PROC_CREATE` C

3 `PMIX_PROC_CREATE` (`m`, `n`)

▲

4 **INOUT** `m`

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

6 **IN** `n`

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

8 3.2.5.4 Free a `pmix_proc_t` array

9 Release an array of `pmix_proc_t` structures

PMIx v1.0

▼ `PMIX_PROC_FREE` C

10 `PMIX_PROC_FREE` (`m`, `n`)

▲

11 **IN** `m`

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

13 **IN** `n`

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

15 3.2.5.5 Load a `pmix_proc_t` structure

16 Load values into a `pmix_proc_t`

PMIx v2.0

▼ `PMIX_PROC_LOAD` C

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

▲

18 **IN** `m`

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

20 **IN** `n`

21 Namespace to be loaded (`pmix_namespace_t`)

22 **IN** `r`

23 Rank to be assigned (`pmix_rank_t`)

1 3.2.6 Process State Structure

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

Advice to users

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

8	<code>PMIX_PROC_STATE_UNDEF</code>	Undefined process state
9	<code>PMIX_PROC_STATE_PREPPED</code>	Process is ready to be launched
10	<code>PMIX_PROC_STATE_LAUNCH_UNDERWAY</code>	Process launch is underway
11	<code>PMIX_PROC_STATE_RESTART</code>	Process is ready for restart
12	<code>PMIX_PROC_STATE_TERMINATE</code>	Process is marked for termination
13	<code>PMIX_PROC_STATE_RUNNING</code>	Process has been locally <code>fork</code> 'ed by the RM
14	<code>PMIX_PROC_STATE_CONNECTED</code>	Process has connected to PMIx server
15	<code>PMIX_PROC_STATE_UNTERMINATED</code>	Define a "boundary" between the terminated states 16 and <code>PMIX_PROC_STATE_CONNECTED</code> so users can easily and quickly determine if a 17 process is still running or not. Any value less than this constant means that the process has 18 not terminated.
19	<code>PMIX_PROC_STATE_TERMINATED</code>	Process has terminated and is no longer running
20	<code>PMIX_PROC_STATE_ERROR</code>	Define a boundary so users can easily and quickly determine if 21 a process abnormally terminated. Any value above this constant means that the process has 22 terminated abnormally.
23	<code>PMIX_PROC_STATE_KILLED_BY_CMD</code>	Process was killed by a command
24	<code>PMIX_PROC_STATE_ABORTED</code>	Process was aborted by a call to <code>PMIx_Abort</code>
25	<code>PMIX_PROC_STATE_FAILED_TO_START</code>	Process failed to start
26	<code>PMIX_PROC_STATE_ABORTED_BY_SIG</code>	Process aborted by a signal
27	<code>PMIX_PROC_STATE_TERM_WO_SYNC</code>	Process exited without calling <code>PMIx_Finalize</code>
28	<code>PMIX_PROC_STATE_COMM_FAILED</code>	Process communication has failed
29	<code>PMIX_PROC_STATE_CALLED_ABORT</code>	Process called <code>PMIx_Abort</code>
30	<code>PMIX_PROC_STATE_MIGRATING</code>	Process failed and is waiting for resources before 31 restarting
32	<code>PMIX_PROC_STATE_CANNOT_RESTART</code>	Process failed and cannot be restarted
33	<code>PMIX_PROC_STATE_TERM_NON_ZERO</code>	Process exited with a non-zero status
34	<code>PMIX_PROC_STATE_FAILED_TO_LAUNCH</code>	Unable to launch process

1 3.2.7 Process Information Structure

2 The `pmix_proc_info_t` structure defines a set of information about a specific process
3 including its name, location, and state.

PMIx v2.0

C

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

C

18 3.2.8 Process Information Structure support macros

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

20 3.2.8.1 Initialize the `pmix_proc_info_t` structure

21 Initialize the `pmix_proc_info_t` fields

PMIx v2.0

C

```
22 PMIX_PROC_INFO_CONSTRUCT(m)
```

C

23 **IN** m

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

1 3.2.8.2 Destruct the `pmix_proc_info_t` structure

2 Destruct the `pmix_proc_info_t` fields

PMIx v2.0

▼  C 

3 **PMIX_PROC_INFO_DESTRUCT** (m)

▲  C 

4 **IN** m

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

6 3.2.8.3 Create a `pmix_proc_info_t` array

7 Allocate and initialize a `pmix_proc_info_t` array

PMIx v2.0

▼  C 

8 **PMIX_PROC_INFO_CREATE** (m, n)

▲  C 

9 **INOUT** m

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

12 **IN** n

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

14 3.2.8.4 Free a `pmix_proc_info_t` array

15 Release an array of `pmix_proc_info_t` structures

PMIx v2.0

▼  C 

16 **PMIX_PROC_INFO_FREE** (m, n)

▲  C 

17 **IN** m

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

19 **IN** n

20 Number of structures in the array (**size_t**)

1 3.2.9 Scope of Put Data

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

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

8 `PMIX_SCOPE_UNDEF` Undefined scope
9 `PMIX_LOCAL` The data is intended only for other application processes on the same node.
10 Data marked in this way will not be included in data packages sent to remote requestors —
11 i.e., it is only available to processes on the local node.
12 `PMIX_REMOTE` The data is intended solely for applications processes on remote nodes. Data
13 marked in this way will not be shared with other processes on the same node — i.e., it is only
14 available to processes on remote nodes.
15 `PMIX_GLOBAL` The data is to be shared with all other requesting processes, regardless of
16 location.
17 *PMIx v2.0* `PMIX_INTERNAL` The data is intended solely for this process and is not shared with other
18 processes.

19 3.2.10 Range of Published Data

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

25 `PMIX_RANGE_UNDEF` Undefined range
26 `PMIX_RANGE_RM` Data is intended for the host resource manager.
27 `PMIX_RANGE_LOCAL` Data is only available to processes on the local node.
28 `PMIX_RANGE_NAMESPACE` Data is only available to processes in the same namespace.
29 `PMIX_RANGE_SESSION` Data is only available to all processes in the session.
30 `PMIX_RANGE_GLOBAL` Data is available to all processes.
31 `PMIX_RANGE_CUSTOM` Range is specified in the `pmix_info_t` associated with this call.
32 `PMIX_RANGE_PROC_LOCAL` Data is only available to this process.

Advice to users

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

1 3.2.11 Data Persistence Structure

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

6 **PMIX_PERSIST_INDEF** Retain data until specifically deleted.
7 **PMIX_PERSIST_FIRST_READ** Retain data until the first access, then the data is deleted.
8 **PMIX_PERSIST_PROC** Retain data until the publishing process terminates.
9 **PMIX_PERSIST_APP** Retain data until the application terminates.
10 **PMIX_PERSIST_SESSION** Retain data until the session/allocation terminates.

11 3.2.12 Data Array Structure

PMIx v2.0

```
12 typedef struct pmix_data_array  
13     pmix_data_type_t type;  
14     size_t size;  
15     void *array;  
16     pmix_data_array_t;
```

C

C

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

19 3.2.13 Data array structure support macros

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

21 3.2.13.1 Initialize the `pmix_data_array_t` structure

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

PMIx v2.2

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

C

C

24 **IN** m
25 Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)
26 **IN** n
27 Number of elements in the array (`size_t`)
28 **IN** t
29 PMIx data type for the array elements (`pmix_data_type_t`)

1 3.2.13.2 Destruct the `pmix_data_array_t` structure

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

PMIx v2.2

▼  ▼

3 `PMIX_DATA_ARRAY_DESTRUCT (m)`

▲  ▲

4 **IN** `m`

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

6 3.2.13.3 Create and initialize a `pmix_data_array_t` object

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

PMIx v2.2

▼  ▼

9 `PMIX_DATA_ARRAY_CREATE (m, n, t)`

▲  ▲

10 **INOUT** `m`

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

12 **IN** `n`

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

14 **IN** `t`

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

16 3.2.13.4 Free a `pmix_data_array_t` object

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

PMIx v2.2

▼  ▼

18 `PMIX_DATA_ARRAY_FREE (m)`

▲  ▲

19 **IN** `m`

20 Pointer to the `pmix_data_array_t` structure (handle)

1 3.2.14 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;

```



3.2.15 Value structure support macros

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

3.2.15.1 Initialize the `pmix_value_t` structure

Initialize the `pmix_value_t` fields

PMIx v1.0



PMIX_VALUE_CONSTRUCT (m)



IN m

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

3.2.15.2 Destruct the `pmix_value_t` structure

Destruct the `pmix_value_t` fields

PMIx v1.0



PMIX_VALUE_DESTRUCT (m)



IN m

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

3.2.15.3 Create a `pmix_value_t` array

Allocate and initialize an array of `pmix_value_t` structures

PMIx v1.0



PMIX_VALUE_CREATE (m, n)



INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

1 3.2.15.4 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 3.2.15.5 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 3.2.15.6 Unload a `pmix_value_t` structure

2 Summary

3 Unload data from a `pmix_value_t` structure.

PMIx v2.2

C

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

C

5 **OUT** `r`

6 Status code indicating result of the operation `pmix_status_t`

7 **IN** `v`

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

9 **INOUT** `d`

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

11 **INOUT** `t`

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

13 Description

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

▼ Advice to users ▼

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

17 3.2.15.7 Transfer data between `pmix_value_t` structures

18 Summary

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

PMIx v2.0

C

20 `PMIX_VALUE_XFER(r, d, s);`

C

21 **OUT** `r`

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

23 **IN** `d`

24 Pointer to the `pmix_value_t` destination (handle)

25 **IN** `s`

26 Pointer to the `pmix_value_t` source (handle)

Description

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

Advice to users

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

3.2.16 Info Structure

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

PMIx v1.0

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

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

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

PMIx v1.0

```
typedef struct pmix_info_array {  
    size_t size;  
    pmix_info_t *array;  
} pmix_info_array_t;
```

3.2.17 Info structure support macros

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

1 3.2.17.1 Initialize the `pmix_info_t` structure

2 Initialize the `pmix_info_t` fields

PMIx v1.0

▼  C 

3 **PMIX_INFO_CONSTRUCT** (m)

▲  C 

4 **IN** m

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

6 3.2.17.2 Destruct the `pmix_info_t` structure

7 Destruct the `pmix_info_t` fields

PMIx v1.0

▼  C 

8 **PMIX_INFO_DESTRUCT** (m)

▲  C 

9 **IN** m

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

11 3.2.17.3 Create a `pmix_info_t` array

12 Allocate and initialize an array of `pmix_info_t` structures

PMIx v1.0

▼  C 

13 **PMIX_INFO_CREATE** (m, n)

▲  C 

14 **INOUT** m

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

16 **IN** n

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

18 3.2.17.4 Free a `pmix_info_t` array

19 Release an array of `pmix_info_t` structures

PMIx v1.0

▼  C 

20 **PMIX_INFO_FREE** (m, n)

▲  C 

21 **IN** m

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

23 **IN** n

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

1 3.2.17.5 Load key and value data into a `pmix_info_t`

PMIx v1.0

C

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

C

3 **IN** `v`

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

6 **IN** `k`

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

9 **IN** `d`

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

11 **IN** `t`

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

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

Advice to users

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

18 3.2.17.6 Copy data between `pmix_info_t` structures

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

PMIx v2.0

C

20 `PMIX_INFO_XFER(d, s);`

C

21 **IN** `d`

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

23 **IN** `s`

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

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

Advice to users

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

1 3.2.17.7 Test a boolean `pmix_info_t`

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

PMIx v2.0

▼ C ▲

3 `PMIX_INFO_TRUE(m)`

▲ C ▼

4 **IN** `m`

5 Pointer to a `pmix_info_t` structure (handle)

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

- 7
- the structure reports a type of `PMIX_UNDEF`, or
 - the structure reports a type of `PMIX_BOOL` and the data flag is `true`
- 8

9 3.2.18 Info Type Directives

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

▼ Advice to users ▲

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

▲

▼ Advice to PMIx library implementers ▲

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

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

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

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

Advice to PMIx server hosts

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

10 3.2.19 Info Directive support macros

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

12 3.2.19.1 Mark an info structure as required

13 Summary

14 Set the **PMIX_INFO_REQD** flag in a `pmix_info_t` structure.

PMIx v2.0

```
15 PMIX_INFO_REQUIRED(info);
```

16 **IN** `info`

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

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

1 3.2.19.2 Mark an info structure as optional

2 Summary

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

PMIx v2.2

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

5 **IN** `info`

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

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

8 3.2.19.3 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 3.2.19.4 Test an info structure for unset *required* directive

16 Summary

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

PMIx v2.0

```
▼ _____ C _____ ▼  
19 PMIX_INFO_IS_OPTIONAL(info);  
▲ _____ C _____ ▲
```

20 **IN** `info`

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

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

1 3.2.19.5 Test an info structure for *end of array* directive

2 Summary

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

PMIx v2.2

```
5 PMIX_INFO_IS_END(info);
```

6 **IN** `info`

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

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

9 3.2.20 Job Allocation Directives

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

14 **PMIX_ALLOC_NEW** A new allocation is being requested. The resulting allocation will be
15 disjoint (i.e., not connected in a job sense) from the requesting allocation.

16 **PMIX_ALLOC_EXTEND** Extend the existing allocation, either in time or as additional
17 resources.

18 **PMIX_ALLOC_RELEASE** Release part of the existing allocation. Attributes in the
19 accompanying `pmix_info_t` array may be used to specify permanent release of the
20 identified resources, or “lending” of those resources for some period of time.

21 **PMIX_ALLOC_REAQUIRE** Reacquire resources that were previously “lent” back to the
22 scheduler.

23 **PMIX_ALLOC_EXTERNAL** A value boundary above which implementers are free to define
24 their own directive values.

25 3.2.21 Lookup Returned Data Structure

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

PMIx v1.0

```
27 typedef struct pmix_pdata {  
28     pmix_proc_t proc;  
29     pmix_key_t key;  
30     pmix_value_t value;  
31 } pmix_pdata_t;
```

1 3.2.22 Lookup data structure support macros

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

3 3.2.22.1 Initialize the `pmix_pdata_t` structure

4 Initialize the `pmix_pdata_t` fields

PMIx v1.0

▼ `PMIX_PDATALIST_INITIALIZE` C

5 `PMIX_PDATALIST_INITIALIZE (m)`

▲ `PMIX_PDATALIST_INITIALIZE` C

6 **IN** `m`

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

8 3.2.22.2 Destruct the `pmix_pdata_t` structure

9 Destruct the `pmix_pdata_t` fields

PMIx v1.0

▼ `PMIX_PDATALIST_DESTRUCT` C

10 `PMIX_PDATALIST_DESTRUCT (m)`

▲ `PMIX_PDATALIST_DESTRUCT` C

11 **IN** `m`

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

13 3.2.22.3 Create a `pmix_pdata_t` array

14 Allocate and initialize an array of `pmix_pdata_t` structures

PMIx v1.0

▼ `PMIX_PDATALIST_CREATE` C

15 `PMIX_PDATALIST_CREATE (m, n)`

▲ `PMIX_PDATALIST_CREATE` C

16 **INOUT** `m`

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

19 **IN** `n`

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

1 3.2.22.4 Free a `pmix_pdata_t` array

2 Release an array of `pmix_pdata_t` structures

PMIx v1.0

▼  

3 **PMIX_PDATA_FREE**(*m*, *n*)

▲  

4 **IN** *m*

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

6 **IN** *n*

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

8 3.2.22.5 Load a lookup data structure

9 Summary

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

PMIx v1.0

▼  

11 **PMIX_PDATA_LOAD**(*m*, *p*, *k*, *d*, *t*);

▲  

12 **IN** *m*

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

15 **IN** *p*

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

18 **IN** *k*

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

21 **IN** *d*

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

23 **IN** *t*

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

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

▼ **Advice to users** 

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

▲ 

1 3.2.22.6 Transfer a lookup data structure

2 Summary

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

PMIx v2.0

C

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

C

5 **IN** `d`

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

7 **IN** `s`

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

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

Advice to users

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

13 3.2.23 Application Structure

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

PMIx v1.0

C

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

C

1 3.2.24 App structure support macros

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

3 3.2.24.1 Initialize the `pmix_app_t` structure

4 Initialize the `pmix_app_t` fields

PMIx v1.0

C

5 **PMIX_APP_CONSTRUCT** (m)

C

6 **IN** m

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

8 3.2.24.2 Destruct the `pmix_app_t` structure

9 Destruct the `pmix_app_t` fields

PMIx v1.0

C

10 **PMIX_APP_DESTRUCT** (m)

C

11 **IN** m

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

13 3.2.24.3 Create a `pmix_app_t` array

14 Allocate and initialize an array of `pmix_app_t` structures

PMIx v1.0

C

15 **PMIX_APP_CREATE** (m, n)

C

16 **INOUT** m

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

18 **IN** n

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

1 3.2.24.4 Free a `pmix_app_t` array

2 Release an array of `pmix_app_t` structures

PMIx v1.0

▼  

3 **PMIX_APP_FREE**(*m*, *n*)

▲  

4 **IN** *m*

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

6 **IN** *n*

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

8 3.2.24.5 Create the `pmix_info_t` array of application directives

9 Create an array of `pmix_info_t` structures for passing application-level directives, updating the
10 *ninfo* field of the `pmix_app_t` structure.

PMIx v2.2

▼  

11 **PMIX_APP_INFO_CREATE**(*m*, *n*)

▲  

12 **IN** *m*

13 Pointer to the `pmix_app_t` structure (handle)

14 **IN** *n*

15 Number of directives to be allocated (**size_t**)

16 3.2.25 Query Structure

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

PMIx v2.0

▼  

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

▲  

24 3.2.26 Query structure support macros

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

1 3.2.26.1 Initialize the `pmix_query_t` structure

2 Initialize the `pmix_query_t` fields

PMIx v2.0

▼  C  ▼

3 **PMIX_QUERY_CONSTRUCT** (m)

▲  C  ▲

4 **IN** m

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

6 3.2.26.2 Destruct the `pmix_query_t` structure

7 Destruct the `pmix_query_t` fields

PMIx v2.0

▼  C  ▼

8 **PMIX_QUERY_DESTRUCT** (m)

▲  C  ▲

9 **IN** m

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

11 3.2.26.3 Create a `pmix_query_t` array

12 Allocate and initialize an array of `pmix_query_t` structures

PMIx v2.0

▼  C  ▼

13 **PMIX_QUERY_CREATE** (m, n)

▲  C  ▲

14 **INOUT** m

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

17 **IN** n

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

19 3.2.26.4 Free a `pmix_query_t` array

20 Release an array of `pmix_query_t` structures

PMIx v2.0

▼  C  ▼

21 **PMIX_QUERY_FREE** (m, n)

▲  C  ▲

22 **IN** m

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

24 **IN** n

25 Number of structures in the array (**size_t**)

1 3.2.26.5 Create the `pmix_info_t` array of query qualifiers

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

PMIx v2.2

```
▼ _____ C _____ ▼  
4 PMIX_QUERY_QUALIFIERS_CREATE(m, n)  
▲ _____ C _____ ▲
```

5 **IN** m
6 Pointer to the `pmix_query_t` structure (handle)
7 **IN** n
8 Number of qualifiers to be allocated (`size_t`)

9 3.2.27 Modex Structure

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

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

PMIx v1.0

```
▼ _____ C _____ ▼  
▲ _____ C _____ ▲  
13 typedef struct pmix_modex_data {  
14     pmix_nspace_t nspace;  
15     int rank;  
16     uint8_t *blob;  
17     size_t size;  
18 } pmix_modex_data_t;  
▲ _____ C _____ ▲
```

19 3.2.28 Modex data structure support macros

20 The following macros are provided to support the `pmix_modex_data_t` structure.

1 3.2.28.1 Initialize the `pmix_modex_data_t` structure

2 Initialize the `pmix_modex_data_t` fields

PMIx v1.0

▼  

3 **PMIX_MODEX_CONSTRUCT** (m)

▲  

4 **IN** m

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

6 3.2.28.2 Destruct the `pmix_modex_data_t` structure

7 Destruct the `pmix_modex_data_t` fields

PMIx v1.0

▼  

8 **PMIX_MODEX_DESTRUCT** (m)

▲  

9 **IN** m

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

11 3.2.28.3 Create a `pmix_modex_data_t` array

12 Allocate and initialize an array of `pmix_modex_data_t` structures

PMIx v1.0

▼  

13 **PMIX_MODEX_CREATE** (m, n)

▲  

14 **INOUT** m

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

17 **IN** n

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

19 3.2.28.4 Free a `pmix_modex_data_t` array

20 Release an array of `pmix_modex_data_t` structures

PMIx v1.0

▼  

21 **PMIX_MODEX_FREE** (m, n)

▲  

22 **IN** m

23 Pointer to the array of `pmix_modex_data_t` structures (handle)

24 **IN** n

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

1 3.3 Data Packing/Unpacking Types and Structures

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

4 3.3.1 Byte Object Type

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

PMIx v1.0

```
▼ _____ C _____ ▼  
6 typedef struct pmix_byte_object {  
7     char *bytes;  
8     size_t size;  
9 } pmix_byte_object_t;  
▲ _____ C _____ ▲
```

10 3.3.2 Byte object support macros

11 The following macros support the `pmix_byte_object_t` structure.

12 3.3.2.1 Initialize the `pmix_byte_object_t` structure

13 Initialize the `pmix_byte_object_t` fields

PMIx v2.0

```
▼ _____ C _____ ▼  
14 PMIX_BYTE_OBJECT_CONSTRUCT(m)  
▲ _____ C _____ ▲
```

15 **IN** m

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

17 3.3.2.2 Destruct the `pmix_byte_object_t` structure

18 Clear the `pmix_byte_object_t` fields

PMIx v2.0

```
▼ _____ C _____ ▼  
19 PMIX_BYTE_OBJECT_DESTRUCT(m)  
▲ _____ C _____ ▲
```

20 **IN** m

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

1 3.3.2.3 Create a `pmix_byte_object_t` structure

2 Allocate and initialize an array of `pmix_byte_object_t` structures

PMIx v2.0

▼  ▼

3 `PMIX_BYTE_OBJECT_CREATE (m, n)`

▲  ▲

4 **INOUT** `m`

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

7 **IN** `n`

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

9 3.3.2.4 Free a `pmix_byte_object_t` array

10 Release an array of `pmix_byte_object_t` structures

PMIx v2.0

▼  ▼

11 `PMIX_BYTE_OBJECT_FREE (m, n)`

▲  ▲

12 **IN** `m`

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

14 **IN** `n`

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

16 3.3.2.5 Load a `pmix_byte_object_t` structure

17 Load values into a `pmix_byte_object_t`

PMIx v2.0

▼  ▼

18 `PMIX_BYTE_OBJECT_LOAD (b, d, s)`

▲  ▲

19 **IN** `b`

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

21 **IN** `d`

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

23 **IN** `s`

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

1 3.3.3 Data Buffer Type

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

PMIx v2.0

```
3 typedef struct pmix_data_buffer {  
4     /** Start of my memory */  
5     char *base_ptr;  
6     /** Where the next data will be packed to (within the allocated  
7     memory starting at base_ptr) */  
8     char *pack_ptr;  
9     /** Where the next data will be unpacked from (within the  
10    allocated memory starting as base_ptr) */  
11    char *unpack_ptr;  
12    /** Number of bytes allocated (starting at base_ptr) */  
13    size_t bytes_allocated;  
14    /** Number of bytes used by the buffer (i.e., amount of data --  
15    including overhead -- packed in the buffer) */  
16    size_t bytes_used;  
17 } pmix_data_buffer_t;
```

18 3.3.4 Data buffer support macros

19 The following macros support the `pmix_data_buffer_t` structure.

20 3.3.4.1 Initialize the `pmix_data_buffer_t` structure

21 Initialize the `pmix_data_buffer_t` fields

PMIx v2.0

```
22 PMIX_DATA_BUFFER_CONSTRUCT(m)
```

23 **IN** m

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

1 3.3.4.2 Destruct the `pmix_data_buffer_t` structure

2 Clear the `pmix_data_buffer_t` fields

PMIx v2.0

▼  

3 **PMIX_DATA_BUFFER_DESTRUCT** (m)

▲  

4 **IN** m

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

6 3.3.4.3 Create a `pmix_data_buffer_t` structure

7 Allocate and initialize a `pmix_data_buffer_t` structure

PMIx v2.0

▼  

8 **PMIX_DATA_BUFFER_CREATE** (m)

▲  

9 **INOUT** m

10 Address where the pointer to the `pmix_data_buffer_t` structure shall be stored
11 (handle)

12 3.3.4.4 Free a `pmix_data_buffer_t`

13 Release a `pmix_data_buffer_t` structure

PMIx v2.0

▼  

14 **PMIX_DATA_BUFFER_RELEASE** (m)

▲  

15 **IN** m

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

17 3.3.4.5 Load a `pmix_data_buffer_t`

18 Load data into a `pmix_data_buffer_t` structure

PMIx v2.2

▼  

19 **PMIX_DATA_BUFFER_LOAD** (b, d, s)

▲  

20 **IN** b

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

22 **IN** d

23 Pointer to the data to be loaded into *b* (**void***)

24 **IN** s

25 Number of bytes in *d* (**size_t**)

1 3.3.4.6 Unload a `pmix_data_buffer_t`

2 Unload the data from a `pmix_data_buffer_t` structure

PMIx v2.2

▼ `C` ————— ▼

3 `PMIX_DATA_BUFFER_UNLOAD(b, d, s)`

▲ `C` ————— ▲

4 **IN** `b`

5 Pointer to the `pmix_data_buffer_t` structure to be unloaded (handle)

6 **INOUT** `d`

7 Pointer to be set to the data region after unloading (**void***)

8 **INOUT** `s`

9 Variable to be set to the number of bytes in the returned data region (**size_t**)

10 3.3.5 Data Array Structure

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

PMIx v2.0

▼ `C` ————— ▼

```
12 typedef struct pmix_data_array {  
13     pmix_data_type_t type;  
14     size_t size;  
15     void *array;  
16 } pmix_data_array_t;
```

▲ `C` ————— ▲

17 3.3.6 Data array support macros

18 The following macros support the `pmix_data_array_t` structure.

19 3.3.6.1 Initialize a `pmix_data_array_t` structure

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

PMIx v2.2

▼ `C` ————— ▼

21 `PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)`

▲ `C` ————— ▲

22 **IN** `m`

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

24 **IN** `n`

25 Number of elements in the array (**size_t**)

26 **IN** `t`

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

1 3.3.6.2 Destruct a `pmix_data_array_t` structure

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

PMIx v2.2



3 **PMIX_DATA_ARRAY_CONSTRUCT** (m)



4 **IN** m

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

6 3.3.6.3 Create a `pmix_data_array_t` structure

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

PMIx v2.2



9 **PMIX_DATA_ARRAY_CREATE** (m, n, t)



10 **INOUT** m

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

12 **IN** n

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

14 **IN** t

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

16 3.3.6.4 Free a `pmix_data_array_t` structure

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

PMIx v2.2



18 **PMIX_DATA_ARRAY_FREE** (m)



19 **IN** m

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

1 3.3.7 Generalized Data Types Used for Packing/Unpacking

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

▼ Advice to PMIx library implementers ▼

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

9 3.3.7.1 PMIx v1 Data Types

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

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

1 **PMIX_BYTE_OBJECT** Byte object ([pmix_byte_object_t](#))
2 **PMIX_KVAL** Key/value pair
3 **PMIX_MODEX (Deprecated in PMIx 2.0)** Modex
4 **PMIX_PERSIST** Persistence ([pmix_persistence_t](#))
5 **PMIX_INFO_ARRAY (Deprecated in PMIx 2.0)** Info array

6 3.3.7.2 PMIx v2 Data Types

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

8 **PMIX_STATUS** Status ([pmix_status_t](#))
9 **PMIX_POINTER** Pointer (**void***)
10 **PMIX_SCOPE** Scope ([pmix_scope_t](#))
11 **PMIX_DATA_RANGE** Data range ([pmix_data_range_t](#))
12 **PMIX_COMMAND** Command
13 **PMIX_INFO_DIRECTIVES** Info directives
14 **PMIX_DATA_TYPE** Data type
15 **PMIX_PROC_STATE** Process state ([pmix_proc_state_t](#))
16 **PMIX_PROC_INFO** Process info ([pmix_proc_info_t](#))
17 **PMIX_DATA_ARRAY** Data array ([pmix_data_array_t](#))
18 **PMIX_PROC_RANK** Process rank ([pmix_rank_t](#))
19 **PMIX_QUERY** Query
20 **PMIX_COMPRESSED_STRING** Compressed string (with zlib)
21 **PMIX_ALLOC_DIRECTIVE** Allocation directive ([pmix_alloc_directive_t](#))
22 **PMIX_DATA_TYPE_MAX** A boundary for implementers above which they can add their own
23 data types.

24 3.4 Reserved attributes

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

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

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

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

4 **PMIX_ATTR_UNDEF NULL (NULL)**
5 Constant representing an undefined attribute.

6 3.4.1 Initialization attributes

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

9 **PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)**
10 Pointer to libevent¹ **event_base** to use in place of the internal progress thread.
11 **PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)**
12 The host RM wants to declare itself as willing to accept tool connection requests.
13 **PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)**
14 Allow connections from remote tools. Forces the PMIx server to not exclusively use
15 loopback device.
16 **PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)**
17 The host RM wants to declare itself as being the local system server for PMIx connection
18 requests.
19 **PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)**
20 Top-level temporary directory for all *client* processes connected to this server, and where the
21 PMIx server will place its *tool* rendezvous point and contact information.
22 **PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)**
23 Temporary directory for this system, and where a PMIx server that declares itself to be a
24 system-level server will place a *tool* rendezvous point and contact information.
25 **PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)**
26 Registration is for the namespace only. Do not copy job data.
27 **PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)**
28 Enable PMIx internal monitoring by the PMIx server.
29 **PMIX_SERVER_NAMESPACE "pmix.srv.namespace" (char*)**
30 Name of the namespace to use for this PMIx server.
31 **PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)**
32 Rank of this PMIx server

¹<http://libevent.org/>

1 3.4.2 Tool-related attributes

2 These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by
3 passing them into the `PMIx_tool_init` API - thus, they are not typically accessed via the
4 `PMIx_Get` API.

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

6 Name of the namespace to use for this tool.

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

8 Rank of this tool.

9 **PMIX_SERVER_PIDINFO** `"pmix.srvr.pidinfo"` (`pid_t`)

10 PID of the target PMIx server for a tool.

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

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

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

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

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

16 uniform resource identifier (URI) of the PMIx server to be contacted.

17 **PMIX_SERVER_HOSTNAME** `"pmix.srvr.host"` (`char*`)

18 Host where target PMIx server is located.

19 **PMIX_CONNECT_MAX_RETRIES** `"pmix.tool.mretries"` (`uint32_t`)

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

21 **PMIX_CONNECT_RETRY_DELAY** `"pmix.tool.retry"` (`uint32_t`)

22 Time in seconds between connection attempts to a PMIx server.

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

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

25 3.4.3 Identification attributes

26 These attributes are defined to identify a process and it's associated PMIx-enabled library. They are
27 not typically accessed via the `PMIx_Get` API, and thus are not associated with a particular rank.

28 **PMIX_USERID** `"pmix.euid"` (`uint32_t`)

29 Effective user id.

30 **PMIX_GRPID** `"pmix.egid"` (`uint32_t`)

31 Effective group id.

32 **PMIX_DSTPATH** `"pmix.dstpath"` (`char*`)

33 Path to shared memory data storage (dstore) files.

34 **PMIX_VERSION_INFO** `"pmix.version"` (`char*`)

35 PMIx version of contractor.

36 **PMIX_PROGRAMMING_MODEL** `"pmix.pgm.model"` (`char*`)

37 Programming model being initialized (e.g., "MPI" or "OpenMP")

38 **PMIX_MODEL_LIBRARY_NAME** `"pmix.mdl.name"` (`char*`)

1 Programming model implementation ID (e.g., “OpenMPI” or “MPICH”)
 2 **PMIX_MODEL_LIBRARY_VERSION** "pmix.mld.vrs" (char*)
 3 Programming model version string (e.g., “2.1.1”)
 4 **PMIX_THREADING_MODEL** "pmix.threads" (char*)
 5 Threading model used (e.g., “pthreads”)
 6 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
 7 The requesting process is a PMIx tool.
 8 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
 9 The requesting process is a PMIx client.

10 3.4.4 UNIX socket rendezvous socket attributes

11 These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing
 12 them into the relevant initialization API - thus, they are not typically accessed via the **PMIx_Get**
 13 API.

14 **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)
 15 Disable legacy UNIX socket (usock) support
 16 **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
 17 POSIX *mode_t* (9 bits valid)
 18 **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)
 19 Use only one rendezvous socket, letting priorities and/or environment parameters select the
 20 active transport.

21 3.4.5 TCP connection attributes

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

25 **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)
 26 If provided, directs that the TCP URI be reported and indicates the desired method of
 27 reporting: '-' for stdout, '+' for stderr, or filename.
 28 **PMIX_TCP_URI** "pmix.tcp.uri" (char*)
 29 The URI of the PMIx server to connect to, or a file name containing it in the form of
 30 **file:<name of file containing it>**.
 31 **PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)
 32 Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to
 33 include when establishing the TCP connection.
 34 **PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)
 35 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
 36 TCP connection.
 37 **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)

1 The IPv4 port to be used.
2 **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)
3 The IPv6 port to be used.
4 **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)
5 Set to **true** to disable IPv4 family of addresses.
6 **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)
7 Set to **true** to disable IPv6 family of addresses.

8 **3.4.6 Global Data Storage (GDS) attributes**

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

12 **PMIX_GDS_MODULE** "pmix.gds.mod" (char*)
13 Comma-delimited string of desired modules.

14 **3.4.7 General process-level attributes**

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

16 **PMIX_CPUSSET** "pmix.cpuset" (char*)
17 hwloc² bitmap to be applied to the process upon launch.
18 **PMIX_CREDENTIAL** "pmix.cred" (char*)
19 Security credential assigned to the process.
20 **PMIX_SPAWNED** "pmix.spawned" (bool)
21 **true** if this process resulted from a call to **PMIx_Spawn** .
22 **PMIX_ARCH** "pmix.arch" (uint32_t)
23 Architecture flag.

24 **3.4.8 Scratch directory attributes**

25 These attributes are used to define an application scratch directory and are referenced using the
26 **PMIX_RANK_WILDCARD** rank.

27 **PMIX_TMPDIR** "pmix.tmpdir" (char*)
28 Full path to the top-level temporary directory assigned to the session.
29 **PMIX_NSDIR** "pmix.nsdire" (char*)
30 Full path to the temporary directory assigned to the namespace, under **PMIX_TMPDIR** .
31 **PMIX_PROCDIR** "pmix.pdire" (char*)
32 Full path to the subdirectory under **PMIX_NSDIR** assigned to the process.
33 **PMIX_TDIR_RMCLEAN** "pmix.tdire.rmclean" (bool)
34 Resource Manager will clean session directories

²<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, and
3 thus are referenced using the process rank except where noted.

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

5 Process identifier

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

7 Namespace of the job.

8 **PMIX_JOBID** "pmix.jobid" (char*)

9 Job identifier assigned by the scheduler.

10 **PMIX_APPNUM** "pmix.appnum" (uint32_t)

11 Application number within the job.

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

13 Process rank within the job.

14 **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)

15 Process rank spanning across all jobs in this session.

16 **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)

17 Process rank within this application.

18 **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)

19 Starting global rank of this job - referenced using [PMIX_RANK_WILDCARD](#) .

20 **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)

21 Local rank on this node within this job.

22 **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)

23 Process rank on this node spanning all jobs.

24 **PMIX_LOCALLDR** "pmix.lldr" (pmix_rank_t)

25 Lowest rank on this node within this job - referenced using [PMIX_RANK_WILDCARD](#) .

26 **PMIX_APPLDR** "pmix.aldr" (pmix_rank_t)

27 Lowest rank in this application within this job - referenced using [PMIX_RANK_WILDCARD](#) .

28 **PMIX_PROC_PID** "pmix.ppid" (pid_t)

29 PID of specified process.

30 **PMIX_SESSION_ID** "pmix.session.id" (uint32_t)

31 Session identifier - referenced using [PMIX_RANK_WILDCARD](#) .

32 **PMIX_NODE_LIST** "pmix.nlist" (char*)

33 Comma-delimited list of nodes running processes for the specified namespace - referenced
34 using [PMIX_RANK_WILDCARD](#) .

35 **PMIX_ALLOCATED_NODELIST** "pmix.alist" (char*)

36 Comma-delimited list of all nodes in this allocation regardless of whether or not they
37 currently host processes - referenced using [PMIX_RANK_WILDCARD](#) .

38 **PMIX_HOSTNAME** "pmix.hname" (char*)

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

40 **PMIX_NODEID** "pmix.nodeid" (uint32_t)

1 Node identifier where the specified process is located, expressed as the node's index
2 (beginning at zero) in the array resulting from expansion of the [PMIX_NODE_MAP](#) regular
3 expression for the [job](#)

4 **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
5 Comma-delimited list of ranks on this node within the specified namespace - referenced
6 using [PMIX_RANK_WILDCARD](#) .

7 **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)
8 Array of [pmix_proc_t](#) of all processes on the specified node - referenced using
9 [PMIX_RANK_WILDCARD](#) .

10 **PMIX_LOCAL_CPUSSETS** "pmix.lcpus" (char*)
11 Colon-delimited cpusets of local peers within the specified namespace - referenced using
12 [PMIX_RANK_WILDCARD](#) .

13 **PMIX_PROC_URI** "pmix.puri" (char*)
14 URI containing contact information for a given process.

15 **PMIX_LOCALITY** "pmix.loc" (uint16_t)
16 Relative locality of the specified process to the requestor.

17 **PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)
18 Process identifier of the parent process of the calling process.

19 3.4.10 Information retrieval attributes

20 The following attributes are used to specify the level of information (e.g., [session](#) , [job](#) , or
21 [application](#)) being requested where ambiguity may exist - see [5.1.5](#) for examples of their use.

22 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
23 Return information about the specified session. If information about a session other than the
24 one containing the requesting process is desired, then the attribute array must contain a
25 [PMIX_SESSION_ID](#) attribute identifying the desired target.

26 **PMIX_JOB_INFO** "pmix.job.info" (bool)
27 Return information about the specified job or namespace. If information about a job or
28 namespace other than the one containing the requesting process is desired, then the attribute
29 array must contain a [PMIX_JOBID](#) or [PMIX_NAMESPACE](#) attribute identifying the desired
30 target. Similarly, if information is requested about a job or namespace in a session other than
31 the one containing the requesting process, then an attribute identifying the target session
32 must be provided.

33 **PMIX_APP_INFO** "pmix.app.info" (bool)
34 Return information about the specified application. If information about an application other
35 than the one containing the requesting process is desired, then the attribute array must
36 contain a [PMIX_APPNUM](#) attribute identifying the desired target. Similarly, if information
37 is requested about an application in a job or session other than the one containing the
38 requesting process, then attributes identifying the target job and/or session must be provided.

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

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

4 **3.4.11 Information storage attributes**

5 The following attributes are used to assemble information by its level (e.g., **session** , **job** , or
6 **application**) for storage where ambiguity may exist - see 10.1.3.1 for examples of their use.

7 **PMIX_SESSION_INFO_ARRAY** "pmix.ssn.arr" (**pmix_data_array_t**)

8 Provide an array of **pmix_info_t** containing session-level information. The
9 **PMIX_SESSION_ID** attribute is *required* to be included in the array.

10 **PMIX_JOB_INFO_ARRAY** "pmix.job.arr" (**pmix_data_array_t**)

11 Provide an array of **pmix_info_t** containing job-level information. The
12 **PMIX_SESSION_ID** attribute of the **session** containing the **job** is *required* to be
13 included in the array whenever the PMIx server library may host multiple sessions (e.g.,
14 when executing with a host RM daemon). As information is registered one job (aka
15 namespace) at a time via the **PMIx_server_register_namespace** API, there is no
16 requirement that the array contain either the **PMIX_NAMESPACE** or **PMIX_JOBID** attributes
17 when used in that context (though either or both of them *may* be included). At least one of
18 the job identifiers *must* be provided in all other contexts where the job being referenced is
19 ambiguous.

20 **PMIX_APP_INFO_ARRAY** "pmix.app.arr" (**pmix_data_array_t**)

21 Provide an array of **pmix_info_t** containing app-level information. The **PMIX_NAMESPACE**
22 or **PMIX_JOBID** attributes of the **job** containing the application, plus its **PMIX_APPNUM**
23 attribute, are *required* to be included in the array when the array is *not* included as part of a
24 call to **PMIx_server_register_namespace** - i.e., when the job containing the
25 application is ambiguous. The job identification is otherwise optional.

26 **PMIX_NODE_INFO_ARRAY** "pmix.node.arr" (**pmix_data_array_t**)

27 Provide an array of **pmix_info_t** containing node-level information. At a minimum,
28 either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute is *required* to be included in the
29 array, though both *may* be included.

30 Note that these assemblages can be used hierarchically:

- 31 • a **PMIX_JOB_INFO_ARRAY** might contain multiple **PMIX_APP_INFO_ARRAY** elements,
32 each describing values for a specific application within the job
- 33 • a **PMIX_JOB_INFO_ARRAY** could contain a **PMIX_NODE_INFO_ARRAY** for each node
34 hosting processes from that job, each array describing job-level values for that node
- 35 • a **PMIX_SESSION_INFO_ARRAY** might contain multiple **PMIX_JOB_INFO_ARRAY**
36 elements, each describing a job executing within the session. Each job array could, in turn,
37 contain both application and node arrays, thus providing a complete picture of the active
38 operations within the allocation

Advice to PMIx library implementers

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

3.4.12 Size information attributes

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

PMIX_UNIV_SIZE "pmix.univ.size" (uint32_t)

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

PMIX_JOB_SIZE "pmix.job.size" (uint32_t)

Total number of processes in this job across all contained applications

PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)

Number of applications in this job.

PMIX_APP_SIZE "pmix.app.size" (uint32_t)

Number of processes in this application.

PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)

Number of processes in this job or application on this node.

PMIX_NODE_SIZE "pmix.node.size" (uint32_t)

Number of processes across all jobs on this node.

PMIX_MAX_PROCS "pmix.max.size" (uint32_t)

Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description.

PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)

Number of slots allocated in this context (session, namespace, application, or node). Note that this attribute is the equivalent to **PMIX_MAX_PROCS** used in the corresponding context - it is included in the Standard for historical reasons.

PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)

Number of nodes in this session, or that are currently executing processes from the associated namespace or application.

1 3.4.13 Memory information attributes

2 These attributes are used to describe memory available and used in the system - all are referenced
3 using [PMIX_RANK_WILDCARD](#) .

4 **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)

5 Total available physical memory on this node.

6 **PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (float)

7 Megabytes of memory currently used by the RM daemon.

8 **PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (float)

9 Average Megabytes of memory used by client processes.

10 3.4.14 Topology information attributes

11 These attributes are used to describe topology information in the PMIx universe - all are referenced
12 using [PMIX_RANK_WILDCARD](#) except where noted.

13 **PMIX_NET_TOPO** "pmix.ntopo" (char*)

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

15 **PMIX_LOCAL_TOPO** "pmix.ltopo" (char*)

16 XML representation of local node topology.

17 **PMIX_TOPOLOGY** "pmix.topo" (hwloc_topology_t)

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

19 **PMIX_TOPOLOGY_SIGNATURE** "pmix.toposig" (char*)

20 Topology signature string.

21 **PMIX_LOCALITY_STRING** "pmix.locstr" (char*)

22 String describing a process's bound location - referenced using the process's rank. The string
23 is of the form:

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

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

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

31 This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches
32 0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.

33 **PMIX_HWLOC_SHMEM_ADDR** "pmix.hwlocaddr" (size_t)

34 Address of the hwloc shared memory segment.

35 **PMIX_HWLOC_SHMEM_SIZE** "pmix.hwlocsize" (size_t)

36 Size of the hwloc shared memory segment.

37 **PMIX_HWLOC_SHMEM_FILE** "pmix.hwlocfile" (char*)

38 Path to the hwloc shared memory file.

39 **PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)

1 XML representation of local topology using hwloc's v1.x format.
2 **PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)
3 XML representation of local topology using hwloc's v2.x format.

4 3.4.15 Request-related attributes

5 These attributes are used to influence the behavior of various PMIx operations - they do not
6 represent values accessed using the **PMIx_Get** API.

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

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

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

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

13 **PMIX_IMMEDIATE** "pmix.immediate" (bool)

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

16 **PMIX_WAIT** "pmix.wait" (int)

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

19 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)

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

24 **PMIX_COLLECTIVE_ALGO_REQD** "pmix.calreqd" (bool)

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

26 **PMIX_NOTIFY_COMPLETION** "pmix.notecomp" (bool)

27 Notify the parent process upon termination of child job.

28 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

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

30 **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)

31 Value for calls to **PMIx_Publish**.

32 **PMIX_DATA_SCOPE** "pmix.scope" (pmix_scope_t)

33 Scope of the data to be found in a **PMIx_Get** call.

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

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

37 **PMIX_EMBED_BARRIER** "pmix.embed.barrier" (bool)

38 Execute a blocking fence operation before executing the specified operation. For example,
39 **PMIx_Finalize** does not include an internal barrier operation by default. This attribute
40 would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation.

1 **PMIX_JOB_TERM_STATUS** "pmix.job.term.status" (pmix_status_t)
2 Status to be returned upon job termination.
3 **PMIX_PROC_STATE_STATUS** "pmix.proc.state" (pmix_proc_state_t)
4 Process state

5 3.4.16 Server-to-PMIx library attributes

6 Attributes used by the host environment to pass data to its PMIx server library. The data will then
7 be parsed and provided to the local PMIx clients. These attributes are all referenced using
8 **PMIX_RANK_WILDCARD** except where noted.

9 **PMIX_REGISTER_NODATA** "pmix.reg.nodata" (bool)
10 Registration is for this namespace only, do not copy job data - this attribute is not accessed
11 using the **PMIx_Get**
12 **PMIX_PROC_DATA** "pmix.pdata" (pmix_data_array_t)
13 Array of process data. Starts with rank, then contains more data.
14 **PMIX_NODE_MAP** "pmix.nmap" (char*)
15 Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.
16 **PMIX_PROC_MAP** "pmix.pmap" (char*)
17 Regular expression describing processes on each node - see 10.1.3.1 for an explanation of its
18 generation.
19 **PMIX_ANL_MAP** "pmix.anlmap" (char*)
20 Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
21 **PMIX_APP_MAP_TYPE** "pmix.apmap.type" (char*)
22 Type of mapping used to layout the application (e.g., **cyclic**).
23 **PMIX_APP_MAP_REGEX** "pmix.apmap.regex" (char*)
24 Regular expression describing the result of the process mapping.

25 3.4.17 Server-to-Client attributes

26 Attributes used internally to communicate data from the PMIx server to the PMIx client - they do
27 not represent values accessed using the **PMIx_Get** API.

28 **PMIX_PROC_BLOB** "pmix.pblob" (pmix_byte_object_t)
29 Packed blob of process data.
30 **PMIX_MAP_BLOB** "pmix.mblob" (pmix_byte_object_t)
31 Packed blob of process location.

1 3.4.18 Event handler registration and notification attributes

2 Attributes to support event registration and notification - they are values passed to the event
3 registration and notification APIs and therefore are not accessed using the `PMIx_Get` API.

Advice to users

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

7 **PMIX_ERROR_NAME** "pmix.errname" (pmix_status_t)
8 Specific error to be notified

9 **PMIX_ERROR_GROUP_COMM** "pmix.errgroup.comm" (bool)
10 Set true to get comm errors notification

11 **PMIX_ERROR_GROUP_ABORT** "pmix.errgroup.abort" (bool)
12 Set true to get abort errors notification

13 **PMIX_ERROR_GROUP_MIGRATE** "pmix.errgroup.migrate" (bool)
14 Set true to get migrate errors notification

15 **PMIX_ERROR_GROUP_RESOURCE** "pmix.errgroup.resource" (bool)
16 Set true to get resource errors notification

17 **PMIX_ERROR_GROUP_SPAWN** "pmix.errgroup.spawn" (bool)
18 Set true to get spawn errors notification

19 **PMIX_ERROR_GROUP_NODE** "pmix.errgroup.node" (bool)
20 Set true to get node status notification

21 **PMIX_ERROR_GROUP_LOCAL** "pmix.errgroup.local" (bool)
22 Set true to get local errors notification

23 **PMIX_ERROR_GROUP_GENERAL** "pmix.errgroup.gen" (bool)
24 Set true to get notified of generic errors

25 **PMIX_ERROR_HANDLER_ID** "pmix.errhandler.id" (int)
26 Errhandler reference id of notification being reported

27 **PMIX_EVENT_HDLR_NAME** "pmix.evname" (char*)
28 String name identifying this handler.

29 **PMIX_EVENT_HDLR_FIRST** "pmix.evfirst" (bool)
30 Invoke this event handler before any other handlers.

31 **PMIX_EVENT_HDLR_LAST** "pmix.evlast" (bool)
32 Invoke this event handler after all other handlers have been called.

33 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfirstcat" (bool)
34 Invoke this event handler before any other handlers in this category.

35 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlastcat" (bool)
36 Invoke this event handler after all other handlers in this category have been called.

37 **PMIX_EVENT_HDLR_BEFORE** "pmix.evbefore" (char*)
38 Put this event handler immediately before the one specified in the (char*) value.

1 **PMIX_EVENT_HDLR_AFTER** "pmix.evafter" (char*)
 2 Put this event handler immediately after the one specified in the (char*) value.
 3 **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (bool)
 4 Prepend this handler to the precedence list within its category.
 5 **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)
 6 Append this handler to the precedence list within its category.
 7 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrangle" (pmix_data_array_t*)
 8 Array of **pmix_proc_t** defining range of event notification.
 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.
 13 **PMIX_EVENT_NON_DEFAULT** "pmix.evnondf" (bool)
 14 Event is not to be delivered to default event handlers.
 15 **PMIX_EVENT_RETURN_OBJECT** "pmix.evobject" (void *)
 16 Object to be returned whenever the registered callback function **cbfunc** is invoked. The
 17 object will *only* be returned to the process that registered it.
 18 **PMIX_EVENT_DO_NOT_CACHE** "pmix.evnocache" (bool)
 19 Instruct the PMIx server not to cache the event.
 20 **PMIX_EVENT_SILENT_TERMINATION** "pmix.evsilentterm" (bool)
 21 Do not generate an event when this job normally terminates.
 22 **PMIX_EVENT_PROXY** "pmix.evproxy" (pmix_proc_t*)
 23 PMIx server that sourced the event
 24 **PMIX_EVENT_TEXT_MESSAGE** "pmix.evtext" (char*)
 25 Text message suitable for output by recipient - e.g., describing the cause of the event

26 3.4.19 Fault tolerance attributes

27 Attributes to support fault tolerance behaviors - they are values passed to the event notification API
 28 and therefore are not accessed using the **PMIx_Get** API.

29 **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (bool)
 30 The RM intends to terminate this session.
 31 **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (bool)
 32 The RM intends to terminate this job.
 33 **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (bool)
 34 The RM intends to terminate all processes on this node.
 35 **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (bool)
 36 The RM intends to terminate just this process.
 37 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evtimeout" (int)
 38 The time in seconds before the RM will execute error response.
 39 **PMIX_EVENT_NO_TERMINATION** "pmix.evnoterm" (bool)

1 Indicates that the handler has satisfactorily handled the event and believes termination of the
2 application is not required.
3 **PMIX_EVENT_WANT_TERMINATION** "pmix.evterm" (bool)
4 Indicates that the handler has determined that the application should be terminated

5 3.4.20 Spawn attributes

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

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

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

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

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.ssn cwd" (bool)
28 Set the application's current working directory to the session working directory assigned by
29 the RM - when accessed using `PMIx_Get` , use the `PMIX_RANK_WILDCARD` value for
30 the rank to discover the session working directory assigned to the provided namespace

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

33 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
34 Timestamp output from applications.

35 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
36 Merge `stdout` and `stderr` streams from application processes.

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

39 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)
40 Mark the `argv` with the rank of the process.

41 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

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

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

6 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)
7 Do not oversubscribe the cpus.

8 **PMIX_REPORT_BINDINGS** "pmix.repbind" (bool)
9 Report bindings of the individual processes.

10 **PMIX_CPU_LIST** "pmix.cpulist" (char*)
11 List of cpus to use for this job - when accessed using `PMIx_Get` , use the
12 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
13 namespace

14 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)
15 Application supports recoverable operations.

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

18 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)
19 Maximum number of times to restart a job - when accessed using `PMIx_Get` , use the
20 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
21 namespace

22 3.4.21 Query attributes

23 Attributes used to describe `PMIx_Query_info_nb` behavior - these are values passed to the
24 **PMIx_Query_info_nb** API and therefore are not passed to the `PMIx_Get` API.

25 **PMIX_QUERY_REFRESH_CACHE** "pmix.qry.rfsh" (bool)
26 Retrieve updated information from server.

27 **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (char*)
28 Request a comma-delimited list of active namespaces.

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

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

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

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

38 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lhtable" (char*)

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

4 **PMIX_QUERY_LOCAL_ONLY** "`pmix.qry.local`" (`bool`)
5 Constrain the query to local information only.

6 **PMIX_QUERY_AUTHORIZATIONS** "`pmix.qry.auths`" (`bool`)
7 Return operations the PMIx tool is authorized to perform.

8 **PMIX_QUERY_SPAWN_SUPPORT** "`pmix.qry.spawn`" (`bool`)
9 Return a comma-delimited list of supported spawn attributes.

10 **PMIX_QUERY_DEBUG_SUPPORT** "`pmix.qry.debug`" (`bool`)
11 Return a comma-delimited list of supported debug attributes.

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

14 **PMIX_QUERY_REPORT_AVG** "`pmix.qry.avg`" (`bool`)
15 Report average values.

16 **PMIX_QUERY_REPORT_MINMAX** "`pmix.qry.minmax`" (`bool`)
17 Report minimum and maximum values.

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

20 **PMIX_TIME_REMAINING** "`pmix.time.remaining`" (`char*`)
21 Query number of seconds (`uint32_t`) remaining in allocation for the specified namespace.

22 3.4.22 Log attributes

23 Attributes used to describe `PMIx_Log_nb` behavior - these are values passed to the
24 `PMIx_Log_nb` API and therefore are not accessed using the `PMIx_Get` API.

25 **PMIX_LOG_STDERR** "`pmix.log.stderr`" (`char*`)
26 Log string to `stderr`.

27 **PMIX_LOG_STDOUT** "`pmix.log.stdout`" (`char*`)
28 Log string to `stdout`.

29 **PMIX_LOG_SYSLOG** "`pmix.log.syslog`" (`char*`)
30 Log data to syslog. Defaults to `ERROR` priority.

31 **PMIX_LOG_MSG** "`pmix.log.msg`" (`pmix_byte_object_t`)
32 Message blob to be sent somewhere.

33 **PMIX_LOG_EMAIL** "`pmix.log.email`" (`pmix_data_array_t`)
34 Log via email based on `pmix_info_t` containing directives.

35 **PMIX_LOG_EMAIL_ADDR** "`pmix.log.emaddr`" (`char*`)
36 Comma-delimited list of email addresses that are to receive the message.

37 **PMIX_LOG_EMAIL_SUBJECT** "`pmix.log.emsub`" (`char*`)
38 Subject line for email.

39 **PMIX_LOG_EMAIL_MSG** "`pmix.log.emmsg`" (`char*`)
40 Message to be included in email.

1 3.4.23 Debugger attributes

2 Attributes used to assist debuggers - these are values that can be passed to the `PMIx_Spawn` or
3 `PMIx_Init` APIs. Some may be accessed using the `PMIx_Get` API with the
4 `PMIX_RANK_WILDCARD` rank.

5 `PMIX_DEBUG_STOP_ON_EXEC` "pmix.dbg.exec" (bool)

6 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
7 debugger, and that the launcher is to pause the resulting application processes on first
8 instruction for debugger attach.

9 `PMIX_DEBUG_STOP_IN_INIT` "pmix.dbg.init" (bool)

10 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
11 debugger, and that the PMIx client library is to pause the resulting application processes
12 during `PMIx_Init` until debugger attach and release.

13 `PMIX_DEBUG_WAIT_FOR_NOTIFY` "pmix.dbg.notify" (bool)

14 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
15 debugger, and that the resulting application processes are to pause at some
16 application-determined location until debugger attach and release.

17 `PMIX_DEBUG_JOB` "pmix.dbg.job" (char*)

18 Namespace of the job to be debugged - provided to the debugger upon launch.

19 `PMIX_DEBUG_WAITING_FOR_NOTIFY` "pmix.dbg.waiting" (bool)

20 Job to be debugged is waiting for a release - this is not a value accessed using the
21 `PMIx_Get` API.

22 3.4.24 Resource manager attributes

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

27 `PMIX_RM_NAME` "pmix.rm.name" (char*)

28 String name of the RM.

29 `PMIX_RM_VERSION` "pmix.rm.version" (char*)

30 RM version string.

31 3.4.25 Environment variable attributes

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

34 `PMIX_SET_ENVAR` "pmix.set.envar" (char*)

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

36 `PMIX_UNSET_ENVAR` "pmix.unset.envar" (char*)

37 Unset the environment variable specified in the string.

1 3.4.26 Job Allocation attributes

2 Attributes used to describe the job allocation - these are values passed to the
3 `PMIx_Allocation_request_nb` API and are not accessed using the `PMIx_Get` API

4 **PMIX_ALLOC_ID** "pmix.alloc.id" (char*)
5 Provide a string identifier for this allocation request which can later be used to query status
6 of the request.

7 **PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)
8 The number of nodes.

9 **PMIX_ALLOC_NODE_LIST** "pmix.alloc.nlist" (char*)
10 Regular expression of the specific nodes.

11 **PMIX_ALLOC_NUM_CPUS** "pmix.alloc.ncpus" (uint64_t)
12 Number of cpus.

13 **PMIX_ALLOC_NUM_CPU_LIST** "pmix.alloc.ncpulist" (char*)
14 Regular expression of the number of cpus for each node.

15 **PMIX_ALLOC_CPU_LIST** "pmix.alloc.cpulist" (char*)
16 Regular expression of the specific cpus indicating the cpus involved.

17 **PMIX_ALLOC_MEM_SIZE** "pmix.alloc.msize" (float)
18 Number of Megabytes.

19 **PMIX_ALLOC_NETWORK** "pmix.alloc.net" (array)
20 Array of `pmix_info_t` describing requested network resources. If not given as part of an
21 `pmix_info_t` struct that identifies the involved nodes, then the description will be
22 applied across all nodes in the requestor's allocation.

23 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)
24 Name of the network.

25 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (float)
26 Mbits/sec.

27 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)
28 Quality of service level.

29 **PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)
30 Time in seconds.

31 3.4.27 Job control attributes

32 Attributes used to request control operations on an executing application - these are values passed
33 to the `PMIx_Job_control_nb` API and are not accessed using the `PMIx_Get` API.

34 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
35 Provide a string identifier for this request.

36 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
37 Pause the specified processes.

38 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)

1 Resume (“un-pause”) the specified processes.

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

3 Cancel the specified request (NULL implies cancel all requests from this requestor).

4 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)

5 Forcibly terminate the specified processes and cleanup.

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

7 Restart the specified processes using the given checkpoint ID.

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

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

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

11 Use event notification to trigger a process checkpoint.

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

13 Use the given signal to trigger a process checkpoint.

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

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

16 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

17 "pmix.jctrl.ckmethod" (pmix_data_array_t)

18 Array of `pmix_info_t` declaring each method and value supported by this application.

19 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)

20 Send given signal to specified processes.

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

22 Regular expression identifying nodes that are to be provisioned.

23 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvnmimg" (char*)

24 Name of the image that is to be provisioned.

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

26 Indicate that the job can be pre-empted.

27 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)

28 Politely terminate the specified processes.

29 3.4.28 Monitoring attributes

30 Attributes used to control monitoring of an executing application- these are values passed to the

31 **PMIx_Process_monitor_nb** API and are not accessed using the **PMIx_Get** API.

32 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)

33 Provide a string identifier for this request.

34 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)

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

36 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)

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

38 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)

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

40 **PMIX_SEND_HEARTBEAT** "pmix.monitor.beat" (void)


```

1         Send heartbeat to local PMIx server.
2     PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
3         Time in seconds before declaring heartbeat missed.
4     PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
5         Number of heartbeats that can be missed before generating the event.
6     PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
7         Register to monitor file for signs of life.
8     PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
9         Monitor size of given file is growing to determine if the application is running.
10    PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
11    Monitor time since last access of given file to determine if the application is running.
12    PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
13    Monitor time since last modified of given file to determine if the application is running.
14    PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
15    Time in seconds between checking the file.
16    PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
17    Number of file checks that can be missed before generating the event.

```

18 3.5 Callback Functions

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

22 3.5.1 Release Callback Function

23 Summary

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

27 Format

PMIx v1.0

```

28 typedef void (*pmix_release_cbfunc_t)
29             (void *cbdata)

```

30 INOUT cbdata

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

1 Description

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

4 3.5.2 Modex Callback Function

5 Summary

6 The `pmix_modex_cbfnc_t` is used by the `pmix_server_fencefn_t` and
7 `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex BCX data.

PMIx v1.0

```
▼ C ▼  
8 typedef void (*pmix_modex_cbfnc_t)  
9     (pmix_status_t status,  
10      const char *data, size_t ndata,  
11      void *cbdata,  
12      pmix_release_cbfnc_t release_fn,  
13      void *release_cbdata)  
▲ C ▲
```

14 **IN status**
15 Status associated with the operation (handle)
16 **IN data**
17 Data to be passed (pointer)
18 **IN ndata**
19 size of the data (`size_t`)
20 **IN cbdata**
21 Callback data passed to original API call (memory reference)
22 **IN release_fn**
23 Callback for releasing *data* (function pointer)
24 **IN release_cbdata**
25 Pointer to be passed to *release_fn* (memory reference)

26 Description

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

1 3.5.3 Spawn Callback Function

2 Summary

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

PMIx v1.0

```
5 typedef void (*pmix_spawn_cbfunc_t)  
6     (pmix_status_t status,  
7     pmix_namespace_t nspace, void *cbdata);
```

8 **IN status**

Status associated with the operation (handle)

10 **IN nspace**

Namespace string (`pmix_namespace_t`)

12 **IN cbdata**

Callback data passed to original API call (memory reference)

14 Description

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

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

21 3.5.4 Op Callback Function

22 Summary

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

PMIx v1.0

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

26 **IN status**

Status associated with the operation (handle)

28 **IN cbdata**

Callback data passed to original API call (memory reference)

Description

Used by a wide range of PMIx API's including [PMIx_Fence_nb](#), [pmix_server_client_connected_fn_t](#), [PMIx_server_register_namespace](#). This callback function is used to return a status to an often nonblocking operation.

3.5.5 Lookup Callback Function

Summary

The [pmix_lookup_cfunc_t](#) is used by [PMIx_Lookup_nb](#) to return data.

PMIx v1.0

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

- IN status**
Status associated with the operation (handle)
- IN data**
Array of data returned ([pmix_pdata_t](#))
- IN ndata**
Number of elements in the *data* array ([size_t](#))
- IN cbdata**
Callback data passed to original API call (memory reference)

Description

A callback function for calls to [PMIx_Lookup_nb](#). The function will be called upon completion of the 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.

1 3.5.6 Value Callback Function

2 Summary

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

PMIx v1.0

```
▼ C ▼  
4 typedef void (*pmix_value_cbfunc_t)  
5     (pmix_status_t status,  
6      pmix_value_t *kv, void *cbdata);  
▲ C ▲
```

7 **IN status**

8 Status associated with the operation (handle)

9 **IN kv**

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

11 **IN cbdata**

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

13 Description

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

17 3.5.7 Info Callback Function

18 Summary

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

PMIx v2.0

```
▼ C ▼  
20 typedef void (*pmix_info_cbfunc_t)  
21     (pmix_status_t status,  
22      pmix_info_t info[], size_t ninfo,  
23      void *cbdata,  
24      pmix_release_cbfunc_t release_fn,  
25      void *release_cbdata);
```

C

```
1  IN  status
2      Status associated with the operation ( pmix_status_t )
3  IN  info
4      Array of pmix_info_t returned by the operation (pointer)
5  IN  ninfo
6      Number of elements in the info array (size_t)
7  IN  cbdata
8      Callback data passed to original API call (memory reference)
9  IN  release_fn
10     Function to be called when done with the info data (function pointer)
11 IN  release_cbdata
12     Callback data to be passed to release_fn (memory reference)
```

Description

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

3.5.8 Event Handler Registration Callback Function

```
17 The pmix_evhdlr_reg_cbfunc_t callback function.
```

Advice to users

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

PMIx v2.0

C

```
22 typedef void (*pmix_evhdlr_reg_cbfunc_t)
23     (pmix_status_t status,
24      size_t evhdlr_ref,
25      void *cbdata)
```

C

```
26 IN  status
27     Status indicates if the request was successful or not ( pmix_status_t )
28 IN  evhdlr_ref
29     Reference assigned to the event handler by PMIx — this reference * must be used to
30     deregister the err handler (size_t)
31 IN  cbdata
32     Callback data passed to original API call (memory reference)
```

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_cbfnc_fn_t](#) is called by event handlers to indicate completion of their operations.

PMIx v2.0

C

```
typedef void (*pmix_event_notification_cbfnc_fn_t)
    (pmix_status_t status,
     pmix_info_t *results, size_t nresults,
     pmix_op_cbfnc_t cbfunc, void *thiscbdata,
     void *notification_cbdata);
```

C

IN status

Status returned by the event handler's operation ([pmix_status_t](#))

IN results

Results from this event handler's operation on the event ([pmix_info_t](#))

IN nresults

Number of elements in the results array ([size_t](#))

IN cbfunc

[pmix_op_cbfnc_t](#) function to be executed when PMIx completes processing the callback (function reference)

IN thiscbdata

Callback data that was passed in to the handler (memory reference)

IN cbdata

Callback data to be returned when PMIx executes cbfunc (memory reference)

Description

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

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

1 3.5.10 Notification Function

2 Summary

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

Advice to users

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

PMIx v2.0

```
7 typedef void (*pmix_notification_fn_t)
8     (size_t evhdlr_registration_id,
9      pmix_status_t status,
10     const pmix_proc_t *source,
11     pmix_info_t info[], size_t ninfo,
12     pmix_info_t results[], size_t nresults,
13     pmix_event_notification_cbfunc_fn_t cbfunc,
14     void *cbdata);
```

15 **IN evhdlr_registration_id**
16 Registration number of the handler being called (`size_t`)

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

19 **IN source**
20 Identifier of the process that generated the event (`pmix_proc_t`). If the source is the
21 SMS, then the namespace will be empty and the rank will be `PMIX_RANK_UNDEF`

22 **IN info**
23 Information describing the event (`pmix_info_t`). This argument will be `NULL` if no
24 additional information was provided by the event generator.

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

27 **IN results**
28 Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This
29 argument will be `NULL` if this is the first handler servicing the event, or if no prior handlers
30 provided results.

31 **IN nresults**
32 Number of elements in the results array (`size_t`)

33 **IN cbfunc**
34 `pmix_event_notification_cbfunc_fn_t` callback function to be executed upon
35 completion of the handler's operation and prior to handler return (function reference).

IN `cbdata`

Callback data to be passed to `cbfunc` (memory reference)

Description

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

Advice to users

Possible uses of the *info* array include:

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

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

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

Advice to PMIx server hosts

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

3.5.11 Server Setup Application Callback Function

The `PMIx_server_setup_application` callback function.

Summary

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

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

```

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

```

- 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 Client Connection Callback Function**

20 **Summary**

21 Callback function for incoming connection request from a local client

22 **Format**

PMIx v1.0

```

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

```

- 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 Callback Function**

5 **Summary**

6 Callback function for incoming tool connections.

7 **Format**

8 *PMIx v2.0*

```
▼ C ▼  
typedef void (*pmix_tool_connection_cbfunc_t) (  
    pmix_status_t status,  
    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**

PMIx v1.0 ▼ _____ C _____ ▼
8 **const char* PMIx_Get_version(void)**
 ▲ _____ C _____ ▲

9 **Description**

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

12 **4.2 Client Initialization and Finalization**

13 Initialization and finalization routines for PMIx clients.

▼ _____ **Advice to users** _____ ▼

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

▲ _____ ▲

17 **4.2.1 PMIx_Init**

18 **Summary**

19 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 libevent¹ **event_base** to use in place of the internal progress thread.

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

▲-----▲

19 Description

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

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

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

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

¹<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  

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

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. For example,
19 `PMIx_Finalize` does not include an internal barrier operation by default. This attribute
20 would direct `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

```
PMIx v2.0  ▼────────────────────────── C ───────────────────────────▶  
7 pmix_status_t  
8 PMIx_tool_init(pmix_proc_t *proc,  
9               pmix_info_t info[], size_t ninfo)  
          ▲────────────────────────── C ───────────────────────────▶
```

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 libevent² **event_base** to use in place of the internal progress thread.

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

▲-----▲

13 Description

14 Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided
 15 **pmix_proc_t** struct. The *info* array is used to pass user requests pertaining to the 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

²<http://libevent.org/>

- 1 • if `PMIX_CONNECT_TO_SYSTEM` is given, then the tool will search for a system-level
2 rendezvous file created by a PMIx server in the directory specified by the
3 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
4 connect to it. An error is returned if the rendezvous file cannot be found or the connection is
5 refused.
 - 6 • if `PMIX_CONNECT_SYSTEM_FIRST` is given, then the tool will search for a system-level
7 rendezvous file created by a PMIx server in the directory specified by the
8 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
9 connect to it. In this case, no error will be returned if the rendezvous file is not found or
10 connection is refused — the PMIx library will silently continue to the next option
 - 11 • by default, the tool will search the directory tree under the directory provided by the
12 `PMIX_SERVER_TMPDIR` environmental variable for rendezvous files of PMIx servers,
13 attempting to connect to each it finds until one accepts the connection. If no rendezvous files are
14 found, or all contacted servers refuse connection, then the PMIx library will return an error.
- 15 If successful, the function will return `PMIX_SUCCESS` and will fill the provided structure (if
16 provided) with the server-assigned namespace and rank of the tool. Note that each connection
17 attempt in the above precedence chain will retry (with delay between each retry) a number of times
18 according to the values of the corresponding attributes. Default is no retries.
- 19 Note that the PMIx tool library is referenced counted, and so multiple calls to `PMIx_tool_init`
20 are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call
21 `PMIx_tool_init` with a non-NULL parameter.

22 4.3.2 `PMIx_tool_finalize`

23 Summary

24 Finalize the PMIx library for a tool connection.

25 Format

PMIx v2.0

26 `pmix_status_t`
27 `PMIx_tool_finalize(void)`

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

29 Description

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

1 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 libevent³ 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

24 *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.

³<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 Keys are required to be unique within a specific level of information as defined in 3.4.10. For
8 example, a value for `PMIX_NUM_NODES` can be specified for each of the `session`, `job`, and
9 `application` levels. However, subsequently specifying another value for that attribute in the
10 `session` level will overwrite the prior value.

5.1 Setting and Accessing Key/Value Pairs

5.1.1 PMIx_Put

Summary

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

Format

15 *PMIx v1.0*

```
16 pmix_status_t  
17 PMIx_Put (pmix_scope_t scope,  
18           const pmix_key_t key,  
19           pmix_value_t *val)
```

20 **IN** `scope`
21 Distribution scope of the provided value (handle)

22 **IN** `key`
23 key (`pmix_key_t`)

24 **IN** `value`
25 Reference to a `pmix_value_t` structure (handle)

26 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 29. 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**

7

process reference (handle)

8

IN **key**

9

key to retrieve (**pmix_key_t**)

10

IN **info**

11

Array of info structures (array of handles)

12

IN **ninfo**

13

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

14

OUT **val**

15

value (handle)

16

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

Required Attributes

17

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

18

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

19

Look only in the client's local data store for the requested value - do not request data from 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 data cannot be found - do not request it from the host RM.

23

24

PMIX_DATA_SCOPE "pmix.scope" (**pmix_scope_t**)

25

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

26

PMIX_SESSION_INFO "pmix.ssn.info" (**bool**)

27

Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a

28

PMIX_SESSION_ID attribute identifying the desired target.

29

30

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

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

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

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

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

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



▼----- Optional Attributes -----▼

17 The following attributes are optional for host environments:

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

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



▼----- Advice to PMIx library implementers -----▼

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



Description

Retrieve information for the specified *key* as published by the process identified in the given `pmix_proc_t`, returning a pointer to the value in the given address.

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

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

Advice to users

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

Data posted by a process via `PMIx_Put` needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of `PMIX_RANK_WILDCARD` when the information being retrieved refers to something non-rank specific (e.g., number of processes on a node, number of processes in a job), and using the rank of the relevant process when requesting information that is rank-specific (e.g., the URI of the process, or the node upon which it is executing). Each subsection of Section 3.4 indicates the appropriate rank value for referencing the defined attribute.

5.1.3 `PMIx_Get_nb`

Summary

Nonblocking `PMIx_Get` operation.

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

Format

PMIx v1.0

C

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

C

- IN proc**
process reference (handle)
- IN key**
key to retrieve (string)
- IN info**
Array of info structures (array of handles)
- IN ninfo**
Number of elements in the *info* array (integer)
- IN cbfunc**
Callback function (function reference)
- IN cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

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

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

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

Required Attributes

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

PMIX_OPTIONAL "pmix.optional" (bool)

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

3 **PMIX_IMMEDIATE** "pmix.immediate" (bool)

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

6 **PMIX_DATA_SCOPE** "pmix.scope" (pmix_scope_t)

7 Scope of the data to be found in a **PMIx_Get** call.

8 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)

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

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

13 Return information about the specified job or namespace. If information about a job or
14 namespace other than the one containing the requesting process is desired, then the attribute
15 array must contain a **PMIX_JOBID** or **PMIX_NAMESPACE** attribute identifying the desired
16 target. Similarly, if information is requested about a job or namespace in a session other than
17 the one containing the requesting process, then an attribute identifying the target session
18 must be provided.

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

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

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

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

▲-----▲
▼-----▼ **Optional Attributes** -----▼

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

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

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

▲-----▲

Advice to PMIx library implementers

We recommend that implementation of the `PMIX_TIMEOUT` attribute be left to the host environment due to race condition considerations between 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`. See 3.4.10 for an explanation regarding use of the *level* attributes.

5.1.4 `PMIx_Store_internal`

Summary

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

1

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`

7

process reference (handle)

8

IN `key`

9

key to retrieve (string)

10

IN `val`

11

Value to store (handle)

12

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

13

Description

14

Store some data locally for retrieval by other areas of the proc. This is data that has only internal

15

scope - it will never be “pushed” externally.

16

5.1.5 Accessing information: examples

17

This section provides examples illustrating methods for accessing information at various levels.

18

The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate

19

how `PMIx_Get` can be used to obtain information on a `session`, `job`, `application`,

20

process, and node.

21

5.1.5.1 Session-level information

22

The `PMIx_Get` API does not include an argument for specifying the `session` associated with

23

the information being requested. Information regarding the session containing the requestor can be

24

obtained by the following methods:

25

- for session-level attributes (e.g., `PMIX_UNIV_SIZE`), specifying the requestor’s namespace and a rank of `PMIX_RANK_WILDCARD`; or

26

27

- for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_SESSION_INFO` attribute to indicate that the session-level information for that attribute is being requested

28

29

Example requests are shown below:

C

```

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

```

C

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

C

```

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

```

C

1 5.1.5.2 Job-level information

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

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

8 Example requests are shown below:

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

24 5.1.5.3 Application-level information

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

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

- for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_APP_INFO` attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

```

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

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

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

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

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

PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);

/* alternatively, we can directly ask for the #nodes in
 * the second application in our job, again remembering that
 * application numbers start with zero */
appnum = 1;
PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
C
-----
```

1 5.1.5.4 Process-level information

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

5 5.1.5.5 Node-level information

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

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

14 Example requests are shown below:

```
15 pmix_info_t info[2];  
16 pmix_value_t *value;  
17 pmix_status_t rc;  
18 pmix_proc_t myproc, otherproc;  
19 uint32_t nodeid;  
20  
21 /* initialize the client library */  
22 PMIx_Init(&myproc, NULL, 0);  
23  
24 /* get the #procs on our node */  
25 rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);  
26  
27 /* get the #slots on another node */  
28 PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);  
29 PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);  
30 rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);  
31
```

Advice to users

32 An explanation of the use of [PMIx_Get](#) versus [PMIx_Query_info_nb](#) is provided in [7.1.3.1](#).

1 5.2 Exchanging Key/Value Pairs

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



4 5.2.1 PMIx_Commit

5 Summary

6 Push all previously **PMIx_Put** values to the local PMIx server.

7 Format

8 *PMIx v1.0*

▼  ▼
pmix_status_t PMIx_Commit(void)
▲  ▲

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

10 Description

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

▼ **Advice to users** ▼

13 The local PMIx server will cache the information locally - i.e., the committed data will not be
14 circulated during **PMIx_Commit**. Availability of the data upon completion of **PMIx_Commit** is
15 therefore implementation-dependent.

16 5.2.2 PMIx_Fence

17 Summary

18 Execute a blocking barrier across the processes identified in the specified array, collecting
19 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

Format

PMIx v1.0

```

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

```

- IN procs**
Array of `pmix_proc_t` structures (array of handles)
- IN nprocs**
Number of 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 does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

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

Advice to PMIx library implementers

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 Passing a `NULL` pointer as the `procs` parameter indicates that the fence is to span all processes in
8 the client's namespace. Each provided `pmix_proc_t` struct can pass `PMIX_RANK_WILDCARD`
9 to indicate that all processes in the given namespace are participating.
10

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

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

5.2.3 `PMIx_Fence_nb`

Summary

20 Execute a nonblocking `PMIx_Fence` across the processes identified in the specified array of
21 processes, collecting information posted via `PMIx_Put` as directed.
22

1

Format

PMIx v1.0

C

2

pmix_status_t

3

PMIx_Fence_nb(const **pmix_proc_t** procs[], **size_t** nprocs,

4

const **pmix_info_t** info[], **size_t** ninfo,

5

pmix_op_cbfunc_t cbfunc, void *cbdata)

C

6

IN procs

7

Array of **pmix_proc_t** structures (array of handles)

8

IN nprocs

9

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

10

IN info

11

Array of info structures (array of handles)

12

IN ninfo

13

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

14

IN cbfunc

15

Callback function (function reference)

16

IN cbdata

17

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

18

Returns one of the following:

19

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

21

22

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

23

24

25

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

26

Required Attributes

27

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

28

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

29

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

Description

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

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

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

1 5.3 Publish and Lookup Data

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

▼ **Advice to PMIx library implementers** ▼

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



▼ **Advice to PMIx server hosts** ▼

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



7 5.3.1 PMIx_Publish

8 **Summary**

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

1 **Format**

PMIx v1.0

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

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

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

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

9 **Required Attributes**

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

13 **Optional Attributes**

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

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

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**.

22 **Advice to PMIx library implementers**

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

Description

Publish the data in the *info* array for subsequent lookup. By default, the data will be published into the `PMIX_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

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 one of the following:

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

Required Attributes

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

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

Format

PMIx v1.0

```

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

```

INOUT data

Array of publishable data structures (array of handles)

IN ndata

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

IN info

Array of info structures (array of handles)

IN ninfo

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

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

Required Attributes

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

PMIX_WAIT "pmix.wait" (int)

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

Advice to PMIx library implementers

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

Description

Lookup information published by this or another process. By default, the search will be conducted across the `PMIX_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 one of the following:

17

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

18

19

20

- a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called

Required Attributes

21

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.

22

23

24

Optional Attributes

25

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

26

PMIX_TIMEOUT "pmix.timeout" (int)

27

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

28

29

30

PMIX_RANGE "pmix.range" (pmix_data_range_t)

31

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

32

PMIX_WAIT "pmix.wait" (int)

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

▲-----▲
▼-----▼ **Advice to PMIx library implementers** -----▼

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

9 **Description**

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

15 **5.3.5 PMIx_Unpublish**

16 **Summary**

17 Unpublish data posted by this process using the given keys.

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_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**.

Format

PMIx v1.0

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

IN keys
(array of strings)

IN info
Array of info structures (array of handles)

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

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

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

Returns one of the following:

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

Required Attributes

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

Advice to PMIx library implementers

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

Description

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

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
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 v1.0

```

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

```

- IN** **job_info**
Array of info structures (array of handles)
- IN** **ninfo**
Number of elements in the *job_info* array (integer)
- IN** **apps**
Array of **pmix_app_t** structures (array of handles)
- IN** **napps**
Number of elements in the *apps* array (integer)
- OUT** **nspace**
Namespace of the new job (string)

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

Required Attributes

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

- PMIX_SPAWNED** "pmix.spawned" (**bool**)
true if this process resulted from a call to **PMIx_Spawn**.
- PMIX_PARENT_ID** "pmix.parent" (**pmix_proc_t**)
Process identifier of the parent process of the calling process.
- PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (**bool**)
The requesting process is a PMIx client.
- PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (**bool**)
The requesting process is a PMIx tool.

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

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

1 Working directory for spawned processes.

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

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

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

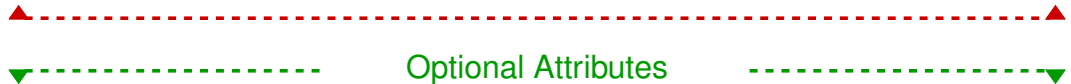
7 Prefix to use for starting spawned processes.

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

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

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

11 Hostfile to use for spawned processes.



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

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

14 Hostfile listing hosts to add to existing allocation.

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

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

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

18 Preload binaries onto nodes.

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

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

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

22 Name of personality to use.

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

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

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

28 Display process mapping upon spawn.

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

30 Number of processes to spawn on each identified resource.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 `PMIX_NO_PROCS_ON_HEAD` "pmix.nolocal" (bool)
5 Do not place processes on the head node.

6 `PMIX_NO_OVERSUBSCRIBE` "pmix.noover" (bool)
7 Do not oversubscribe the cpus.

8 `PMIX_REPORT_BINDINGS` "pmix.repbind" (bool)
9 Report bindings of the individual processes.

10 `PMIX_CPU_LIST` "pmix.cpulist" (char*)
11 List of cpus to use for this job - when accessed using `PMIx_Get` , use the
12 `PMIX_RANK_WILDCARD` value for the rank to discover the cpu list used for the provided
13 namespace

14 `PMIX_JOB_RECOVERABLE` "pmix.recover" (bool)
15 Application supports recoverable operations.

16 `PMIX_JOB_CONTINUOUS` "pmix.continuous" (bool)
17 Application is continuous, all failed processes should be immediately restarted.

18 `PMIX_MAX_RESTARTS` "pmix.maxrestarts" (uint32_t)
19 Maximum number of times to restart a job - when accessed using `PMIx_Get` , use the
20 `PMIX_RANK_WILDCARD` value for the rank to discover the max restarts for the provided
21 namespace



22 Description

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

26 By default, the spawned processes will be PMIx "connected" to the parent process upon successful
27 launch (see `PMIx_Connect` description for details). Note that this only means that (a) the parent
28 process will be given a copy of the new job's information so it can query job-level info without
29 incurring any communication penalties, (b) newly spawned child processes will receive a copy of
30 the parent processes job-level info, and (c) both the parent process and members of the child job
31 will receive notification of errors from processes in their combined assemblage.

▼ Advice to users ▼

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

1 6.2.2 PMIx_Spawn_nb

2 Summary

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

4 Format

PMIx v1.0

C

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

C

- 9 **IN** `job_info`
10 Array of info structures (array of handles)
- 11 **IN** `ninfo`
12 Number of elements in the `job_info` array (integer)
- 13 **IN** `apps`
14 Array of [pmix_app_t](#) structures (array of handles)
- 15 **IN** `cbfunc`
16 Callback function [pmix_spawn_cbfunc_t](#) (function reference)
- 17 **IN** `cbdata`
18 Data to be passed to the callback function (memory reference)

19 Returns one of the following:

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

Required Attributes

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

27 **PMIX_SPAWNED** "pmix.spawned" (bool)
28 true if this process resulted from a call to [PMIx_Spawn](#).

29 **PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)

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

2 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)

3 The requesting process is a PMIx client.

4 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)

5 The requesting process is a PMIx tool.

6

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

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

13 Working directory for spawned processes.

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

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

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

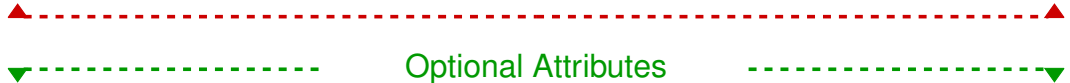
19 Prefix to use for starting spawned processes.

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

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

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

23 Hostfile to use for spawned processes.



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

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

26 Hostfile listing hosts to add to existing allocation.

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

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

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

30 Preload binaries onto nodes.

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

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

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

1 Name of personality to use.

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

3 Mapping mechanism to use for placing spawned processes - when accessed using

4 **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping

5 mechanism used for the provided namespace.

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

7 Display process mapping upon spawn.

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

9 Number of processes to spawn on each identified resource.

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

11 Process mapping policy - when accessed using **PMIx_Get** , use the

12 **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the

13 provided namespace

14 **PMIX_RANKBY** "pmix.rankby" (char*)

15 Process ranking policy - when accessed using **PMIx_Get** , use the

16 **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the

17 provided namespace

18 **PMIX_BINDTO** "pmix.bindto" (char*)

19 Process binding policy - when accessed using **PMIx_Get** , use the

20 **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the

21 provided namespace

22 **PMIX_NON_PMI** "pmix.nonpmi" (bool)

23 Spawned processes will not call **PMIx_Init** .

24 **PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)

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

26 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)

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

28 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)

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

30 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)

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

32 **PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)

33 Spawned application consists of debugger daemons.

34 **PMIX_TAG_OUTPUT** "pmix.tagout" (bool)

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

36 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)

1 Timestamp output from applications.

2 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)

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

4 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)

5 Output application output to the specified file.

6 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

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

8 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

9 Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the
10 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
11 provided namespace

12 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)

13 Do not place processes on the head node.

14 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)

15 Do not oversubscribe the cpus.

16 **PMIX_REPORT_BINDINGS** "pmix.repbind" (bool)

17 Report bindings of the individual processes.

18 **PMIX_CPU_LIST** "pmix.cpulist" (char*)

19 List of cpus to use for this job - when accessed using **PMIx_Get** , use the
20 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
21 namespace

22 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)

23 Application supports recoverable operations.

24 **PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)

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

26 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)

27 Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the
28 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
29 namespace



Description

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

Advice to users

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

6.3 Connecting and Disconnecting Processes

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies the following:

- job-level information for each namespace involved in the operation is to be made available to all processes in the connected assemblage
- any data posted by a process in the connected assemblage (via calls to `PMIx_Put` committed via `PMIx_Commit`) prior to execution of the `PMIx_Connect` operation is to be made accessible to all processes in the assemblage - any data posted after execution of the *connect* operation must be exchanged via a separate `PMIx_Fence` operation spanning the connected processes
- the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

Advice to PMIx server hosts

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

Advice to users

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

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

7 6.3.1 PMIx_Connect

8 Summary

9 Connect namespaces.

10 Format

PMIx v1.0

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

14 IN **procs**

15 Array of proc structures (array of handles)

16 IN **nprocs**

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

18 IN **info**

19 Array of info structures (array of handles)

20 IN **ninfo**

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

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

Required Attributes

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

Description

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

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

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

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

6.3.2 PMIx_Connect_nb

Summary

Nonblocking **PMIx_Connect_nb** routine.

1

Format

PMIx v1.0

C

2

pmix_status_t

3

PMIx_Connect_nb(const **pmix_proc_t** procs[], **size_t** nprocs,

4

const **pmix_info_t** info[], **size_t** ninfo,

5

pmix_op_cbfunc_t cbfunc, void *cbdata)

C

6

IN procs

7

Array of proc structures (array of handles)

8

IN nprocs

9

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

10

IN info

11

Array of info structures (array of handles)

12

IN ninfo

13

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

14

IN cbfunc

15

Callback function **pmix_op_cbfunc_t** (function reference)

16

IN cbdata

17

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

18

Returns one of the following:

19

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

21

22

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

23

24

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

25

Required Attributes

26

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.

27

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

Description

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

6.3.3 PMIx_Disconnect

Summary

Disconnect a previously connected set of processes.

1 **Format**

PMIx v1.0

```

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

```

- 5 **IN procs**
6 Array of proc structures (array of handles)
- 7 **IN nprocs**
8 Number of elements in the *procs* array (integer)
- 9 **IN info**
10 Array of info structures (array of handles)
- 11 **IN ninfo**
12 Number of element in the *info* array (integer)

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

Required Attributes

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

Optional Attributes

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

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

Advice to PMIx library implementers

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

Description

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

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

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

6.3.4 `PMIx_Disconnect_nb`

Summary

Nonblocking `PMIx_Disconnect` routine.

1

Format

PMIx v1.0

C

2

pmix_status_t

3

PMIx_Disconnect_nb(const **pmix_proc_t** procs[], **size_t** nprocs,

4

const **pmix_info_t** info[], **size_t** ninfo,

5

pmix_op_cbfunc_t cbfunc, void *cbdata);

C

6

IN procs

7

Array of proc structures (array of handles)

8

IN nprocs

9

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

10

IN info

11

Array of info structures (array of handles)

12

IN ninfo

13

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

14

IN cbfunc

15

Callback function **pmix_op_cbfunc_t** (function reference)

16

IN cbdata

17

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

18

Returns one of the following:

19

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

21

22

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

23

24

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

25

Required Attributes

26

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.

27

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

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 Nonblocking `PMIx_Disconnect` routine. The callback function is called once all processes
8 identified in *procs* have called either `PMIx_Disconnect_nb` or its blocking version, *and* the
9 host environment has completed any required supporting operations. See the advice provided in the
10 description for `PMIx_Disconnect` for more information.
11

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:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library *must not* invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will *not* be executed

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

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

▼----- Required Attributes -----▼

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

PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)

Retrieve updated information from server.

PMIX_SESSION_INFO "pmix.ssn.info" (bool)

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

PMIX_JOB_INFO "pmix.job.info" (bool)

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

PMIX_APP_INFO "pmix.app.info" (bool)

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

PMIX_NODE_INFO "pmix.node.info" (bool)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Optional Attributes

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

12 **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (char*)
13 Request a comma-delimited list of active namespaces.

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

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

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

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

23 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lptable" (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** for processes in job on same
26 node.

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

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

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

33 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
34 Report average values.

35 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)

1 Report minimum and maximum values.

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

3 String identifier of the allocation whose status is being requested.

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

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

6
7 **PMIX_SERVER_URI** "pmix.srvr.uri" (char*)

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

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

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



14 Description

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

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

21 The *status* argument to the callback function indicates if requested data was found or not. An array
22 of `pmix_info_t` will contain each key that was provided and the corresponding value that was
23 found. Requests for keys that are not found will return the key paired with a value of type
24 `PMIX_UNDEF`.

Advice to users

25 The desire to query a list of attributes supported by the implementation and/or the host environment
26 has been expressed and noted. The PMIx community is exploring the possibility and it will likely
27 become available in a future release

Advice to PMIx library implementers

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

1 7.1.3.1 Using `PMIx_Get` vs `PMIx_Query_info_nb`

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

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

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

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

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

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

29 7.2 Allocation Requests

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

- 32 • *Evolving* applications that dynamically request and return resources as they execute

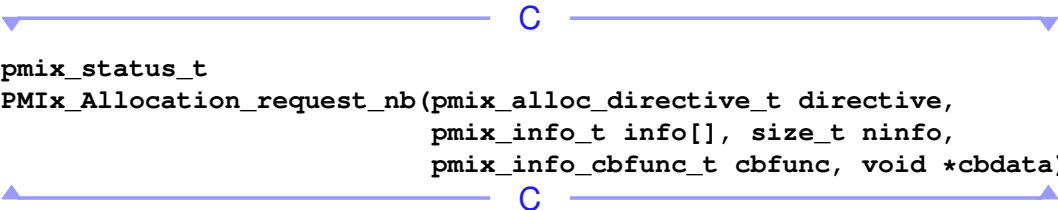
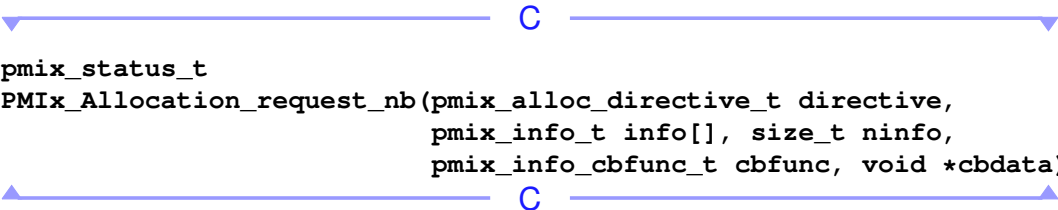
- 1 • *Malleable* environments where the scheduler redirects resources away from executing
- 2 applications for higher priority jobs or load balancing
- 3 • *Resilient* applications that need to request replacement resources in the face of failures
- 4 • *Rigid* jobs where the user has requested a static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

7 7.2.1 PMIx_Allocation_request_nb

8 Summary

9 Request an allocation operation from the host resource manager.

10 Format

11 *PMIx v2.0*  C 

```

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

```

- 15 **IN directive**
Allocation directive (handle)
- 16 **IN info**
Array of [pmix_info_t](#) structures (array of handles)
- 17 **IN ninfo**
Number of elements in the *info* array (integer)
- 18 **IN cbfunc**
Callback function [pmix_info_cbfunc_t](#) (function reference)
- 19 **IN cbdata**
Data to be passed to the callback function (memory reference)

20 Returns one of the following:

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

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)

1 Mbits/sec.
2 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)
3 Quality of service level.

4 **Description**

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

- 7 • Request allocation of additional resources, including memory, bandwidth, and compute. This
8 should be accomplished in a non-blocking manner so that the application can continue to
9 progress while waiting for resources to become available. Note that the new allocation will be
10 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one
11 allocation will not impact the other.
- 12 • Extend the reservation on currently allocated resources, subject to scheduling availability and
13 priorities. This includes extending the time limit on current resources, and/or requesting
14 additional resources be allocated to the requesting job. Any additional allocated resources will be
15 considered as part of the current allocation, and thus will be released at the same time.
- 16 • Return no-longer-required resources to the scheduler. This includes the “loan” of resources back
17 to the scheduler with a promise to return them upon subsequent request.

18 **7.2.2 PMIx_Job_control_nb**

19 The **PMIx_Job_control_nb** API enables the application and SMS to coordinate the response
20 to failures and other events. This can include requesting termination of the entire job or a subset of
21 processes within a job, but can also be used in combination with other PMIx capabilities (e.g.,
22 allocation support and event notification) for more nuanced responses. For example, an application
23 notified of an incipient over-temperature condition on a node could use the
24 **PMIx_Allocation_request_nb** interface to request replacement nodes while
25 simultaneously using the **PMIx_Job_control_nb** interface to direct that a checkpoint event be
26 delivered to all processes in the application. If replacement resources are not available, the
27 application might use the **PMIx_Job_control_nb** interface to request that the job continue at
28 a lower power setting, perhaps sufficient to avoid the over-temperature failure.

29 The job control API can also be used by an application to register itself as available for preemption
30 when operating in an environment such as a cloud or where incentives, financial or otherwise, are
31 provided to jobs willing to be preempted. Registration can include attributes indicating how many
32 resources are being offered for preemption (e.g., all or only some portion), whether the application
33 will require time to prepare for preemption, etc. Jobs that request a warning will receive an event
34 notifying them of an impending preemption (possibly including information as to the resources that
35 will be taken away, how much time the application will be given prior to being preempted, whether

1 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save
2 their work. Once the application is ready, it calls the provided event completion callback function to
3 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any
4 desired restart.

5 **Summary**

6 Request a job control action.

7 **Format**

PMIx v2.0

C

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

C

12 **IN targets**
13 Array of proc structures (array of handles)
14 **IN ntargets**
15 Number of element in the *targets* array (integer)
16 **IN directives**
17 Array of info structures (array of handles)
18 **IN ndirs**
19 Number of element in the *directives* array (integer)
20 **IN cbfunc**
21 Callback function **pmix_info_cbfunc_t** (function reference)
22 **IN cbdata**
23 Data to be passed to the callback function (memory reference)

24 Returns one of the following:

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

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)
Forcibly terminate the specified processes and cleanup.
- PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
Send given signal to specified processes.
- PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
Politely terminate the specified processes.

Optional Attributes

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

- PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
Cancel the specified request (**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)

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

2 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

3 "pmix.jctrl.ckmethod" (pmix_data_array_t)

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

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

6 Regular expression identifying nodes that are to be provisioned.

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

8 Name of the image that is to be provisioned.

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

10 Indicate that the job can be pre-empted.



11 Description

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

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

20 7.3 Process and Job Monitoring

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

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

1 7.3.1 PMIx_Process_monitor_nb

2 Summary

3 Request that application processes be monitored.

4 Format

PMIx v2.0

C

5 `pmix_status_t`

```
6 PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,  
7                         const pmix_info_t directives[], size_t ndirs,  
8                         pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

9 **IN monitor**

10 info (handle)

11 **IN error**

12 status (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_info_cbfunc_t` (function reference)

19 **IN cbdata**

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

21 Returns one of the following:

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

Optional Attributes

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

PMIX_MONITOR_ID "pmix.monitor.id" (char*)

Provide a string identifier for this request.

PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)

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

PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)

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

PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)

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

PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)

Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)

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

PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)

Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)

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

PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)

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

PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)

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

PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)

Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)

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

1 Description

2 Request that application processes be monitored via several possible methods. For example, that
3 the server monitor this process for periodic heartbeats as an indication that the process has not
4 become “wedged”. When a monitor detects the specified alarm condition, it will generate an event
5 notification using the provided error code and passing along any available relevant information. It
6 is up to the caller to register a corresponding event handler.

7 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,
8 [PMIX_MONITOR_FILE](#) to indicate that the requestor is asking that a file be monitored.

9 The *error* argument is the status code to be used when generating an event notification alerting that
10 the monitor has been triggered. The range of the notification defaults to
11 [PMIX_RANGE_NAMESPACE](#). This can be changed by providing a [PMIX_RANGE](#) directive.

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

14 The *cbfunc* function provides a *status* to indicate whether or not the request was granted, and to
15 provide some information as to the reason for any denial in the [pmix_info_cbfunc_t](#) array of
16 [pmix_info_t](#) structures.

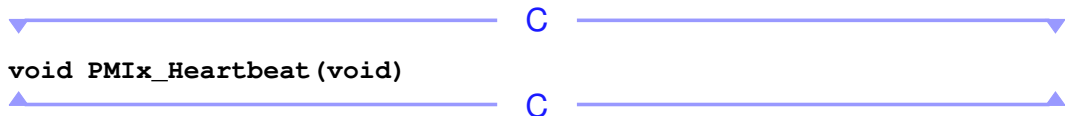
17 7.3.2 PMIx_Heartbeat

18 Summary

19 Send a heartbeat to the PMIx server library

20 Format

PMIx v2.0

21 

22 Description

23 A simplified macro wrapping [PMIx_Process_monitor_nb](#) that sends a heartbeat to the
24 PMIx server library.

25 7.4 Logging

26 The logging interface supports posting information by applications and SMS elements to persistent
27 storage. This function is *not* intended for output of computational results, but rather for reporting
28 status and saving state information such as inserting computation progress reports into the
29 application’s SMS job log or error reports to the local syslog.

Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

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

PMIX_LOG_STDERR "pmix.log.stderr" (char*)

Log string to **stderr**.

PMIX_LOG_STDOUT "pmix.log.stdout" (char*)

Log string to **stdout**.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)

Log data to syslog. Defaults to **ERROR** priority.

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.

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.evrangle" (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.

8 Host environments that implement support for PMIx event notification are required to support the
9 following attributes:

10 **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (`pmix_proc_t`)
11 The single process that was affected.
12 **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaffected" (`pmix_data_array_t*`)
13 Array of `pmix_proc_t` defining affected processes.

▲-----▲
▼-----▼ Optional Attributes ▼-----▼

14 Host environments that support PMIx event notification *may* offer notifications for environmental
15 events impacting the job and for SMS events relating to the job. The following attributes are
16 optional for host environments that support this operation:

17 **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (`bool`)
18 The RM intends to terminate this session.
19 **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (`bool`)
20 The RM intends to terminate this job.
21 **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (`bool`)
22 The RM intends to terminate all processes on this node.
23 **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (`bool`)
24 The RM intends to terminate just this process.
25 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evtimeout" (`int`)
26 The time in seconds before the RM will execute error response.
27 **PMIX_EVENT_SILENT_TERMINATION** "pmix.avsilentterm" (`bool`)
28 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.

The callback function *must not* be executed prior to returning from the API.

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 The callback function *must not* be executed prior to returning from the API.

20 **8.1.3 PMIx_Notify_event**

21 **Summary**

22 Report an event for notification via any registered event handler.

23 **Format**

PMIx v2.0

C

```

24 pmix_status_t
25 PMIx_Notify_event(pmix_status_t status,
26                  const pmix_proc_t *source,
27                  pmix_data_range_t range,
28                  pmix_info_t info[], size_t ninfo,
29                  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. The callback function *must not* be executed prior to returning from
 21 the API.

22 **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
 23 returned *success* - the *cbfunc* will *not* be called

24 **PMIX_ERR_BAD_PARAM** The request contains at least one incorrect entry that prevents it from
 25 being processed. The callback function will *not* be called.

26 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification,
 27 or in the case of a PMIx server calling the API, the range extended beyond the local node and
 28 the host SMS environment does not support event notification. The callback function will
 29 *not* be called.

Required Attributes

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

31 **PMIX_EVENT_NON_DEFAULT** "pmix.evnondef" (`bool`)
 32 Event is not to be delivered to default event handlers.

33 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrangle" (`pmix_data_array_t*`)
 34 Array of `pmix_proc_t` defining range of event notification.

Host environments that implement support for PMIx event notification are required to provide the following attributes for all events generated by the environment:

PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)

The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)

Array of pmix_proc_t defining affected processes.

Description

Report an event for notification via any registered event handler. This function can be called by any PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server calls this API to report events it detected itself so that the host SMS daemon distribute and handle them, and to pass events given to it by its host down to any attached client processes for processing. Examples might include notification of the failure of another process, detection of an impending node failure due to rising temperatures, or an intent to preempt the application. Events may be locally generated or come from anywhere in the system.

Host SMS daemons call the API to pass events down to its embedded PMIx server both for transmittal to local client processes and for the server's own internal processing.

Client application processes can call this function to notify the SMS and/or other application processes of an event it encountered. Note that processes are not constrained to report status values defined in the official PMIx standard — any integer value can be used. Thus, applications are free to define their own internal events and use the notification system for their own internal purposes.

Advice to users

The callback function will be called upon completion of the **notify_event** function's actions. At that time, any messages required for executing the operation (e.g., to send the notification to the local PMIx server) will have been queued, but may not yet have been transmitted. The caller is required to maintain the input data until the callback function has been executed — the sole purpose of the callback function is to indicate when the input data is no longer required.

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* ▼  
14 **PMIX_DATA_BUFFER_CREATE** (`buffer`);
14 ▲  

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 ▼ `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 ▼ `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 Returns one of the following:

- 19 `PMIX_SUCCESS` The data has been packed as requested
 20 `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.
 21 `PMIX_ERR_BAD_PARAM` The provided buffer or src is `NULL`
 22 `PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this
 23 implementation
 24 `PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation
 25 `PMIX_ERROR` General error

26 Description

27 The pack function packs one or more values of a specified type into the specified buffer. The buffer
 28 must have already been initialized via the `PMIX_DATA_BUFFER_CREATE` or
 29 `PMIX_DATA_BUFFER_CONSTRUCT` macros — otherwise, `PMIx_Data_pack` will return an
 30 error. Providing an unsupported type flag will likewise be reported as an error.

31 Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may
 32 lose precision when unpacked by a non-homogeneous recipient. The `PMIx_Data_pack` function
 33 will do its best to deal with heterogeneity issues between the packer and unpacker in such cases.
 34 Sending a number larger than can be handled by the recipient will return an error code (generated
 35 upon unpacking) — the error cannot be detected during packing.

36 The namespace of the intended recipient of the packed buffer (i.e., the process that will be
 37 unpacking it) is used solely to resolve any data type differences between PMIx versions. The
 38 recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx

1 library is aware of the version the recipient is using. Note that all processes in a given namespace
2 are *required* to use the same PMIx version — thus, the caller must only know at least one process
3 from the target’s namespace.

4 9.2.2 PMIx_Data_unpack

5 Summary

6 Unpack values from a `pmix_data_buffer_t`

7 Format

PMIx v2.0

C

```
8 pmix_status_t  
9 PMIx_Data_unpack(const pmix_proc_t *source,  
10 pmix_data_buffer_t *buffer, void *dest,  
11 int32_t *max_num_values,  
12 pmix_data_type_t type);  
13
```

C

14 IN source

15 Pointer to a `pmix_proc_t` structure containing the nspace/rank of the process that packed
16 the provided buffer. A NULL value may be used to indicate that the source is based on the
17 same PMIx version as the caller. Note that only the source’s nspace is relevant. (handle)

18 IN buffer

19 A pointer to the buffer from which the value will be extracted. (handle)

20 INOUT dest

21 A pointer to the memory location into which the data is to be stored. Note that these values
22 will be stored contiguously in memory. For strings, this pointer must be to (char**) to
23 provide a means of supporting multiple string operations. The unpack function will allocate
24 memory for each string in the array - the caller must only provide adequate memory for the
25 array of pointers. (void*)

26 INOUT max_num_values

27 The number of values to be unpacked — upon completion, the parameter will be set to the
28 actual number of values unpacked. In most cases, this should match the maximum number
29 provided in the parameters — but in no case will it exceed the value of this parameter. Note
30 that unpacking fewer values than are actually available will leave the buffer in an unpackable
31 state — the function will return an error code to warn of this condition. (int32_t)

32 IN type

33 The type of the data to be unpacked — must be one of the PMIx defined data types (
34 `pmix_data_type_t`)

1 Returns one of the following:

2 **PMIX_SUCCESS** The data has been unpacked as requested

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

4 **PMIX_ERR_BAD_PARAM** The provided buffer or dest is **NULL**

5 **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
6 implementation

7 **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation

8 **PMIX_ERROR** General error

9 **Description**

10 The unpack function unpacks the next value (or values) of a specified type from the given buffer.
11 The buffer must have already been initialized via an **PMIX_DATA_BUFFER_CREATE** or
12 **PMIX_DATA_BUFFER_CONSTRUCT** call (and assumedly filled with some data) — otherwise,
13 the unpack_value function will return an error. Providing an unsupported type flag will likewise be
14 reported as an error, as will specifying a data type that *does not* match the type of the next item in
15 the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an
16 error.

17 NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper
18 variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte
19 field in a string array that so happens to have a value that matches the specified data type flag).
20 Therefore, the data type error check is *not* completely safe.

21 Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer.
22 It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the
23 unpack_ptr.

24 Warning: The caller is responsible for providing adequate memory storage for the requested data.
25 The user must provide a parameter indicating the maximum number of values that can be unpacked
26 into the allocated memory. If more values exist in the buffer than can fit into the memory storage,
27 then the function will unpack what it can fit into that location and return an error code indicating
28 that the buffer was only partially unpacked.

29 Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may
30 lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with
31 heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than
32 can be handled by the recipient will return an error code generated upon unpacking — these errors
33 cannot be detected during packing.

34 The namespace of the process that packed the buffer is used solely to resolve any data type
35 differences between PMIx versions. The packer must, therefore, be known to the user prior to
36 calling the pack function so that the PMIx library is aware of the version the packer is using. Note
37 that all processes in a given namespace are *required* to use the same PMIx version — thus, the
38 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 Returns one of the following:

17 **PMIX_SUCCESS** The data has been copied as requested

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

19 **PMIX_ERR_BAD_PARAM** The provided src or dest is **NULL**

20 **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
21 implementation

22 **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation

23 **PMIX_ERROR** General error

24 Description

25 Since registered data types can be complex structures, the system needs some way to know how to
26 copy the data from one location to another (e.g., for storage in the registry). This function, which
27 can call other copy functions to build up complex data types, defines the method for making a copy
28 of the specified data type.

29 9.2.4 PMIx_Data_print

30 Summary

31 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);
```

C

5 **IN output**

6 The address of a pointer into which the address of the resulting output is to be stored.
7 (char**)

8 **IN prefix**

9 String to be prepended to the resulting output (char*)

10 **IN src**

11 A pointer to the memory location of the data value to be printed (handle)

12 **IN type**

13 The type of the data value to be printed — must be one of the PMIx defined data types. (
14 pmix_data_type_t)

15 Returns one of the following:

16 **PMIX_SUCCESS** The data has been printed as requested

17 **PMIX_ERR_BAD_PARAM** The provided data type is not recognized.

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

19 **Description**

20 Since registered data types can be complex structures, the system needs some way to know how to
21 print them (i.e., convert them to a string representation). Primarily for debug purposes.

22 **9.2.5 PMIx_Data_copy_payload**

23 **Summary**

24 Copy a payload from one buffer to another

25 **Format**

PMIx v2.0

C

```
26 pmix_status_t
27 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
28                       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 Returns one of the following:

6 **PMIX_SUCCESS** The data has been copied as requested
7 **PMIX_ERR_BAD_PARAM** The src and dest `pmix_data_buffer_t` types do not match
8 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

9 **Description**

10 This function will append a copy of the payload in one buffer into another buffer. Note that this is
11 *not* a destructive procedure — the source buffer’s payload will remain intact, as will any pre-existing
12 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)
```

IN input

String to process (string)

OUT regex

Regular expression representation of *input* (string)

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 order of the individual values in the *input* string is
4 preserved in the resulting *regex* string. The caller is responsible for free'ing the resulting string.

5 If values have leading zero's, then that is preserved, as are prefix and suffix strings. For example, an
6 input string of
7 “**odin009.org,odin010.org,odin011.org,odin012.org,odin[102-107].org**”
8 will return a regular expression of “**pmix:odin[009-012,102-107].org**”

Advice to users

9 The returned regular expression will have a “**pmix:**” at the beginning of the string. This informs
10 the PMIx parser that the string was produced using the PRI's regular expression generator, and thus
11 that same plugin should be used for parsing the string

12 **10.1.2 PMIx_generate_ppn**

13 **Summary**

14 Generate a regular expression representation of the input string.

15 **Format**

PMIx v1.0

```
▼ _____ C _____ ▼  
16   pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)  
▲ _____ C _____ ▲
```

17 **IN** **input**
18 String to process (string)

19 **OUT** **regex**
20 Regular expression representation of *input* (string)

21 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 one of the following:

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

Required Attributes

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

9 **PMIX_REGISTER_NODATA** "pmix.reg.nodata" (bool)
10 Registration is for this namespace only, do not copy job data - this attribute is not accessed
11 using the **PMIx_Get**

12 Host environments are *required* to provide the following attributes:

- 14 • for the session containing the given namespace:
 - 15 - **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)
16 Number of allocated slots in a session - each slot may or may not be occupied by an
17 executing process. Note that this attribute is the equivalent to the combination of
18 **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it
19 is included in the Standard for historical reasons.
- 20 • for the given namespace:
 - 21 - **PMIX_JOBID** "pmix.jobid" (char*)
22 Job identifier assigned by the scheduler.
 - 23 - **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)
24 Total number of processes in this job across all contained applications
 - 25 - **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
26 Maximum number of processes that can be executed in this context (session,
27 namespace, application, or node). Typically, this is a constraint imposed by a scheduler
28 or by user settings in a hostfile or other resource description.
 - 29 - **PMIX_NODE_MAP** "pmix.nmap" (char*)
30 Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.
 - 31 - **PMIX_PROC_MAP** "pmix.pmap" (char*)
32 Regular expression describing processes on each node - see 10.1.3.1 for an explanation
33 of its generation.
- 34 • for its own node:

- 1 - **PMIX_LOCAL_SIZE** "pmix.local.size" (uint32_t)
- 2 Number of processes in this job or application on this node.
- 3 - **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
- 4 Comma-delimited list of ranks on this node within the specified namespace - referenced
- 5 using **PMIX_RANK_WILDCARD** .
- 6 - **PMIX_LOCAL_CPUSSETS** "pmix.lcpus" (char*)
- 7 Colon-delimited cpusets of local peers within the specified namespace - referenced
- 8 using **PMIX_RANK_WILDCARD** .
- 9 • for each process in the given namespace:
 - 10 - **PMIX_RANK** "pmix.rank" (pmix_rank_t)
 - 11 Process rank within the job.
 - 12 - **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)
 - 13 Local rank on this node within this job.
 - 14 - **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
 - 15 Process rank on this node spanning all jobs.
 - 16 - **PMIX_NODEID** "pmix.nodeid" (uint32_t)
 - 17 Node identifier where the specified process is located, expressed as the node's index
 - 18 (beginning at zero) in the array resulting from expansion of the **PMIX_NODE_MAP**
 - 19 regular expression for the **job**

20 If more than one application is included in the namespace, then the host environment is also
 21 *required* to provide the following attributes:

- 22 • for each application:
 - 23 - **PMIX_APPNUM** "pmix.appnum" (uint32_t)
 - 24 Application number within the job.
 - 25 - **PMIX_APPLDR** "pmix.aldr" (pmix_rank_t)
 - 26 Lowest rank in this application within this job - referenced using
 - 27 **PMIX_RANK_WILDCARD** .
 - 28 - **PMIX_APP_SIZE** "pmix.app.size" (uint32_t)
 - 29 Number of processes in this application.
- 30 • for each process:
 - 31 - **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
 - 32 Process rank within this application.
 - 33 - **PMIX_APPNUM** "pmix.appnum" (uint32_t)
 - 34 Application number within the job.



Optional Attributes

The following attributes *may* be provided by host environments:

- for the session containing the given namespace:
 - **PMIX_SESSION_ID** "pmix.session.id" (uint32_t)
Session identifier - referenced using **PMIX_RANK_WILDCARD** .
- for the given namespace:
 - **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (char*)
Name of the namespace to use for this PMIx server.
 - **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
Rank of this PMIx server
 - **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .
 - **PMIX_ALLOCATED_NODELIST** "pmix.alist" (char*)
Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using **PMIX_RANK_WILDCARD** .
 - **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)
Number of applications in this job.
 - **PMIX_MAPBY** "pmix.mapby" (char*)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace
 - **PMIX_RANKBY** "pmix.rankby" (char*)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace
 - **PMIX_BINDTO** "pmix.bindto" (char*)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace
- for its own node:
 - **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
Total available physical memory on this node.
 - **PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)
XML representation of local topology using hwloc's v1.x format.
 - **PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)

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

2 - **PMIX_LOCALLDR** "pmix.lldr" (pmix_rank_t)
3 Lowest rank on this node within this job - referenced using **PMIX_RANK_WILDCARD** .
4

5 - **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)
6 Number of processes across all jobs on this node.

7 - **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)
8 Array of **pmix_proc_t** of all processes on the specified node - referenced using
9 **PMIX_RANK_WILDCARD** .

10 • for each process in the given namespace:

11 - **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
12 Process identifier

13 - **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
14 Process rank spanning across all jobs in this session.

15 - **PMIX_HOSTNAME** "pmix.hname" (char*)
16 Name of the host where the specified process is running.

17 Attributes not directly provided by the host environment *may* be derived by the PMIx server library
18 from other required information and included in the data made available to the server library's
19 clients.



20 Description

21 Pass job-related information to the PMIx server library for distribution to local client processes.



Advice to PMIx server hosts

22 Host environments are *required* to execute this operation prior to starting any local application
23 process within the given namespace.

24 The PMIx server must register *all* namespaces that will participate in collective operations with
25 local processes. This means that the server must register a namespace even if it will not host any
26 local processes from within that namespace *if* any local process of another namespace might at
27 some point perform an operation involving one or more processes from the new namespace. This is
28 necessary so that the collective operation can identify the participants and know when it is locally
29 complete.

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



Advice to users

1 The number of local processes for any given namespace is generally fixed at the time of application
2 launch. Calls to `PMIx_Spawn` result in processes launched in their own namespace, not that of
3 their parent. However, it is possible for processes to *migrate* to another node via a call to
4 `PMIx_Job_control_nb`, thus resulting in a change to the number of local processes on both
5 the initial node and the node to which the process moved. It is therefore *critical* that applications
6 not migrate processes without first ensuring that PMIx-based collective operations are not in
7 progress, and that no such operations be initiated until process migration has completed.

8 10.1.3.1 Assembling the registration information

9 The following description is not intended to represent the actual layout of information in a given
10 PMIx library. Instead, it describes how information provided in the *info* parameter of the
11 `PMIx_server_register_namespace` shall be organized for proper processing by a PMIx server
12 library. The ordering of the various information elements is arbitrary - they are presented in a
13 top-down hierarchical form solely for clarity in reading.

Advice to PMIx server hosts

14 Creating the *info* array of data requires knowing in advance the number of elements required for the
15 array. This can be difficult to compute and somewhat fragile in practice. One method for resolving
16 the problem is to create a linked list of objects, each containing a single `pmix_info_t` structure.
17 Allocation and manipulation of the list can then be accomplished using existing standard methods.
18 Upon completion, the final *info* array can be allocated based on the number of elements on the list,
19 and then the values in the list object `pmix_info_t` structures transferred to the corresponding
20 array element utilizing the `PMIX_INFO_XFER` macro.

21 A common building block used in several areas is the construction of a regular expression
22 identifying the nodes involved in that area - e.g., the nodes in a `session` or `job`. PMIx provides
23 several tools to facilitate this operation, beginning by constructing an argv-like array of node
24 names. This array is then passed to the `PMIx_generate_regex` function to create a regular
25 expression parseable by the PMIx server library, as shown below:

```

1  char **nodes = NULL;
2  char *nodelist;
3  char *regex;
4  size_t n;
5  pmix_status_t rc;
6  pmix_info_t info;
7
8  /* loop over an array of nodes, adding each
9   * name to the array */
10 for (n=0; n < num_nodes; n++)
11     /* filter the nodes to ignore those not included
12      * in the target range (session, job, etc.). In
13      * this example, all nodes are accepted */
14     PMIX_ARGV_APPEND(&nodes, node[n]->name);
15
16
17 /* join into a comma-delimited string */
18 nodelist = PMIX_ARGV_JOIN(nodes, ',');
19
20 /* release the array */
21 PMIX_ARGV_FREE(nodes);
22
23 /* generate regex */
24 rc = PMIX_generate_regex(nodelist, &regex);
25
26 /* release list */
27 free(nodelist);
28
29 /* pass the regex as the value to the PMIX_NODE_MAP key */
30 PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
31 /* release the regex */
32 free(regex);
33

```

34 Changing the filter criteria allows the construction of node maps for any level of information.

35 A similar method is used to construct the map of processes on each node from the namespace being
36 registered. This may be done for each information level of interest (e.g., to identify the process map
37 for the entire **job** or for each **application** in the job) by changing the search criteria. An
38 example is shown below for the case of creating the process map for a **job** :

```

1  char **ndppn;
2  char rank[30];
3  char **ppnarray = NULL;
4  char *ppn;
5  char *localranks;
6  char *regex;
7  size_t n, m;
8  pmix_status_t rc;
9  pmix_info_t info;
10
11 /* loop over an array of nodes */
12 for (n=0; n < num_nodes; n++)
13     /* for each node, construct an array of ranks on that node */
14     ndppn = NULL;
15     for (m=0; m < node[n]->num_procs; m++)
16         /* ignore processes that are not part of the target job */
17         if (!PMIX_CHECK_NAMESPACE(targetjob,node[n]->proc[m].nspace))
18             continue;
19
20         snprintf(rank, 30, "%d", node[n]->proc[m].rank);
21         PMIX_ARGV_APPEND(&ndppn, rank);
22
23     /* convert the array into a comma-delimited string of ranks */
24     localranks = PMIX_ARGV_JOIN(ndppn, ',');
25     /* release the local array */
26     PMIX_ARGV_FREE(ndppn);
27     /* add this node's contribution to the overall array */
28     PMIX_ARGV_APPEND(&ppnarray, localranks);
29     /* release the local list */
30     free(localranks);
31
32
33 /* join into a semicolon-delimited string */
34 ppn = PMIX_ARGV_JOIN(ppnarray, ';');
35
36 /* release the array */
37 PMIX_ARGV_FREE(ppnarray);
38
39 /* generate ppn regex */
40 rc = PMIx_generate_ppn(ppn, &regex);
41
42 /* release list */

```

```

1  free(ppn);
2
3  /* pass the regex as the value to the PMIX_PROC_MAP key */
4  PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
5  /* release the regex */
6  free(regex);
7

```



8 Note that the **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes are linked in that the order of
9 entries in the process map must match the ordering of nodes in the node map - i.e., there is no
10 provision in the PMIx process map regular expression generator/parser pair supporting an
11 out-of-order node or a node that has no corresponding process map entry (e.g., a node with no
12 processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

- 13 • Session-level information includes all session-specific values. In many cases, only two values (
14 **PMIX_SESSION_ID** and **PMIX_UNIV_SIZE**) are included in the registration array. Since
15 both of these values are session-specific, they can be specified independently - i.e., in their own
16 **pmix_info_t** elements of the *info* array. Alternatively, they can be provided as a
17 **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_SESSION_INFO_ARRAY**
18 attribute and identified by including the **PMIX_SESSION_ID** attribute in the array - this is
19 *required* in cases where non-specific attributes (e.g., **PMIX_NUM_NODES** or
20 **PMIX_NODE_MAP**) are passed to describe aspects of the session. Note that the node map can
21 include nodes not used by the job being registered as no corresponding process map is specified.

22 The *info* array at this point might look like (where the labels identify the corresponding attribute
23 - e.g., “Session ID” corresponds to the **PMIX_SESSION_ID** attribute):

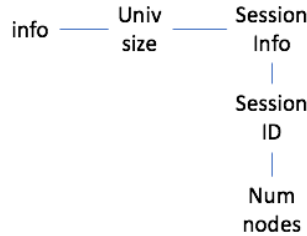


Figure 10.1.: Session-level information elements

- 24 • Job-level information includes all job-specific values such as **PMIX_JOB_SIZE** ,
25 **PMIX_JOB_NUM_APPS** , and **PMIX_JOBID** . Since each invocation of
26 **PMIx_server_register_nspace** describes a single **job** , job-specific values can be
27 specified independently - i.e., in their own **pmix_info_t** elements of the *info* array.
28 Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t**
29 identified by the **PMIX_JOB_INFO_ARRAY** attribute - this is *required* in cases where

1 non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the job. Note
 2 that since the invocation only involves a single namespace, there is no need to include the
 3 **PMIX_NAMESPACE** attribute in the array.
 4 Upon conclusion of this step, the *info* array might look like:

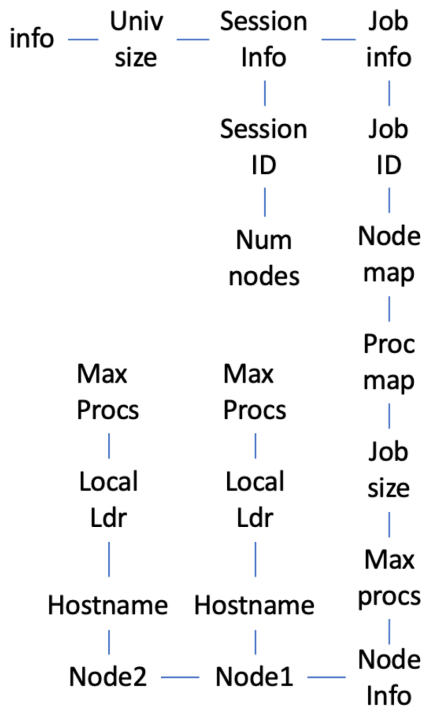


Figure 10.2.: Job-level information elements

5 Note that in this example, **PMIX_NUM_NODES** is not required as that information is contained
 6 in the **PMIX_NODE_MAP** attribute. Similarly, **PMIX_JOB_SIZE** is not technically required as
 7 that information is contained in the **PMIX_PROC_MAP** when combined with the corresponding
 8 node map - however, there is no issue with including the job size as a separate entry.

9 The example also illustrates the hierarchical use of the **PMIX_NODE_INFO_ARRAY** attribute.
 10 In this case, we have chosen to pass several job-related values for each node - since those values
 11 are non-unique across the job, they must be passed in a node-info container. Note that the choice
 12 of what information to pass into the PMIx server library versus what information to derive from
 13 other values at time of request is left to the host environment. PMIx implementors in turn may, if
 14 they choose, pre-parse registration data to create expanded views (thus enabling faster response
 15 to requests at the expense of memory footprint) or to compress views into tighter representations
 16 (thus trading minimized footprint for longer response times).

- Application-level information includes all application-specific values such as `PMIX_APP_SIZE` and `PMIX_APPLDR`. If the `job` contains only a single `application`, then the application-specific values can be specified independently - i.e., in their own `pmix_info_t` elements of the `info` array - or as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_APP_INFO_ARRAY` attribute and identified by including the `PMIX_APPNUM` attribute in the array. Use of the array format is *required* in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application *must* be provided using the `PMIX_APP_INFO_ARRAY` format, each identified by its `PMIX_APPNUM` value.

Upon conclusion of this step, the `info` array might look like that shown in 10.3, assuming there are two applications in the job being registered:

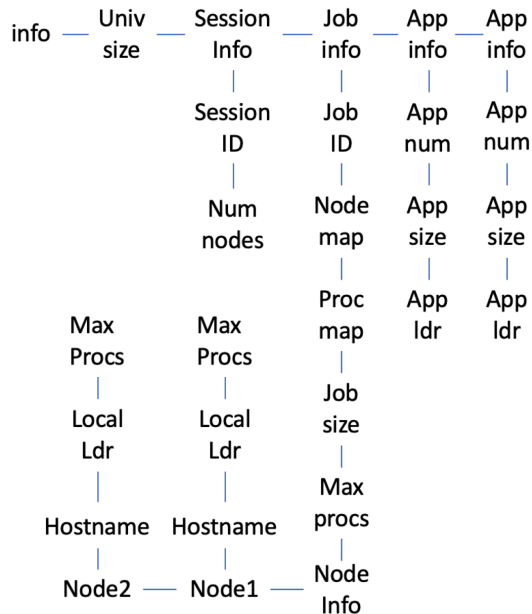


Figure 10.3.: Application-level information elements

- Process-level information includes an entry for each process in the job being registered, each entry marked with the `PMIX_PROC_DATA` attribute. The `rank` of the process *must* be the first entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the `info` array might look like the diagram in 10.4:
- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently - i.e., in their

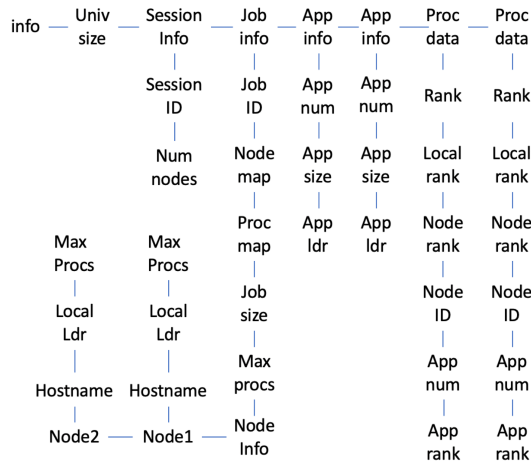


Figure 10.4.: Process-level information elements

1 own `pmix_info_t` elements of the `info` array. Alternatively, they can be provided as a
 2 `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_NODE_INFO_ARRAY`
 3 attribute - this is *required* in cases where non-specific attributes are passed to describe aspects of
 4 the node, or where values for multiple nodes are being provided.

5 The node-level information requires two elements that must be constructed in a manner similar to
 6 that used for the node map. The `PMIX_LOCAL_PEERS` value is computed based on the
 7 processes on the local node, filtered to select those from the job being registered, as shown below
 8 using the tools provided by PMIx:

C

```

9  char **ndppn = NULL;
10 char rank[30];
11 char *localranks;
12 size_t m;
13 pmix_info_t info;
14
15 for (m=0; m < mynode->num_procs; m++)
16     /* ignore processes that are not part of the target job */
17     if (!PMIX_CHECK_NAMESPACE(target job, mynode->proc[m].namespace))
18         continue;
19
20     snprintf(rank, 30, "%d", mynode->proc[m].rank);
21     PMIX_ARGV_APPEND(&ndppn, rank);
22
23     /* convert the array into a comma-delimited string of ranks */
  
```

```

1      localranks = PMIX_ARGV_JOIN(ndppn, ',');
2      /* release the local array */
3      PMIX_ARGV_FREE(ndppn);
4
5      /* pass the string as the value to the PMIX_LOCAL_PEERS key */
6      PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
7      /* release the list */
8      free(localranks);
9

```

C

10 The **PMIX_LOCAL_CPUSSETS** value is constructed in a similar manner. In the provided
11 example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a
12 comma-delimited string of processor IDs) of the processors assigned to each process has
13 previously been generated and stored on the process description. Thus, the value can be
14 constructed as shown below:

C

```

15      char **ndcpus = NULL;
16      char *localcpus;
17      size_t m;
18      pmix_info_t info;
19
20      for (m=0; m < mynode->num_procs; m++)
21          /* ignore processes that are not part of the target job */
22          if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].nspace))
23              continue;
24
25          PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
26
27          /* convert the array into a colon-delimited string */
28          localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
29          /* release the local array */
30          PMIX_ARGV_FREE(ndcpus);
31
32          /* pass the string as the value to the PMIX_LOCAL_CPUSSETS key */
33          PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSSETS, localcpus, PMIX_STRING);
34          /* release the list */
35          free(localcpus);
36

```

C

37 Note that for efficiency, these two values can be computed at the same time.

38 The final *info* array might therefore look like the diagram in [10.5](#):

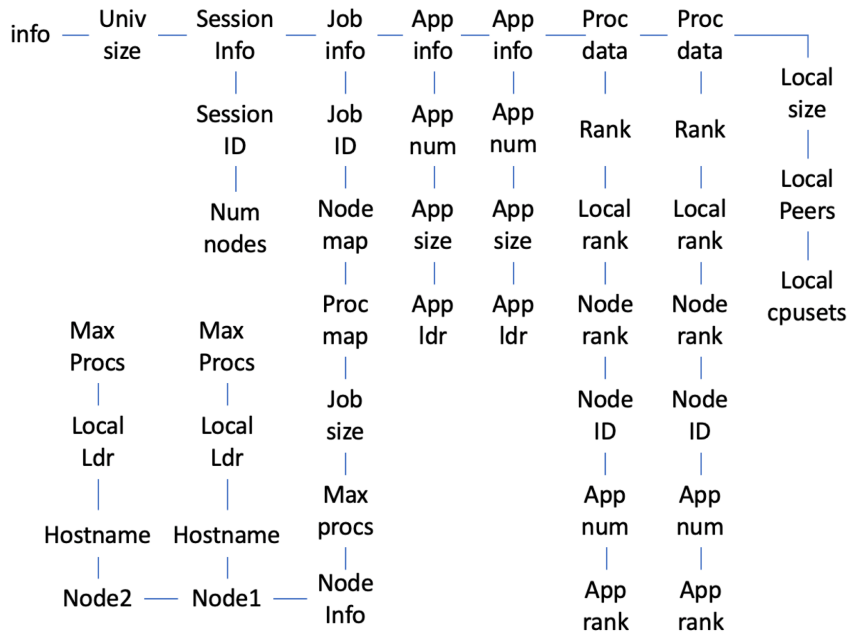


Figure 10.5.: Final information array

1 10.1.4 PMIx_server_deregister_namespace

2 Summary

3 Deregister a namespace.

4 Format

PMIx v1.0

```

C
void PMIx_server_deregister_namespace(const pmix_namespace_t nspace,
pmix_op_cbfunc_t cbfunc, void *cbdata)
C

```

- 7 **IN nspace**
8 Namespace (string)
- 9 **IN cbfunc**
10 Callback function `pmix_op_cbfunc_t` (function reference)
- 11 **IN cbdata**
12 Data to be passed to the callback function (memory reference)

Description

Deregister the specified *nospace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMI_x servers by providing an opportunity for the host RM to tell the PMI_x server library to release all memory for a completed job. Note that the library *must not* invoke the callback function prior to returning from the API.

10.1.5 PMI_x_server_register_client

Summary

Register a client process with the PMI_x server library.

Format

PMI_x v1.0

C

```
pmix_status_t
PMIx_server_register_client(const pmix_proc_t *proc,
                           uid_t uid, gid_t gid,
                           void *server_object,
                           pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **proc**
 pmix_proc_t structure (handle)

IN **uid**
 user id (integer)

IN **gid**
 group id (integer)

IN **server_object**
 (memory reference)

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

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

Returns one of the following:

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

1 **Description**

2 Register a client process with the PMIx server library.

3 The host server can also, if it desires, provide an object it wishes to be returned when a server
4 function is called that relates to a specific process. For example, the host server may have an object
5 that tracks the specific client. Passing the object to the library allows the library to provide that
6 object to the host server during subsequent calls related to that client, such as a
7 [pmix_server_client_connected_fn_t](#) function. This allows the host server to access
8 the object without performing a lookup based on the client's namespace and rank.

 ▼ **Advice to PMIx server hosts** ▲

9 Host environments are *required* to execute this operation prior to starting the client process. The
10 expected user ID and group ID of the child process allows the server library to properly authenticate
11 clients as they connect by requiring the two values to match. Accordingly, the detected user and
12 group ID's of the connecting process are not included in the
13 [pmix_server_client_connected_fn_t](#) server module function.



 ▼ **Advice to PMIx library implementers** ▼

14 For security purposes, the PMIx server library should check the user and group ID's of a
15 connecting process against those provided for the declared client process identifier via the
16 [PMIx_server_register_client](#) prior to completing the connection.



17 **10.1.6 PMIx_server_deregister_client**

18 **Summary**

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

1 Format

PMIx v1.0

```
2 void  
3 PMIx_server_deregister_client(const pmix_proc_t *proc,  
4                               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

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

11 Description

12 The `PMIx_server_deregister_namespace` API will delete all client information for that
13 namespace. The PMIx server library will automatically perform that operation upon disconnect of
14 all local clients. This API is therefore intended primarily for use in exception cases, but can be
15 called in non-exception cases if desired. Note that the library *must not* invoke the callback function
16 prior to returning from the API.

17 10.1.7 PMIx_server_setup_fork

18 Summary

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

20 Format

PMIx v1.0

```
21 pmix_status_t  
22 PMIx_server_setup_fork(const pmix_proc_t *proc,  
23                        char ***env)
```

```
24 IN proc  
25     pmix_proc_t structure (handle)  
26 IN env  
27     Environment array (array of strings)
```

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

1 **Description**

2 Setup the environment of a child process to be forked by the host so it can correctly interact with
3 the PMIx server.

▼ **Advice to PMIx server hosts** ▼

4 Host environments are *required* to execute this operation prior to starting the client process.
▲

5 The PMIx client needs some setup information so it can properly connect back to the server. This
6 function will set appropriate environmental variables for this purpose, and will also provide any
7 environmental variables that were specified in the launch command (e.g., via [PMIx_Spawn](#)) plus
8 other values (e.g., variables required to properly initialize the client’s fabric library).

9 **10.1.8 PMIx_server_dmodex_request**

10 **Summary**

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

12 **Format**

PMIx v1.0

```
▼ C ▼  
pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,  
                                          pmix_dmodex_response_fn_t cbfunc,  
                                          void *cbdata)  
▲ C ▲
```

- 16 **IN** **proc**
17 [pmix_proc_t](#) structure (handle)
- 18 **IN** **cbfunc**
19 Callback function [pmix_dmodex_response_fn_t](#) (function reference)
- 20 **IN** **cbdata**
21 Data to be passed to the callback function (memory reference)

22 Returns one of the following:

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

Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the `PMIx_Put` and `PMIx_Commit` functions followed by a `PMIx_Fence` barrier that globally exchanges the posted information. However, the barrier operation represents a significant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the `PMIx_server_dmodex_request` function, receiving the response in the provided `cbfunc` once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

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.

1

Format

PMIx v2.0

C

2

`pmix_status_t`

3

```
PMIx_server_setup_application(const pmix_namespace_t nspace,
```

4

```
    pmix_info_t info[], size_t ninfo,
```

5

```
    pmix_setup_application_cbfunc_t cbfunc,
```

6

```
    void *cbdata)
```

C

7

IN `nspace`

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_setup_application_cbfunc_t` (function reference)

15

IN `cbdata`

16

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

17

Returns one of the following:

18

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

19

20

21

- a PMIx error constant indicating either an error in the input - the *cbfunc* will *not* be called

22

Description

23

Provide a function by which the RM can request application-specific setup data (e.g., environmental variables, fabric configuration and security credentials) from supporting PMIx server library subsystems prior to initiating launch of an application.

24

25

Advice to PMIx server hosts

26

Host environments are *required* to execute this operation prior to launching an application.

27

This is defined as a non-blocking operation in case contributing subsystems need to perform some potentially time consuming action (e.g., query a remote service) before responding. The returned data must be distributed by the RM and subsequently delivered to the local PMIx server on each node where application processes will execute prior to initiating execution of those processes.

28

29

30

31

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

32

Advice to PMIx library implementers

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

10.1.10 PMIx_server_setup_local_support

Summary

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

Format

PMIx v2.0

```
pmix_status_t  
PMIx_server_setup_local_support(const pmix_namespace_t nspace,  
                                pmix_info_t info[], size_t ninfo,  
                                pmix_op_cbfunc_t cbfunc,  
                                void *cbdata);
```

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_op_cbfunc_t](#) (function reference)

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

Returns one of the following:

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

Description

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a network library might need to setup the local driver for “instant on” addressing. The data provided in the *info* array is the data provided to there host RM from the a call to `PMIx_server_setup_application`.

Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to starting any local application processes from the specified namespace.

10.2 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. Client calls to such functions will return a `PMIX_ERR_NOT_SUPPORTED` status.

The host RM will provide the function pointers in a `pmix_server_module_t` structure passed to `PMIx_server_init`. That module structure and associated function references are defined in this section.

Advice to PMIx server hosts

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

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

10.2.1 `pmix_server_module_t` Module

Summary

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

Format

C

```
1
2 typedef struct pmix_server_module_2_0_0_t
3     /* v1x interfaces */
4     pmix_server_client_connected_fn_t    client_connected;
5     pmix_server_client_finalized_fn_t    client_finalized;
6     pmix_server_abort_fn_t               abort;
7     pmix_server_fence_nb_fn_t            fence_nb;
8     pmix_server_dmodex_req_fn_t          direct_modex;
9     pmix_server_publish_fn_t             publish;
10    pmix_server_lookup_fn_t               lookup;
11    pmix_server_unpublish_fn_t            unpublish;
12    pmix_server_spawn_fn_t                spawn;
13    pmix_server_connect_fn_t              connect;
14    pmix_server_disconnect_fn_t            disconnect;
15    pmix_server_register_events_fn_t       register_events;
16    pmix_server_deregister_events_fn_t     deregister_events;
17    pmix_server_listener_fn_t              listener;
18    /* v2x interfaces */
19    pmix_server_notify_event_fn_t          notify_event;
20    pmix_server_query_fn_t                 query;
21    pmix_server_tool_connection_fn_t       tool_connected;
22    pmix_server_log_fn_t                   log;
23    pmix_server_alloc_fn_t                 allocate;
24    pmix_server_job_control_fn_t           job_control;
25    pmix_server_monitor_fn_t               monitor;
26 pmix_server_module_t;
```

C

27 10.2.2 pmix_server_client_connected_fn_t

28 Summary

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

30 Format

PMIx v1.0

C

```
1 typedef pmix_status_t (*pmix_server_client_connected_fn_t) (  
2     const pmix_proc_t *proc,  
3     void* server_object,  
4     pmix_op_cbfunc_t cbfunc,  
5     void *cbdata)
```

C

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

Returns one of the following:

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

Description

Notify the host environment that a client has called **PMIx_Init**. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The *server_object* parameter will be the value of the *server_object* parameter passed to **PMIx_server_register_client** by the host server when registering the connecting client. If provided, an implementation of **pmix_server_client_connected_fn_t** is only required to call the callback function designated. A host server can choose to not be notified when clients connect by setting **pmix_server_client_connected_fn_t** to **NULL**.

It is possible that only a subset of the clients in a namespace call **PMIx_Init**. The server's **pmix_server_client_connected_fn_t** implementation should not depend on being called once per rank in a namespace or delay calling the callback function until all ranks have connected. However, if a rank makes any PMIx calls, it must first call **PMIx_Init** and therefore the server's **pmix_server_client_connected_fn_t** will be called before any other server functions specific to the rank.

Advice to PMIx server hosts

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

10.2.3 pmix_server_client_finalized_fn_t

Summary

Notify the host environment that a client called `PMIx_Finalize`.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_client_finalized_fn_t) (  
    const pmix_proc_t *proc,  
    void* server_object,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

C

IN `proc`
 pmix_proc_t structure (handle)

IN `server_object`
 object reference (memory reference)

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

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

Returns one of the following:

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

Description

Notify the host environment that a client called `PMIx_Finalize`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The `server_object` parameter will be the value of the `server_object` parameter passed to `PMIx_server_register_client` by the host server when registering the connecting client. If provided, an implementation of `pmix_server_client_finalized_fn_t` is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting `pmix_server_client_finalized_fn_t` to `NULL`.

Note that the host server is only being informed that the client has called `PMIx_Finalize`. The client might not have exited. If a client exits without calling `PMIx_Finalize`, the server support library will not call the `pmix_server_client_finalized_fn_t` implementation.

Advice to PMIx server hosts

This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

10.2.4 `pmix_server_abort_fn_t`

Summary

Notify the host environment that a local client called `PMIx_Abort`.

Format

PMIx v1.0

C

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


1 **IN** **proc**
 2 **pmix_proc_t** structure identifying the process requesting the abort (handle)
 3 **IN** **server_object**
 4 object reference (memory reference)
 5 **IN** **status**
 6 exit status (integer)
 7 **IN** **msg**
 8 exit status message (string)
 9 **IN** **procs**
 10 Array of **pmix_proc_t** structures identifying the processes to be terminated (array of
 11 handles)
 12 **IN** **nprocs**
 13 Number of elements in the *procs* array (integer)
 14 **IN** **cbfunc**
 15 Callback function **pmix_op_cbfunc_t** (function reference)
 16 **IN** **cbdata**
 17 Data to be passed to the callback function (memory reference)

18 Returns one of the following:

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

26 **Description**

27 A local client called **PMIx_Abort** . Note that the client will be in a blocked state until the host
 28 server executes the callback function, thus allowing the PMIx server library to release the client.
 29 The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all
 30 processes in the client's namespace are to be terminated.

31 **10.2.5 pmix_server_fence_nb_fn_t**

32 **Summary**

33 At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

1

Format

PMIx v1.0

C

2

```

typedef pmix_status_t (*pmix_server_fence_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

10

IN **procs**

Array of [pmix_proc_t](#) structures identifying operation participants(array of handles)

11

12

IN **nprocs**

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

13

14

IN **info**

Array of info structures (array of handles)

15

16

IN **ninfo**

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

17

18

IN **data**

(string)

19

20

IN **ndata**

(integer)

21

22

IN **cbfunc**

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

23

24

IN **cbdata**

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

25

26

Returns one of the following:

27

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

28

29

30

- [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

31

32

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

33

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

The following attributes are required to be supported by all host environments:

PMIX_COLLECT_DATA "pmix.collect" (bool)

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

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx server hosts

Host environment are *required* to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is optional.

Description

All local clients in the provided array of *procs* called either `PMIx_Fence` or `PMIx_Fence_nb`. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the `PMIx_Fence` / `PMIx_Fence_nb` operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_fence_nb_fn_t` function.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

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

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.

1

Format

PMIx v1.0

C

2

```

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)

```

3

4

5

6

7

C

8

IN `proc`

`pmix_proc_t` structure identifying the process whose data is being requested (handle)

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 `pmix_modex_cbfunc_t` (function reference)

15

16

IN `cbdata`

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

17

18

Returns one of the following:

19

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

20

21

22

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

23

Required Attributes

24

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

25

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

26

PMIX_TIMEOUT "pmix.timeout" (int)

27

Time in seconds before the specified operation should time out (0 indicating infinite) in

28

error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

29

Description

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to **PMIx_Put** and **PMIx_Commit**.

The array of *info* structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional *unless* the *mandatory* flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

10.2.7 pmix_server_publish_fn_t

Summary

Publish data per the PMIx API specification.

Format

PMIx v1.0

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

- IN proc**
 pmix_proc_t structure of the process publishing the data (handle)
- IN info**
 Array of info structures (array of handles)
- IN ninfo**
 Number of elements in the *info* array (integer)
- IN cbfunc**
 Callback function **pmix_op_cbfunc_t** (function reference)
- IN cbdata**
 Data to be passed to the callback function (memory reference)

Returns one of the following:

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

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE "pmix.range" (pmix_data_range_t)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)
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.

Format

PMIx v1.0

C

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


1 **IN** **proc**
 2 **pmix_proc_t** structure of the process seeking the data (handle)
 3 **IN** **keys**
 4 (array of strings)
 5 **IN** **info**
 6 Array of info structures (array of handles)
 7 **IN** **ninfo**
 8 Number of elements in the *info* array (integer)
 9 **IN** **cbfunc**
 10 Callback function **pmix_lookup_cbfunc_t** (function reference)
 11 **IN** **cbdata**
 12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

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

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

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) (
    const pmix_proc_t *proc,
    char **keys,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata)

```

3

4

5

6

7

8

C

9

IN `proc`

`pmix_proc_t` structure identifying the process making the request (handle)

10

11

IN `keys`

(array of strings)

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 one of the following:

22

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

23

24

25

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the `cbfunc` will *not* be called

26

27

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

28

Required Attributes

29

PMIx libraries are required to pass any provided attributes to the host environment for processing.

30

In addition, the following attributes are required to be included in the passed `info` array:

31

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

32

33

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

34

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE "pmix.range" (`pmix_data_range_t`)

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

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)

```

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 one of the following:

25

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

26

27

28

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

29

30

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

31

Required Attributes

32

PMIx libraries are required to pass any provided attributes to the host environment for processing.

33

In addition, the following attributes are required to be included in the passed *info* array:

34

PMIX_USERID "pmix.euid" (`uint32_t`)

35

Effective user id.

1 **PMIX_GRPID** "pmix.egid" (uint32_t)

2 Effective group id.

3
4 Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED**
5 and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those
6 values can be returned to clients upon connection to the PMIx server. In addition, they are required
7 to support the following attributes when present in either the *job_info* or the *info* array of an
8 element of the *apps* array:

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

10 Working directory for spawned processes.

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

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

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

16 Prefix to use for starting spawned processes.

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

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

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

20 Hostfile to use for spawned processes.



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

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

23 Hostfile listing hosts to add to existing allocation.

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

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

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

27 Preload binaries onto nodes.

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

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

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

31 Name of personality to use.

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

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

4 `PMIX_DISPLAY_MAP "pmix.dispmap" (bool)`
5 Display process mapping upon spawn.

6 `PMIX_PPR "pmix.ppr" (char*)`
7 Number of processes to spawn on each identified resource.

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

12 `PMIX_RANKBY "pmix.rankby" (char*)`
13 Process ranking policy - when accessed using `PMIx_Get` , use the
14 `PMIX_RANK_WILDCARD` value for the rank to discover the ranking algorithm used for the
15 provided namespace

16 `PMIX_BINDTO "pmix.bindto" (char*)`
17 Process binding policy - when accessed using `PMIx_Get` , use the
18 `PMIX_RANK_WILDCARD` value for the rank to discover the binding policy used for the
19 provided namespace

20 `PMIX_NON_PMI "pmix.nonpmi" (bool)`
21 Spawned processes will not call `PMIx_Init` .

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

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

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

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

30 `PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)`
31 Spawned application consists of debugger daemons.

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

34 `PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)`
35 Timestamp output from applications.

36 `PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)`

1 Merge `stdout` and `stderr` streams from application processes.

2 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)

3 Output application output to the specified file.

4 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

5 Mark the `argv` with the rank of the process.

6 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

7 Number of cpus to assign to each rank - when accessed using `PMIx_Get`, use the
8 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
9 provided namespace

10 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)

11 Do not place processes on the head node.

12 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)

13 Do not oversubscribe the cpus.

14 **PMIX_REPORT_BINDINGS** "pmix.repbinding" (bool)

15 Report bindings of the individual processes.

16 **PMIX_CPU_LIST** "pmix.cpulists" (char*)

17 List of cpus to use for this job - when accessed using `PMIx_Get`, use the
18 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
19 namespace

20 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)

21 Application supports recoverable operations.

22 **PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)

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

24 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)

25 Maximum number of times to restart a job - when accessed using `PMIx_Get`, use the
26 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
27 namespace

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

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



Description

Spawn a set of applications/processes as per the [PMIx_Spawn](#) API. Note that applications are not required to be MPI or any other programming model. Thus, the host server cannot make any assumptions as to their required support. The callback function is to be executed once all processes have been started. An error in starting any application or process in this request shall cause all applications and processes in the request to be terminated, and an error returned to the originating caller.

Note that a timeout can be specified in the `job_info` array to indicate that failure to start the requested job within the given time should result in termination to avoid hangs.

10.2.11 pmix_server_connect_fn_t

Summary

Record the specified processes as *connected*.

Format

PMIx v1.0

C

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

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 one of the following:

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

Required Attributes

8 PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

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

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

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

14 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)

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

19 **PMIX_COLLECTIVE_ALGO_REQD** "pmix.calreqd" (bool)

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

1 **Description**

2 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The
3 callback is to be executed once every daemon hosting at least one participant has called the host
4 server's `pmix_server_connect_fn_t` function, *and* the host environment has completed any
5 supporting operations required to meet the terms of the PMIx definition of *connected* processes.

 ▼ Advice to PMIx library implementers ▼

6 The PMIx server library is required to aggregate participation by local clients, passing the request
7 to the host environment once all local participants have executed the API.



 ▼ Advice to PMIx server hosts ▼

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



11 **10.2.12 pmix_server_disconnect_fn_t**

12 **Summary**

13 Disconnect a previously connected set of processes.

1

Format

PMIx v1.0

C

2

```

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)

```

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 one of the following:

22

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

23

24

25

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

26

27

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

28

Required Attributes

29

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server’s `pmix_server_disconnect_fn_t` function, *and* the host environment has completed any required supporting operations.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

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

A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of *procs* was not previously *connected* via a call to the `pmix_server_connect_fn_t` function.

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) (
3     pmix_status_t *codes,
4     size_t ncodes,
5     const pmix_info_t info[],
6     size_t ninfo,
7     pmix_op_cbfunc_t cbfunc,
8     void *cbdata)

```

C

9 **IN codes**
 10 Array of `pmix_status_t` values (array of handles)

11 **IN ncodes**
 12 Number of elements in the `codes` array (integer)

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

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

17 **IN cbfunc**
 18 Callback function `pmix_op_cbfunc_t` (function reference)

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

21 Returns one of the following:

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

Required Attributes

29 PMIx libraries are required to pass any provided attributes to the host environment for processing.
 30 In addition, the following attributes are 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`)
 34 Effective group id.



1

Description

2

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.

3

Advice to PMIx library implementers

4

The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

5

6

- 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

7

8

9

- 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

10



Advice to PMIx server hosts

11

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).

12

13

14



15 10.2.14 pmix_server_deregister_events_fn_t

16

Summary

17

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

Array of `pmix_status_t` values (array of handles)

8

9

IN ncodes

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

10

11

IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

12

13

IN cbdata

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

14

15

Returns one of the following:

16

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

17

18

19

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the `cbfunc` will *not* be called

20

21

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

22

23

Description

24

Deregister to receive notifications for the specified events to which the PMIx server has previously registered.

25

Advice to PMIx library implementers

26

The PMIx server library must track all client registrations. This module function shall only be called when:

27

28

- the library is deregistering environmental codes (i.e., a PMIx codes in the range between **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER**, inclusive) or codes that lies outside the defined PMIx range of constants; and

29

30

31

- no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

32

33

1 10.2.15 pmix_server_notify_event_fn_t

2 Summary

3 Notify the specified processes of an event.

4 Format

PMIx v2.0

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t)(pmix_status_t code,  
const pmix_proc_t *source,  
pmix_data_range_t range,  
pmix_info_t info[],  
size_t ninfo,  
pmix_op_cbfunc_t cbfunc,  
void *cbdata);
```

- 12 **IN code**
13 The `pmix_status_t` event code being referenced structure (handle)
- 14 **IN source**
15 `pmix_proc_t` of process that generated the event (handle)
- 16 **IN range**
17 `pmix_data_range_t` range over which the event is to be distributed (handle)
- 18 **IN info**
19 Optional array of `pmix_info_t` structures containing additional information on the event
20 (array of handles)
- 21 **IN ninfo**
22 Number of elements in the *info* array (integer)
- 23 **IN cbfunc**
24 Callback function `pmix_op_cbfunc_t` (function reference)
- 25 **IN cbdata**
26 Data to be passed to the callback function (memory reference)

27 Returns one of the following:

- 28 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
29 will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function
30 prior to returning from the API.
- 31 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
32 returned *success* - the *cbfunc* will *not* be called

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

Required Attributes

3 PMIx libraries are required to pass any provided attributes to the host environment for processing.

4
5 Host environments that provide this module entry point are required to support the following
6 attributes:

7 **PMIX_RANGE** "pmix.range" (**pmix_data_range_t**)

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

Description

9
10 Notify the specified processes (described through a combination of *range* and attributes provided in
11 the *info* array) of an event generated either by the PMIx server itself or by one of its local clients.
12 The process generating the event is provided in the *source* parameter, and any further descriptive
13 information is included in the *info* array.

Advice to PMIx server hosts

14 The callback function is to be executed once the host environment no longer requires that the PMIx
15 server library maintain the provided data structures. It does *not* necessarily indicate that the event
16 has been delivered to any process, nor that the event has been distributed for delivery

17 10.2.16 pmix_server_listener_fn_t

18 Summary

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

1 **Format**

PMIx v1.0

C

```
2 typedef pmix_status_t (*pmix_server_listener_fn_t) (  
3     int listening_sd,  
4     pmix_connection_cbfunc_t cbfunc,  
5     void *cbdata)
```

C

6 **IN** `incoming_sd`
7 (integer)
8 **IN** `cbfunc`
9 Callback function `pmix_connection_cbfunc_t` (function reference)
10 **IN** `cbdata`
11 (memory reference)

12 Returns `PMIX_SUCCESS` indicating that the request is accepted, or a negative value
13 corresponding to a PMIx error constant indicating that the request has been rejected.

14 **Description**

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

21 **10.2.17 pmix_server_query_fn_t**

22 **Summary**

23 Query information from the resource manager.

24 **Format**

PMIx v2.0

C

```
25 typedef pmix_status_t (*pmix_server_query_fn_t) (  
26     pmix_proc_t *proct,  
27     pmix_query_t *queries, size_t nqueries,  
28     pmix_info_cbfunc_t cbfunc,  
29     void *cbdata)
```

1 **IN** `proct`
 2 `pmix_proc_t` structure of the requesting process (handle)
 3 **IN** `queries`
 4 Array of `pmix_query_t` structures (array of handles)
 5 **IN** `nqueries`
 6 Number of elements in the `queries` array (integer)
 7 **IN** `cbfunc`
 8 Callback function `pmix_info_cbfunc_t` (function reference)
 9 **IN** `cbdata`
 10 Data to be passed to the callback function (memory reference)

11 Returns one of the following:

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

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.

Optional Attributes

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

26 **PMIX_QUERY_NAMESPACES** "`pmix.qry.ns`" (`char*`)
 27 Request a comma-delimited list of active namespaces.

28 **PMIX_QUERY_JOB_STATUS** "`pmix.qry.jst`" (`pmix_status_t`)
 29 Status of a specified, currently executing job.

30 **PMIX_QUERY_QUEUE_LIST** "`pmix.qry qlst`" (`char*`)
 31 Request a comma-delimited list of scheduler queues.

32 **PMIX_QUERY_QUEUE_STATUS** "`pmix.qry.qst`" (TBD)

1 Status of a specified scheduler queue.

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

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

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

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

9 **PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (bool)

10 Return a comma-delimited list of supported spawn attributes.

11 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)

12 Return a comma-delimited list of supported debug attributes.

13 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)

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

15 **PMIX_QUERY_LOCAL_ONLY** "pmix.qry.local" (bool)

16 Constrain the query to local information only.

17 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)

18 Report average values.

19 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)

20 Report minimum and maximum values.

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

22 String identifier of the allocation whose status is being requested.

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

24 Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
25



26 Description

27 Query information from the host environment. The query will include the namespace/rank of the
28 process that is requesting the info, an array of **pmix_query_t** describing the request, and a
29 callback function/data for the return.



Advice to PMIx library implementers

30 The PMIx server library should not block in this function as the host environment may, depending
31 upon the information being requested, require significant time to respond.



1 10.2.18 pmix_server_tool_connection_fn_t

2 Summary

3 Register that a tool has connected to the server.

4 Format

PMIx v2.0

```
▼────────────────────────────────────────── C ───────────────────────────────────▼  
5 typedef void (*pmix_server_tool_connection_fn_t) (  
6             pmix_info_t info[], size_t ninfo,  
7             pmix_tool_connection_cbfnc_t cbfunc,  
8             void *cbdata)  
▲────────────────────────────────────────── C ───────────────────────────────────▲
```

- 9 **IN info**
10 Array of [pmix_info_t](#) structures (array of handles)
- 11 **IN ninfo**
12 Number of elements in the *info* array (integer)
- 13 **IN cbfunc**
14 Callback function [pmix_tool_connection_cbfnc_t](#) (function reference)
- 15 **IN cbdata**
16 Data to be passed to the callback function (memory reference)

▼── Required Attributes ───────────────────────────────────▼

17 PMIx libraries are required to pass the following attributes in the *info* array:

- 18 **PMIX_USERID** "pmix.euid" (uint32_t)
19 Effective user id.
- 20 **PMIX_GRPID** "pmix.egid" (uint32_t)
21 Effective group id.



▼── Optional Attributes ───────────────────────────────────▼

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

- 23 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
24 Forward **stdout** from spawned processes to this process.
- 25 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)
26 Forward **stderr** from spawned processes to this process.
- 27 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)
28 Forward this process's **stdin** to the designated process.



Description

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The `pmix_info_t` array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

Advice to PMIx server hosts

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests. The host *must not* execute the callback function prior to returning from the API.

10.2.19 pmix_server_log_fn_t

Summary

Log data on behalf of a client.

Format

PMIx v2.0

```
typedef void (*pmix_server_log_fn_t) (  
    const pmix_proc_t *client,  
    const pmix_info_t data[], size_t ndata,  
    const pmix_info_t directives[], size_t ndirs,  
    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

IN client
 `pmix_proc_t` structure (handle)

IN data
 Array of info structures (array of handles)

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

IN directives
 Array of info structures (array of handles)

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

IN cbfunc
 Callback function `pmix_op_cbfunc_t` (function reference)

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_LOG_STDERR "pmix.log.stderr" (char*)

Log string to **stderr**.

PMIX_LOG_STDOUT "pmix.log.stdout" (char*)

Log string to **stdout**.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)

Log data to syslog. Defaults to **ERROR** priority.

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.

1 **Description**

2 Log data on behalf of a client. This function is *not* intended for output of computational results, but
3 rather for reporting status and error messages. The host *must not* execute the callback function prior
4 to returning from the API.

5 **10.2.20 pmix_server_alloc_fn_t**

6 **Summary**

7 Request allocation operations on behalf of a client.

8 **Format**

PMIx v2.0

```
▼ C ▼
typedef pmix_status_t (*pmix_server_alloc_fn_t) (
    const pmix_proc_t *client,
    pmix_alloc_directive_t directive,
    const pmix_info_t data[], size_t ndata,
    pmix_info_cbfunc_t cbfunc, void *cbdata)
▲ C ▲
```

- 14 **IN client**
15 `pmix_proc_t` structure of process making request (handle)
- 16 **IN directive**
17 Specific action being requested (`pmix_alloc_directive_t`)
- 18 **IN data**
19 Array of info structures (array of handles)
- 20 **IN ndata**
21 Number of elements in the *data* array (integer)
- 22 **IN cbfunc**
23 Callback function `pmix_info_cbfunc_t` (function reference)
- 24 **IN cbdata**
25 Data to be passed to the callback function (memory reference)

26 Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_ALLOC_ID "pmix.alloc.id" (char*)

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*)

1 Name of the network.
2 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (float)
3 Mbits/sec.
4 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)
5 Quality of service level.



6 **Description**

7 Request new allocation or modifications to an existing allocation on behalf of a client. Several
8 broad categories are envisioned, including the ability to:

- 9 • Request allocation of additional resources, including memory, bandwidth, and compute for an
10 existing allocation. Any additional allocated resources will be considered as part of the current
11 allocation, and thus will be released at the same time.
- 12 • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
13 affiliated with) the allocation of the requestor - thus the termination of one allocation will not
14 impact the other.
- 15 • Extend the reservation on currently allocated resources, subject to scheduling availability and
16 priorities.
- 17 • Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to
18 the scheduler with a promise to return them upon subsequent request.

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

22 **10.2.21 pmix_server_job_control_fn_t**

23 **Summary**

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

1 **Format**

PMIx v2.0

C

```

2 typedef pmix_status_t (*pmix_server_job_control_fn_t) (
3     const pmix_proc_t *requestor,
4     const pmix_proc_t targets[], size_t ntargets,
5     const pmix_info_t directives[], size_t ndirs,
6     pmix_info_cbfunc_t cbfunc, void *cbdata)

```

C

- 7 **IN requestor**
 pmix_proc_t structure of requesting process (handle)
- 8
- 9 **IN targets**
 Array of proc structures (array of handles)
- 10
- 11 **IN ntargets**
 Number of elements in the *targets* array (integer)
- 12
- 13 **IN directives**
 Array of info structures (array of handles)
- 14
- 15 **IN ndirs**
 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 one of the following:

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

Required Attributes

29 PMIx libraries are required to pass any provided attributes to the host environment for processing.
30 In addition, the following attributes are 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)
 34 Effective group id.

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

Host environments that provide this module entry point are required to support the following attributes:

- PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
Provide a string identifier for this request.
- PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
Pause the specified processes.
- PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
Resume ("un-pause") the specified processes.
- PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
Forcibly terminate the specified processes and cleanup.
- PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
Send given signal to specified processes.
- PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
Politely terminate the specified processes.

▲-----▲
▼-----▼ **Optional Attributes** -----▼

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

- PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
Cancel the specified request (**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*)

1 Regular expression identifying nodes that are to be provisioned.

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

3 Name of the image that is to be provisioned.

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

5 Indicate that the job can be pre-empted.



6 **Description**

7 Execute a job control action on behalf of a client. The *targets* array identifies the processes to

8 which the requested job control action is to be applied. A **NULL** value can be used to indicate all

9 processes in the caller's namespace. The use of **PMIX_RANK_WILDARD** can also be used to

10 indicate that all processes in the given namespace are to be included.

11 The directives are provided as **pmix_info_t** structures in the *directives* array. The callback

12 function provides a *status* to indicate whether or not the request was granted, and to provide some

13 information as to the reason for any denial in the **pmix_info_cbfunc_t** array of

14 **pmix_info_t** structures.

15 **10.2.22 pmix_server_monitor_fn_t**

16 **Summary**

17 Request that a client be monitored for activity.

18 **Format**

19 *PMIx v2.0*

```

20 /* Request that a client be monitored for activity */
21 typedef pmix_status_t (*pmix_server_monitor_fn_t) (
22     const pmix_proc_t *requestor,
23     const pmix_info_t *monitor, pmix_status_t error,
24     const pmix_info_t directives[], size_t ndirs,
25     pmix_info_cbfunc_t cbfunc, void *cbdata);

```

25 **IN requestor**
 26 **pmix_proc_t** structure of requesting process (handle)

27 **IN monitor**
 28 **pmix_info_t** identifying the type of monitor being requested (handle)

29 **IN error**
 30 Status code to use in generating event if alarm triggers (integer)

1 **IN directives**
2 Array of info structures (array of handles)
3 **IN ndirs**
4 Number of elements in the *info* array (integer)
5 **IN cbfunc**
6 Callback function `pmix_op_cbfunc_t` (function reference)
7 **IN cbdata**
8 Data to be passed to the callback function (memory reference)

9 Returns one of the following:

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

17 This entry point is only called for monitoring requests that are not directly supported by the PMIx
18 server library itself.

▼----- Required Attributes -----▼

19 If supported by the PMIx server library, then the library must not pass any supported attributes to
20 the host environment. All attributes not directly supported by the server library must be passed to
21 the host environment if it provides this module entry. In addition, the following attributes are
22 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 _____

28 Host environments are not required to support any specific monitoring attributes.

▲-----

Optional Attributes

The following attributes may be implemented by a host environment.

- PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
Provide a string identifier for this request.
- PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
Identifier to be canceled (NULL means cancel all monitoring for this process).
- PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
The application desires to control the response to a monitoring event.
- PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)
Register to have the PMIx server monitor the requestor for heartbeats.
- PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)
Time in seconds before declaring heartbeat missed.
- PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)
Number of heartbeats that can be missed before generating the event.
- PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)
Register to monitor file for signs of life.
- PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)
Monitor size of given file is growing to determine if the application is running.
- PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)
Monitor time since last access of given file to determine if the application is running.
- PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)
Monitor time since last modified of given file to determine if the application is running.
- PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)
Time in seconds between checking the file.
- PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)
Number of file checks that can be missed before generating the event.

Description

Request that a client be monitored for activity.

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

- application, [10](#), [11](#), [64](#), [65](#), [106](#), [114](#), [163](#),
[203](#), [207](#)
 - Defintion, [13](#)
- host environment
 - Defintion, [14](#)
- job, [10](#), [11](#), [64](#), [65](#), [106](#), [114](#), [116](#), [163](#), [199](#),
[202](#), [203](#), [205](#), [207](#)
 - Defintion, [13](#)
- namespace
 - Defintion, [13](#)
- PMIx_Abort, [8](#), [26](#), [137](#), [222](#), [223](#)
 - Defintion, [136](#)
- PMIX_ADD_HOST, [139](#), [143](#), [236](#)
 - Defintion, [72](#)
- PMIX_ADD_HOSTFILE, [139](#), [143](#), [236](#)
 - Defintion, [72](#)
- PMIX_ALLOC_BANDWIDTH, [165](#), [257](#)
 - Defintion, [77](#)
- PMIX_ALLOC_CPU_LIST, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_DIRECTIVE, [58](#)
- PMIx_Alloc_directive_string, [9](#)
 - Defintion, [92](#)
- pmix_alloc_directive_t, [42](#), [58](#), [92](#), [255](#)
 - Defintion, [42](#)
- PMIX_ALLOC_EXTEND, [42](#)
- PMIX_ALLOC_EXTERNAL, [42](#)
- PMIX_ALLOC_ID, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_MEM_SIZE, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NETWORK, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NETWORK_ID, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NETWORK_QOS, [166](#),
[257](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NEW, [42](#)
- PMIX_ALLOC_NODE_LIST, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NUM_CPU_LIST, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NUM_CPUS, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_NUM_NODES, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOC_REAQUIRE, [42](#)
- PMIX_ALLOC_RELEASE, [42](#)
- PMIX_ALLOC_TIME, [165](#), [256](#)
 - Defintion, [77](#)
- PMIX_ALLOCATED_NODELIST, [200](#)
 - Defintion, [63](#)
- PMIx_Allocation_request_nb, [9](#), [77](#), [156](#),
[166](#)
 - Defintion, [164](#)
- PMIX_ANL_MAP
 - Defintion, [69](#)
- PMIX_APP, [57](#)
- PMIX_APP_CONSTRUCT
 - Defintion, [46](#)
- PMIX_APP_CREATE
 - Defintion, [46](#)
- PMIX_APP_DESTRUCT
 - Defintion, [46](#)
- PMIX_APP_FREE
 - Defintion, [47](#)
- PMIX_APP_INFO, [109](#), [112](#), [117](#), [159](#)
 - Defintion, [64](#)

PMIX_APP_INFO_ARRAY, 65, 207
 Defintion, 65
 PMIX_APP_INFO_CREATE, 11
 Defintion, 47
 PMIX_APP_MAP_REGEX
 Defintion, 69
 PMIX_APP_MAP_TYPE
 Defintion, 69
 PMIX_APP_RANK, 199
 Defintion, 63
 PMIX_APP_SIZE, 116, 199, 207
 Defintion, 66
 pmix_app_t, 11, 45–47, 138, 142, 235
 Defintion, 45
 PMIX_APPLDR, 199, 207
 Defintion, 63
 PMIX_APPNUM, 64, 65, 109, 112, 116,
 159, 199, 207
 Defintion, 63
 PMIX_ARCH
 Defintion, 62
 PMIX_ATTR_UNDEF
 Defintion, 59
 PMIX_AVAIL_PHYS_MEMORY, 200
 Defintion, 67
 PMIX_BINDTO, 140, 144, 200, 237
 Defintion, 73
 PMIX_BOOL, 57
 PMIX_BUFFER, 57
 PMIX_BYTE, 57
 PMIX_BYTE_OBJECT, 58
 PMIX_BYTE_OBJECT_CREATE
 Defintion, 52
 PMIX_BYTE_OBJECT_DESTRUCT
 Defintion, 51
 PMIX_BYTE_OBJECT_FREE
 Defintion, 52
 PMIX_BYTE_OBJECT_LOAD
 Defintion, 52
 pmix_byte_object_t, 51, 52, 58
 Defintion, 51
 PMIX_CLIENT_AVG_MEMORY
 Defintion, 67
 PMIX_COLLECT_DATA, 120, 122, 225
 Defintion, 68
 PMIX_COLLECTIVE_ALGO, 10, 120,
 123, 148, 151, 225, 240
 Defintion, 68
 PMIX_COLLECTIVE_ALGO_REQD, 120,
 123, 148, 151, 225, 240
 Defintion, 68
 PMIX_COMMAND, 58
 PMIx_Commit, 8, 89, 107, 119, 146, 215,
 228
 Defintion, 119
 PMIX_COMPRESSED_STRING, 58
 PMIx_Connect, 8, 10, 20, 141, 146, 149,
 151, 153
 Defintion, 147
 PMIX_CONNECT_MAX_RETRIES, 99
 Defintion, 60
 PMIx_Connect_nb, 8, 149
 Defintion, 149
 PMIX_CONNECT_RETRY_DELAY, 99
 Defintion, 60
 PMIX_CONNECT_SYSTEM_FIRST, 99,
 101
 Defintion, 60
 PMIX_CONNECT_TO_SYSTEM, 99, 101
 Defintion, 60
 pmix_connection_cbfunc_t, 249
 Defintion, 89
 PMIX_COSPAWN_APP
 Defintion, 73
 PMIX_CPU_LIST, 141, 145, 238
 Defintion, 74
 PMIX_CPUS_PER_PROC, 140, 145, 238
 Defintion, 73
 PMIX_CPUSET
 Defintion, 62
 PMIX_CREDENTIAL
 Defintion, 62
 PMIX_DAEMON_MEMORY
 Defintion, 67
 PMIX_DATA_ARRAY, 58
 PMIX_DATA_ARRAY_CONSTRUCT

Defintion, **30, 55**
 PMIX_DATA_ARRAY_CREATE
 Defintion, **31, 56**
 PMIX_DATA_ARRAY_DESTRUCT
 Defintion, **31, 56**
 PMIX_DATA_ARRAY_FREE
 Defintion, **31**
 PMIX_DATA_ARRAY_RELEASE
 Defintion, **56**
 pmix_data_array_t, **10, 30–32, 55, 56, 58, 74, 75, 161, 205, 207, 208, 251**
 Defintion, **30, 55**
 PMIX_DATA_BUFFER_CONSTRUCT,
 189, 191
 Defintion, **53, 186**
 PMIX_DATA_BUFFER_CREATE, **189, 191**
 Defintion, **54, 185**
 PMIX_DATA_BUFFER_DESTRUCT
 Defintion, **54, 186**
 PMIX_DATA_BUFFER_LOAD
 Defintion, **54, 187**
 PMIX_DATA_BUFFER_RELEASE
 Defintion, **54, 186**
 pmix_data_buffer_t, **53–55, 185–190, 194**
 Defintion, **53**
 PMIX_DATA_BUFFER_UNLOAD
 Defintion, **55, 187**
 PMIx_Data_copy, **9**
 Defintion, **192**
 PMIx_Data_copy_payload, **9**
 Defintion, **193**
 PMIx_Data_pack, **9, 189**
 Defintion, **188**
 PMIx_Data_print, **9**
 Defintion, **192**
 PMIX_DATA_RANGE, **58**
 PMIx_Data_range_string, **9**
 Defintion, **91**
 pmix_data_range_t, **29, 58, 91, 183, 247**
 Defintion, **29**
 PMIX_DATA_SCOPE, **108, 112**
 Defintion, **68**
 PMIX_DATA_TYPE, **58**
 PMIX_DATA_TYPE_MAX, **58**
 PMIx_Data_type_string, **9**
 Defintion, **92**
 pmix_data_type_t, **30, 31, 34, 38, 44, 55–57, 92, 189, 190, 192, 193**
 Defintion, **57**
 PMIx_Data_unpack, **9**
 Defintion, **190**
 PMIX_DEBUG_JOB
 Defintion, **76**
 PMIX_DEBUG_STOP_IN_INIT
 Defintion, **76**
 PMIX_DEBUG_STOP_ON_EXEC
 Defintion, **76**
 PMIX_DEBUG_WAIT_FOR_NOTIFY
 Defintion, **76**
 PMIX_DEBUG_WAITING_FOR_NOTIFY
 Defintion, **76**
 PMIX_DEBUGGER_DAEMONS, **140, 144, 237**
 Defintion, **73**
 PMIx_Deregister_event_handler, **9**
 Defintion, **181**
 PMIx_Disconnect, **8, 10, 20, 149, 153, 155**
 Defintion, **151**
 PMIx_Disconnect_nb, **8, 155**
 Defintion, **153**
 PMIX_DISPLAY_MAP, **139, 144, 237**
 Defintion, **72**
 pmix_dmodex_response_fn_t, **214**
 Defintion, **88**
 PMIX_DOUBLE, **57**
 PMIX_DSTPATH
 Defintion, **60**
 PMIX_EMBED_BARRIER, **97**
 Defintion, **68**
 PMIX_ERR_BAD_PARAM, **19**
 PMIX_ERR_COMM_FAILURE, **20**
 PMIX_ERR_DATA_VALUE_NOT_FOUND,
 19
 PMIX_ERR_DEBUGGER_RELEASE, **19**
 PMIX_ERR_EVENT_REGISTRATION, **20**

PMIX_ERR_HANDSHAKE_FAILED, 19
 PMIX_ERR_IN_ERRNO, 19
 PMIX_ERR_INIT, 19
 PMIX_ERR_INVALID_ARG, 19
 PMIX_ERR_INVALID_ARGS, 20
 PMIX_ERR_INVALID_CRED, 19
 PMIX_ERR_INVALID_KEY, 19
 PMIX_ERR_INVALID_KEY_LENGTH, 19
 PMIX_ERR_INVALID_KEYVALP, 20
 PMIX_ERR_INVALID_LENGTH, 20
 PMIX_ERR_INVALID_NAMESPACE, 20
 PMIX_ERR_INVALID_NUM_ARGS, 20
 PMIX_ERR_INVALID_NUM_PARSED, 20
 PMIX_ERR_INVALID_OPERATION, 21
 PMIX_ERR_INVALID_SIZE, 20
 PMIX_ERR_INVALID_TERMINATION, 20
 PMIX_ERR_INVALID_VAL, 19
 PMIX_ERR_INVALID_VAL_LENGTH, 20
 PMIX_ERR_JOB_TERMINATED, 20
 PMIX_ERR_LOST_CONNECTION_TO_CLIENT, 20
 PMIX_ERR_LOST_CONNECTION_TO_SERVER, 20
 PMIX_ERR_LOST_PEER_CONNECTION, 20
 PMIX_ERR_NO_PERMISSIONS, 19
 PMIX_ERR_NODE_DOWN, 21
 PMIX_ERR_NODE_OFFLINE, 21
 PMIX_ERR_NOMEM, 19
 PMIX_ERR_NOT_FOUND, 20
 PMIX_ERR_NOT_IMPLEMENTED, 20
 PMIX_ERR_NOT_SUPPORTED, 20
 PMIX_ERR_OUT_OF_RESOURCE, 19
 PMIX_ERR_PACK_FAILURE, 19
 PMIX_ERR_PACK_MISMATCH, 19
 PMIX_ERR_PROC_ABORTED, 19
 PMIX_ERR_PROC_ABORTING, 19
 PMIX_ERR_PROC_CHECKPOINT, 19
 PMIX_ERR_PROC_ENTRY_NOT_FOUND, 19
 PMIX_ERR_PROC_MIGRATE, 19
 PMIX_ERR_PROC_REQUESTED_ABORT, 19
 PMIX_ERR_PROC_RESTART, 19
 PMIX_ERR_READY_FOR_HANDSHAKE, 19
 PMIX_ERR_RESOURCE_BUSY, 19
 PMIX_ERR_SERVER_FAILED_REQUEST, 19
 PMIX_ERR_SERVER_NOT_AVAIL, 20
 PMIX_ERR_SILENT, 19
 PMIX_ERR_SYS_OTHER, 21
 PMIX_ERR_TIMEOUT, 19
 PMIX_ERR_TYPE_MISMATCH, 19
 PMIX_ERR_UNKNOWN_DATA_TYPE, 19
 PMIX_ERR_UNPACK_FAILURE, 19
 PMIX_ERR_UNPACK_INADEQUATE_SPACE, 19
 PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, 20
 PMIX_ERR_UNREACH, 19
 PMIX_ERR_UPDATE_ENDPOINTS, 20
 PMIX_ERR_WOULD_BLOCK, 19
 PMIX_ERROR, 19
 PMIX_ERROR_GROUP_ABORT
 Defintion, 70
 PMIX_ERROR_GROUP_COMM
 Defintion, 70
 PMIX_ERROR_GROUP_GENERAL
 Defintion, 70
 PMIX_ERROR_GROUP_LOCAL
 Defintion, 70
 PMIX_ERROR_GROUP_MIGRATE
 Defintion, 70
 PMIX_ERROR_GROUP_NODE
 Defintion, 70
 PMIX_ERROR_GROUP_RESOURCE
 Defintion, 70
 PMIX_ERROR_GROUP_SPAWN
 Defintion, 70
 PMIX_ERROR_HANDLER_ID
 Defintion, 70
 PMIX_ERROR_NAME
 Defintion, 70

PMIx_Error_string, 9
 Defintion, **91**
 PMIX_EVENT_ACTION_COMPLETE, 21
 PMIX_EVENT_ACTION_DEFERRED, 21
 PMIX_EVENT_ACTION_TIMEOUT, 180
 Defintion, **71**
 PMIX_EVENT_AFFECTED_PROC, 180,
 184
 Defintion, **71**
 PMIX_EVENT_AFFECTED_PROCS, 180,
 184
 Defintion, **71**
 PMIX_EVENT_BASE, 96, 100, 104
 Defintion, **59**
 PMIX_EVENT_CUSTOM_RANGE, 179,
 183
 Defintion, **71**
 PMIX_EVENT_DO_NOT_CACHE
 Defintion, **71**
 PMIX_EVENT_HDLR_AFTER, 179
 Defintion, **71**
 PMIX_EVENT_HDLR_APPEND, 179
 Defintion, **71**
 PMIX_EVENT_HDLR_BEFORE, 179
 Defintion, **70**
 PMIX_EVENT_HDLR_FIRST, 179
 Defintion, **70**
 PMIX_EVENT_HDLR_FIRST_IN_CATEGORY,
 179
 Defintion, **70**
 PMIX_EVENT_HDLR_LAST, 179
 Defintion, **70**
 PMIX_EVENT_HDLR_LAST_IN_CATEGORY,
 179
 Defintion, **70**
 PMIX_EVENT_HDLR_NAME, 179
 Defintion, **70**
 PMIX_EVENT_HDLR_PREPEND, 179
 Defintion, **71**
 PMIX_EVENT_NO_ACTION_TAKEN, 21
 PMIX_EVENT_NO_TERMINATION
 Defintion, **71**
 PMIX_EVENT_NON_DEFAULT, 183
 Defintion, **71**
 pmix_event_notification_cbfunc_fn_t, 85, 86
 Defintion, **85**
 PMIX_EVENT_PARTIAL_ACTION_TAKEN,
 21
 PMIX_EVENT_PROXY
 Defintion, **71**
 PMIX_EVENT_RETURN_OBJECT, 180
 Defintion, **71**
 PMIX_EVENT_SILENT_TERMINATION,
 180
 Defintion, **71**
 PMIX_EVENT_TERMINATE_JOB, 180
 Defintion, **71**
 PMIX_EVENT_TERMINATE_NODE, 180
 Defintion, **71**
 PMIX_EVENT_TERMINATE_PROC, 180
 Defintion, **71**
 PMIX_EVENT_TERMINATE_SESSION,
 180
 Defintion, **71**
 PMIX_EVENT_TEXT_MESSAGE
 Defintion, **71**
 PMIX_EVENT_WANT_TERMINATION
 Defintion, **72**
 pmix_evhdlr_reg_cbfunc_t, 84, 179
 Defintion, **84**
 PMIX_EXISTS, 19
 PMIX_EXTERNAL_ERR_BASE, 21
 PMIx_Fence, 3, 7, 8, 12, 104, 121, 123, 146,
 149, 153, 215, 223, 226
 Defintion, **119**
 PMIx_Fence_nb, 8, 82, 123, 223, 226
 Defintion, **121**
 PMIx_Finalize, 8, 20, 26, 68, 96, 97, 146,
 221, 222
 Defintion, **97**
 PMIX_FLOAT, 57
 PMIX_FWD_STDERR, 140, 144, 237, 252
 Defintion, **73**
 PMIX_FWD_STDIN, 140, 144, 237, 252
 Defintion, **73**
 PMIX_FWD_STDOUT, 140, 144, 237, 252

Definition, **73**
 PMIX_GDS_ACTION_COMPLETE, **21**
 PMIX_GDS_MODULE, **96, 100, 104**
 Definition, **62**
 PMIx_generate_ppn, **8**
 Definition, **196**
 PMIx_generate_regex, **8, 202**
 Definition, **195**
 PMIx_Get, **3, 8, 10, 32, 58–62, 66, 68–78, 96, 108, 110, 112–116, 118, 139–141, 143–145, 162, 163, 198, 200, 236–238**
 Definition, **107**
 PMIx_Get_nb, **8, 83**
 Definition, **110**
 PMIx_Get_version, **9, 15**
 Definition, **94**
 PMIX_GLOBAL, **29**
 PMIX_GLOBAL_RANK, **201**
 Definition, **63**
 PMIX_GRPID, **125, 127, 129, 131, 133, 135, 160, 165, 168, 171, 174, 229–234, 236, 244, 250, 252, 254, 256, 258, 261**
 Definition, **60**
 PMIx_Heartbeat, **9**
 Definition, **172**
 PMIX_HOST, **139, 143, 236**
 Definition, **72**
 PMIX_HOSTFILE, **139, 143, 236**
 Definition, **72**
 PMIX_HOSTNAME, **65, 109, 112, 118, 160, 201**
 Definition, **63**
 PMIX_HWLOC_SHMEM_ADDR
 Definition, **67**
 PMIX_HWLOC_SHMEM_FILE
 Definition, **67**
 PMIX_HWLOC_SHMEM_SIZE
 Definition, **67**
 PMIX_HWLOC_XML_V1, **200**
 Definition, **67**
 PMIX_HWLOC_XML_V2, **200**
 Definition, **68**
 PMIX_IMMEDIATE, **108, 112**
 Definition, **68**
 PMIX_INDEX_ARGV, **140, 145, 238**
 Definition, **73**
 PMIX_INFO, **57**
 PMIX_INFO_ARRAY, **58**
 pmix_info_array, **36**
 Definition, **36**
 PMIX_INFO_ARRAY_END, **40**
 pmix_info_cbfunc_t, **79, 83, 158, 164, 167, 169, 170, 172, 250, 255, 257, 260**
 Definition, **83**
 PMIX_INFO_CONSTRUCT
 Definition, **37**
 PMIX_INFO_CREATE, **40, 42**
 Definition, **37**
 PMIX_INFO_DESTRUCT
 Definition, **37**
 PMIX_INFO_DIRECTIVES, **58**
 PMIx_Info_directives_string, **9**
 Definition, **92**
 pmix_info_directives_t, **39, 40, 92**
 Definition, **39**
 PMIX_INFO_FREE
 Definition, **37**
 PMIX_INFO_IS_END, **11**
 Definition, **42**
 PMIX_INFO_IS_OPTIONAL, **11**
 Definition, **41**
 PMIX_INFO_IS_REQUIRED, **39, 40**
 Definition, **41**
 PMIX_INFO_LOAD
 Definition, **38**
 PMIX_INFO_OPTIONAL
 Definition, **41**
 PMIX_INFO_REQD, **40**
 PMIX_INFO_REQUIRED, **39**
 Definition, **40**
 pmix_info_t, **3, 9, 11, 12, 29, 36–42, 47, 49, 65, 75, 77, 78, 84–86, 95, 97, 98, 102, 104, 126, 130, 162, 164, 165, 169, 172, 174, 183, 202, 205, 207,**

208, 247, 252–254, 256, 257, 259,
 260
 Definition, **36**
 PMIX_INFO_TRUE
 Definition, **39**
 PMIX_INFO_XFER, 202
 Definition, **38**
 PMIx_Init, 9, 73, 76, 94, 96, 97, 140, 144,
 220, 237
 Definition, **94**
 PMIx_Initialized, 8
 Definition, **93**
 PMIX_INT, 57
 PMIX_INT16, 57
 PMIX_INT32, 57
 PMIX_INT64, 57
 PMIX_INT8, 57
 PMIX_INTERNAL, 29
 PMIX_JCTRL_CHECKPOINT, 20
 PMIX_JCTRL_CHECKPOINT_COMPLETE,
 20
 PMIX_JCTRL_PREEMPT_ALERT, 20
 PMIX_JOB_CONTINUOUS, 141, 145, 238
 Definition, **74**
 PMIx_Job_control_nb, 9, 77, 156, 166, 202
 Definition, **166**
 PMIX_JOB_CTRL_CANCEL, 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_CHECKPOINT, 168,
 259
 Definition, **78**
 PMIX_JOB_CTRL_CHECKPOINT_EVENT,
 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_CHECKPOINT_METHOD,
 169, 259
 Definition, **78**
 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL,
 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT,
 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_ID, 168, 259
 Definition, **77**
 PMIX_JOB_CTRL_KILL, 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_PAUSE, 168, 259
 Definition, **77**
 PMIX_JOB_CTRL_PREEMPTIBLE, 169,
 260
 Definition, **78**
 PMIX_JOB_CTRL_PROVISION, 169, 259
 Definition, **78**
 PMIX_JOB_CTRL_PROVISION_IMAGE,
 169, 260
 Definition, **78**
 PMIX_JOB_CTRL_RESTART, 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_RESUME, 168, 259
 Definition, **77**
 PMIX_JOB_CTRL_SIGNAL, 168, 259
 Definition, **78**
 PMIX_JOB_CTRL_TERMINATE, 168, 259
 Definition, **78**
 PMIX_JOB_INFO, 108, 112, 116, 159
 Definition, **64**
 PMIX_JOB_INFO_ARRAY, 11, 65, 205
 Definition, **65**
 PMIX_JOB_NUM_APPS, 116, 200, 205
 Definition, **66**
 PMIX_JOB_RECOVERABLE, 141, 145,
 238
 Definition, **74**
 PMIX_JOB_SIZE, 10, 110, 113, 116, 198,
 205, 206
 Definition, **66**
 PMIX_JOB_TERM_STATUS
 Definition, **69**
 PMIX_JOBID, 64, 65, 109, 112, 116, 159,
 198, 205
 Definition, **63**
 pmix_key_t, 22, 106, 108
 Definition, **22**
 PMIX_KVAL, 58
 PMIX_LOCAL, 29

PMIX_LOCAL_CPUSETS, 199, 209
 Defintion, [64](#)
 PMIX_LOCAL_PEERS, 199, 208
 Defintion, [64](#)
 PMIX_LOCAL_PROCS, 201
 Defintion, [64](#)
 PMIX_LOCAL_RANK, 160, 161, 199
 Defintion, [63](#)
 PMIX_LOCAL_SIZE, 199
 Defintion, [66](#)
 PMIX_LOCAL_TOPO
 Defintion, [67](#)
 PMIX_LOCALITY
 Defintion, [64](#)
 PMIX_LOCALITY_STRING
 Defintion, [67](#)
 PMIX_LOCALLDR, 201
 Defintion, [63](#)
 PMIX_LOG_EMAIL, 174, 254
 Defintion, [75](#)
 PMIX_LOG_EMAIL_ADDR, 174, 254
 Defintion, [75](#)
 PMIX_LOG_EMAIL_MSG, 174, 254
 Defintion, [75](#)
 PMIX_LOG_EMAIL_SUBJECT, 174, 254
 Defintion, [75](#)
 PMIX_LOG_MSG, 174, 254
 Defintion, [75](#)
 PMIx_Log_nb, 9, 75, 175
 Defintion, [173](#)
 PMIX_LOG_STDERR, 174, 254
 Defintion, [75](#)
 PMIX_LOG_STDOUT, 174, 254
 Defintion, [75](#)
 PMIX_LOG_SYSLOG, 174, 254
 Defintion, [75](#)
 PMIx_Lookup, 8, 42, 124, 130, 132
 Defintion, [128](#)
 pmix_lookup_cbfunc_t, 82, 231
 Defintion, [82](#)
 PMIx_Lookup_nb, 82
 Defintion, [130](#)
 PMIX_MAP_BLOB
 Defintion, [69](#)
 PMIX_MAPBY, 139, 144, 200, 237
 Defintion, [72](#)
 PMIX_MAPPER, 72, 139, 144, 236
 Defintion, [72](#)
 PMIX_MAX_KEYLEN, 17
 PMIX_MAX_NSLEN, 17
 PMIX_MAX_PROCS, 11, 66, 118, 198
 Defintion, [66](#)
 PMIX_MAX_RESTARTS, 141, 145, 238
 Defintion, [74](#)
 PMIX_MERGE_STDERR_STDOUT, 140,
 145, 237
 Defintion, [73](#)
 PMIX_MODEL_DECLARED, 21
 PMIX_MODEL_LIBRARY_NAME
 Defintion, [60](#)
 PMIX_MODEL_LIBRARY_VERSION
 Defintion, [61](#)
 PMIX_MODEX, 58
 pmix_modex_cbfunc_t, 79, 80, 224, 227
 Defintion, [80](#)
 PMIX_MODEX_CONSTRUCT
 Defintion, [50](#)
 PMIX_MODEX_CREATE
 Defintion, [50](#)
 pmix_modex_data_t, 49, 50
 Defintion, [49](#)
 PMIX_MODEX_DESTRUCT
 Defintion, [50](#)
 PMIX_MODEX_FREE
 Defintion, [50](#)
 PMIX_MONITOR_APP_CONTROL, 171,
 262
 Defintion, [78](#)
 PMIX_MONITOR_CANCEL, 171, 262
 Defintion, [78](#)
 PMIX_MONITOR_FILE, 171, 172, 262
 Defintion, [79](#)
 PMIX_MONITOR_FILE_ACCESS, 171,
 262
 Defintion, [79](#)
 PMIX_MONITOR_FILE_ALERT, 20

PMIX_MONITOR_FILE_CHECK_TIME,
 171, 262
 Defintion, **79**
 PMIX_MONITOR_FILE_DROPS, 171, 262
 Defintion, **79**
 PMIX_MONITOR_FILE_MODIFY, 171,
 262
 Defintion, **79**
 PMIX_MONITOR_FILE_SIZE, 171, 262
 Defintion, **79**
 PMIX_MONITOR_HEARTBEAT, 171, 262
 Defintion, **78**
 PMIX_MONITOR_HEARTBEAT_ALERT,
 20
 PMIX_MONITOR_HEARTBEAT_DROPS,
 171, 262
 Defintion, **79**
 PMIX_MONITOR_HEARTBEAT_TIME,
 171, 262
 Defintion, **79**
 PMIX_MONITOR_ID, 171, 262
 Defintion, **78**
 PMIX_NET_TOPO
 Defintion, **67**
 PMIX_NO_OVERSUBSCRIBE, 141, 145,
 238
 Defintion, **74**
 PMIX_NO_PROCS_ON_HEAD, 141, 145,
 238
 Defintion, **74**
 PMIX_NODE_INFO, 109, 112, 118, 159
 Defintion, **64**
 PMIX_NODE_INFO_ARRAY, 65, 206, 208
 Defintion, **65**
 PMIX_NODE_LIST
 Defintion, **63**
 PMIX_NODE_MAP, 64, 198, 199, 205–207
 Defintion, **69**
 PMIX_NODE_RANK, 199
 Defintion, **63**
 PMIX_NODE_SIZE, 118, 201
 Defintion, **66**
 PMIX_NODEID, 65, 109, 112, 118, 160,
 199
 Defintion, **63**
 PMIX_NON_PMI, 140, 144, 237
 Defintion, **73**
 pmix_notification_fn_t, 86, 179
 Defintion, **86**
 PMIX_NOTIFY_ALLOC_COMPLETE, 20
 PMIX_NOTIFY_COMPLETION
 Defintion, **68**
 PMIx_Notify_event, 9
 Defintion, **182**
 PMIX_NPROC_OFFSET, 200
 Defintion, **63**
 PMIX_NSDIR, 62
 Defintion, **62**
 PMIX_NSPACE, 64, 65, 109, 112, 116,
 159–161, 206
 Defintion, **63**
 pmix_nspace_t, 23, 25, 81
 Defintion, **23**
 PMIX_NUM_NODES, 106, 110, 113, 114,
 116, 117, 205, 206
 Defintion, **66**
 PMIX_NUM_SLOTS, 11
 Defintion, **66**
 pmix_op_cbfunc_t, 81, 85, 88, 127, 134,
 150, 154, 173, 182, 183, 197, 210,
 211, 213, 217, 220, 221, 223, 228,
 233, 239, 242, 244, 246, 247, 253,
 258, 261
 Defintion, **81**
 PMIX_OPERATION_SUCCEEDED, 21
 PMIX_OPTIONAL, 108, 111
 Defintion, **68**
 PMIX_OUTPUT_TO_FILE, 140, 145, 238
 Defintion, **73**
 PMIX_PARENT_ID, 138, 142, 143, 236
 Defintion, **64**
 PMIX_PDATA, 57
 PMIX_PDATA_CONSTRUCT
 Defintion, **43**
 PMIX_PDATA_CREATE
 Defintion, **43**

PMIX_PDATA_DESTRUCT
 Definition, [43](#)

PMIX_PDATA_FREE
 Definition, [44](#)

PMIX_PDATA_LOAD
 Definition, [44](#)

pmix_pdata_t, [42–45](#), [82](#), [130](#)
 Definition, [42](#)

PMIX_PDATA_XFER
 Definition, [45](#)

PMIX_PERSIST, [58](#)

PMIX_PERSIST_APP, [30](#)

PMIX_PERSIST_FIRST_READ, [30](#)

PMIX_PERSIST_INDEF, [30](#)

PMIX_PERSIST_PROC, [30](#)

PMIX_PERSIST_SESSION, [30](#)

PMIX_PERSISTENCE, [125](#), [128](#), [229](#)
 Definition, [68](#)

PMIx_Persistence_string, [9](#)
 Definition, [91](#)

pmix_persistence_t, [30](#), [58](#), [91](#)
 Definition, [30](#)

PMIX_PERSONALITY, [139](#), [143](#), [236](#)
 Definition, [72](#)

PMIX_PID, [57](#)

PMIX_POINTER, [58](#)

PMIX_PPR, [139](#), [144](#), [237](#)
 Definition, [72](#)

PMIX_PREFIX, [139](#), [143](#), [236](#)
 Definition, [72](#)

PMIX_PRELOAD_BIN, [139](#), [143](#), [236](#)
 Definition, [73](#)

PMIX_PRELOAD_FILES, [139](#), [143](#), [236](#)
 Definition, [73](#)

PMIX_PROC, [57](#)

PMIX_PROC_BLOB
 Definition, [69](#)

PMIX_PROC_CONSTRUCT, [24](#)
 Definition, [24](#), [51](#)

PMIX_PROC_CREATE
 Definition, [25](#)

PMIX_PROC_DATA, [207](#)
 Definition, [69](#)

PMIX_PROC_DESTRUCT
 Definition, [24](#)

PMIX_PROC_FREE, [157](#)
 Definition, [25](#)

PMIX_PROC_INFO, [58](#)

PMIX_PROC_INFO_CONSTRUCT
 Definition, [27](#)

PMIX_PROC_INFO_CREATE
 Definition, [28](#)

PMIX_PROC_INFO_DESTRUCT
 Definition, [28](#)

PMIX_PROC_INFO_FREE
 Definition, [28](#)

pmix_proc_info_t, [27](#), [28](#), [58](#), [74](#), [75](#), [161](#),
[251](#)
 Definition, [27](#)

PMIX_PROC_LOAD
 Definition, [25](#)

PMIX_PROC_MAP, [198](#), [205](#), [206](#)
 Definition, [69](#)

PMIX_PROC_PID
 Definition, [63](#)

PMIX_PROC_RANK, [58](#)

PMIX_PROC_STATE, [58](#)

PMIX_PROC_STATE_ABORTED, [26](#)

PMIX_PROC_STATE_ABORTED_BY_SIG,
[26](#)

PMIX_PROC_STATE_CALLED_ABORT,
[26](#)

PMIX_PROC_STATE_CANNOT_RESTART,
[26](#)

PMIX_PROC_STATE_COMM_FAILED,
[26](#)

PMIX_PROC_STATE_CONNECTED, [26](#)

PMIX_PROC_STATE_ERROR, [26](#)

PMIX_PROC_STATE_FAILED_TO_LAUNCH,
[26](#)

PMIX_PROC_STATE_FAILED_TO_START,
[26](#)

PMIX_PROC_STATE_KILLED_BY_CMD,
[26](#)

PMIX_PROC_STATE_LAUNCH_UNDERWAY,
[26](#)

PMIX_PROC_STATE_MIGRATING, 26
 PMIX_PROC_STATE_PREPPED, 26
 PMIX_PROC_STATE_RESTART, 26
 PMIX_PROC_STATE_RUNNING, 26
 PMIX_PROC_STATE_STATUS
 Defintion, 69
 PMIx_Proc_state_string, 9
 Defintion, 91
 pmix_proc_state_t, 26, 58, 91
 Defintion, 26
 PMIX_PROC_STATE_TERM_NON_ZERO, 26
 PMIX_PROC_STATE_TERM_WO_SYNC, 26
 PMIX_PROC_STATE_TERMINATE, 26
 PMIX_PROC_STATE_TERMINATED, 26
 PMIX_PROC_STATE_UNDEF, 26
 PMIX_PROC_STATE_UNTERMINATED, 26
 pmix_proc_t, 23–25, 44, 57, 64, 71, 86, 90, 96, 98, 100, 110, 120–122, 136, 180, 183, 184, 189, 190, 201, 211, 213, 214, 220, 221, 223, 224, 227, 228, 231, 233, 235, 239, 242, 247, 250, 253, 255, 258, 260
 Defintion, 24
 PMIX_PROC_TERMINATED, 20
 PMIX_PROC_URI, 162
 Defintion, 64
 PMIX_PROCDIR
 Defintion, 62
 PMIx_Process_monitor_nb, 9, 78, 156, 172
 Defintion, 170
 PMIX_PROCID, 160, 161, 201
 Defintion, 63
 PMIX_PROGRAMMING_MODEL
 Defintion, 60
 PMIx_Publish, 8, 29, 30, 68, 125, 126, 128, 229, 230
 Defintion, 124
 PMIx_Publish_nb, 8, 128
 Defintion, 126
 PMIx_Put, 8, 29, 32, 89, 107, 110, 113, 119, 121, 146, 163, 215, 228
 Defintion, 106
 PMIX_QUERY, 58
 PMIX_QUERY_ALLOC_STATUS, 162, 251
 Defintion, 75
 PMIX_QUERY_AUTHORIZATIONS
 Defintion, 75
 PMIX_QUERY_CONSTRUCT
 Defintion, 48
 PMIX_QUERY_CREATE
 Defintion, 48
 PMIX_QUERY_DEBUG_SUPPORT, 161, 251
 Defintion, 75
 PMIX_QUERY_DESTRUCT
 Defintion, 48
 PMIX_QUERY_FREE
 Defintion, 48
 PMIx_Query_info_nb, 9, 10, 47, 66, 74, 118, 146, 156, 162, 163
 Defintion, 158
 PMIX_QUERY_JOB_STATUS, 161, 250
 Defintion, 74
 PMIX_QUERY_LOCAL_ONLY, 251
 Defintion, 75
 PMIX_QUERY_LOCAL_PROC_TABLE, 161, 251
 Defintion, 74
 PMIX_QUERY_MEMORY_USAGE, 161, 251
 Defintion, 75
 PMIX_QUERY_NAMESPACES, 161, 250
 Defintion, 74
 PMIX_QUERY_PARTIAL_SUCCESS, 20
 PMIX_QUERY_PROC_TABLE, 161, 251
 Defintion, 74
 PMIX_QUERY_QUALIFIERS_CREATE, 10
 Defintion, 49
 PMIX_QUERY_QUEUE_LIST, 161, 250
 Defintion, 74
 PMIX_QUERY_QUEUE_STATUS, 161,

250
 Definition, [74](#)
 PMIX_QUERY_REFRESH_CACHE, [159](#),
 [162](#), [163](#)
 Definition, [74](#)
 PMIX_QUERY_REPORT_AVG, [161](#), [251](#)
 Definition, [75](#)
 PMIX_QUERY_REPORT_MINMAX, [161](#),
 [251](#)
 Definition, [75](#)
 PMIX_QUERY_SPAWN_SUPPORT, [161](#),
 [251](#)
 Definition, [75](#)
 pmix_query_t, [10](#), [47–49](#), [160](#), [161](#), [163](#),
 [250](#), [251](#)
 Definition, [47](#)
 PMIX_RANGE, [125](#), [128](#), [129](#), [131](#), [133](#),
 [135](#), [180](#), [229](#), [231](#), [234](#), [248](#)
 Definition, [68](#)
 PMIX_RANGE_CUSTOM, [29](#)
 PMIX_RANGE_GLOBAL, [29](#)
 PMIX_RANGE_LOCAL, [29](#)
 PMIX_RANGE_NAMESPACE, [29](#)
 PMIX_RANGE_PROC_LOCAL, [29](#)
 PMIX_RANGE_RM, [29](#)
 PMIX_RANGE_SESSION, [29](#)
 PMIX_RANGE_UNDEF, [29](#)
 PMIX_RANK, [160](#), [161](#), [199](#)
 Definition, [63](#)
 PMIX_RANK_INVALID, [24](#)
 PMIX_RANK_LOCAL_NODE, [23](#)
 PMIX_RANK_LOCAL_PEERS, [24](#)
 pmix_rank_t, [23–25](#), [58](#)
 Definition, [23](#)
 PMIX_RANK_UNDEF, [23](#)
 PMIX_RANK_VALID, [24](#)
 PMIX_RANK_WILDCARD, [23](#)
 PMIX_RANKBY, [140](#), [144](#), [200](#), [237](#)
 Definition, [72](#)
 PMIx_Register_event_handler, [9](#), [85](#), [156](#)
 Definition, [178](#)
 PMIX_REGISTER_NODATA, [198](#)
 Definition, [59](#), [69](#)
 pmix_release_cbfunc_t, [79](#)
 Definition, [79](#)
 PMIX_REMOTE, [29](#)
 PMIX_REPORT_BINDINGS, [141](#), [145](#), [238](#)
 Definition, [74](#)
 PMIX_REQUESTOR_IS_CLIENT, [138](#),
 [143](#)
 Definition, [61](#)
 PMIX_REQUESTOR_IS_TOOL, [138](#), [143](#)
 Definition, [61](#)
 PMIx_Resolve_nodes, [8](#)
 Definition, [157](#)
 PMIx_Resolve_peers, [8](#)
 Definition, [157](#)
 PMIX_RM_NAME
 Definition, [76](#)
 PMIX_RM_VERSION
 Definition, [76](#)
 PMIX_SCOPE, [58](#)
 PMIx_Scope_string, [9](#)
 Definition, [91](#)
 pmix_scope_t, [29](#), [58](#), [91](#), [107](#)
 Definition, [29](#)
 PMIX_SCOPE_UNDEF, [29](#)
 PMIX_SEND_HEARTBEAT
 Definition, [78](#)
 pmix_server_abort_fn_t
 Definition, [222](#)
 pmix_server_alloc_fn_t
 Definition, [255](#)
 pmix_server_client_connected_fn_t, [82](#), [212](#),
 [220](#)
 Definition, [219](#)
 pmix_server_client_finalized_fn_t, [222](#)
 Definition, [221](#)
 pmix_server_connect_fn_t, [241](#), [243](#)
 Definition, [239](#)
 PMIx_server_deregister_client, [8](#)
 Definition, [212](#)
 pmix_server_deregister_events_fn_t
 Definition, [245](#)
 PMIx_server_deregister_nspace, [8](#), [213](#)
 Definition, [210](#)

pmix_server_disconnect_fn_t, 243
 Defintion, 241
 pmix_server_dmodex_req_fn_t, 10, 80
 Defintion, 226
 PMIx_server_dmodex_request, 9, 88, 89,
 215
 Defintion, 214
 PMIX_SERVER_ENABLE_MONITORING
 Defintion, 59
 pmix_server_fencenb_fn_t, 80, 226
 Defintion, 223
 PMIx_server_finalize, 8
 Defintion, 104
 PMIX_SERVER_HOSTNAME
 Defintion, 60
 PMIx_server_init, 8, 94, 218
 Defintion, 102
 pmix_server_job_control_fn_t
 Defintion, 257
 pmix_server_listener_fn_t
 Defintion, 248
 pmix_server_log_fn_t
 Defintion, 253
 pmix_server_lookup_fn_t
 Defintion, 230
 pmix_server_module_t, 102, 104, 218
 Defintion, 218
 pmix_server_monitor_fn_t
 Defintion, 260
 pmix_server_notify_event_fn_t, 87
 Defintion, 247
 PMIX_SERVER_NAMESPACE, 102, 200
 Defintion, 59
 PMIX_SERVER_PIDINFO, 99, 100
 Defintion, 60
 pmix_server_publish_fn_t
 Defintion, 228
 pmix_server_query_fn_t
 Defintion, 249
 PMIX_SERVER_RANK, 102, 200
 Defintion, 59
 PMIx_server_register_client, 8, 212, 220,
 222
 Defintion, 211
 pmix_server_register_events_fn_t
 Defintion, 243
 PMIx_server_register_nspace, 8, 10, 15, 65,
 82, 202, 205
 Defintion, 197
 PMIX_SERVER_REMOTE_CONNECTIONS,
 104
 Defintion, 59
 PMIx_server_setup_application, 9, 87, 88,
 218
 Defintion, 215
 PMIx_server_setup_fork, 9
 Defintion, 213
 PMIx_server_setup_local_support, 9
 Defintion, 217
 pmix_server_spawn_fn_t, 81
 Defintion, 234
 PMIX_SERVER_SYSTEM_SUPPORT, 103
 Defintion, 59
 PMIX_SERVER_TMPDIR, 102
 Defintion, 59
 pmix_server_tool_connection_fn_t
 Defintion, 252
 PMIX_SERVER_TOOL_SUPPORT, 102
 Defintion, 59
 pmix_server_unpublish_fn_t
 Defintion, 232
 PMIX_SERVER_URI, 98, 100, 162
 Defintion, 60
 PMIX_SESSION_ID, 64, 65, 108, 112, 115,
 159, 200, 205
 Defintion, 63
 PMIX_SESSION_INFO, 108, 112, 114, 159
 Defintion, 64
 PMIX_SESSION_INFO_ARRAY, 11, 65,
 66, 198, 205
 Defintion, 65
 PMIX_SET_ENVAR
 Defintion, 76
 PMIX_SET_SESSION_CWD, 139, 143, 236
 Defintion, 73
 pmix_setup_application_cbfunc_t, 216

Defintion, [87](#)
 PMIX_SINGLE_LISTENER, [95](#)
 Defintion, [61](#)
 PMIX_SIZE, [57](#)
 PMIX_SOCKET_MODE, [95, 99, 103](#)
 Defintion, [61](#)
 PMIx_Spawn, [8, 45, 62, 72, 76, 137, 138, 142, 143, 146, 202, 214, 234, 239](#)
 Defintion, [137](#)
 pmix_spawn_cbfunc_t, [81, 142, 235](#)
 Defintion, [81](#)
 PMIx_Spawn_nb, [8, 45, 81](#)
 Defintion, [142](#)
 PMIX_SPAWNED, [138, 142, 143, 236](#)
 Defintion, [62](#)
 PMIX_STATUS, [58](#)
 pmix_status_t, [18, 22, 35, 58, 84–86, 88–91, 179, 183, 244, 246, 247](#)
 Defintion, [18](#)
 PMIX_STDIN_TGT, [140, 144, 237](#)
 Defintion, [73](#)
 PMIx_Store_internal, [9](#)
 Defintion, [113](#)
 PMIX_STRING, [57](#)
 PMIX_SUCCESS, [19](#)
 pmix_system_event
 Defintion, [21](#)
 PMIX_SYSTEM_TMPDIR, [102](#)
 Defintion, [59](#)
 PMIX_TAG_OUTPUT, [140, 144, 237](#)
 Defintion, [73](#)
 PMIX_TCP_DISABLE_IPV4, [96, 100, 103](#)
 Defintion, [62](#)
 PMIX_TCP_DISABLE_IPV6, [96, 100, 103](#)
 Defintion, [62](#)
 PMIX_TCP_IF_EXCLUDE, [95, 99, 103](#)
 Defintion, [61](#)
 PMIX_TCP_IF_INCLUDE, [95, 99, 103](#)
 Defintion, [61](#)
 PMIX_TCP_IPV4_PORT, [96, 99, 103](#)
 Defintion, [61](#)
 PMIX_TCP_IPV6_PORT, [96, 99, 103](#)
 Defintion, [62](#)
 PMIX_TCP_REPORT_URI, [95, 99, 103](#)
 Defintion, [61](#)
 PMIX_TCP_URI, [99, 100](#)
 Defintion, [61](#)
 PMIX_TDIR_RMCLEAN
 Defintion, [62](#)
 PMIX_THREADING_MODEL
 Defintion, [61](#)
 PMIX_TIME, [57](#)
 PMIX_TIME_REMAINING, [156, 162, 251](#)
 Defintion, [75](#)
 PMIX_TIMEOUT, [3, 12, 109, 110, 112, 113, 120, 121, 123, 125, 128–133, 135, 148, 151, 152, 154, 155, 225, 227, 229, 232, 234, 238, 240, 243](#)
 Defintion, [68](#)
 PMIX_TIMESTAMP_OUTPUT, [140, 144, 237](#)
 Defintion, [73](#)
 PMIX_TIMEVAL, [57](#)
 PMIX_TMPDIR, [62](#)
 Defintion, [62](#)
 pmix_tool_connection_cbfunc_t, [252](#)
 Defintion, [90](#)
 PMIX_TOOL_DO_NOT_CONNECT, [98, 100](#)
 Defintion, [60](#)
 PMIx_tool_finalize, [9](#)
 Defintion, [101](#)
 PMIx_tool_init, [9, 60, 94, 101](#)
 Defintion, [98](#)
 PMIX_TOOL_NAMESPACE, [98](#)
 Defintion, [60](#)
 PMIX_TOOL_RANK, [98](#)
 Defintion, [60](#)
 PMIX_TOPOLOGY
 Defintion, [67](#)
 PMIX_TOPOLOGY_SIGNATURE
 Defintion, [67](#)
 PMIX_UINT, [57](#)
 PMIX_UINT16, [57](#)
 PMIX_UINT32, [57](#)
 PMIX_UINT64, [57](#)

PMIX_UINT8, [57](#)
 PMIX_UNDEF, [57](#)
 PMIX_UNIV_SIZE, [10](#), [11](#), [110](#), [113](#), [114](#),
 [198](#), [205](#)
 Definition, [66](#)
 PMix_Unpublish, [8](#), [134](#), [135](#)
 Definition, [132](#)
 PMix_Unpublish_nb, [8](#)
 Definition, [134](#)
 PMIX_UNSET_ENVAR
 Definition, [76](#)
 PMIX_USERID, [125](#), [127](#), [129](#), [131](#), [133](#),
 [135](#), [160](#), [165](#), [168](#), [171](#), [174](#),
 [229–235](#), [244](#), [250](#), [252](#), [254](#), [256](#),
 [258](#), [261](#)
 Definition, [60](#)
 PMIX_USOCK_DISABLE, [95](#), [103](#)
 Definition, [61](#)
 PMIX_VALUE, [57](#)
 pmix_value_cbfunc_t, [83](#)
 Definition, [83](#)
 PMIX_VALUE_CONSTRUCT
 Definition, [33](#)
 PMIX_VALUE_CREATE
 Definition, [33](#)
 PMIX_VALUE_DESTRUCT
 Definition, [33](#)
 PMIX_VALUE_FREE
 Definition, [34](#)

PMIX_VALUE_LOAD
 Definition, [34](#)
 pmix_value_t, [32–36](#), [57](#), [83](#), [106](#), [107](#)
 Definition, [32](#)
 PMIX_VALUE_UNLOAD
 Definition, [35](#)
 PMIX_VALUE_XFER
 Definition, [35](#)
 PMIX_VERSION_INFO
 Definition, [60](#)
 PMIX_WAIT, [129–131](#), [231](#)
 Definition, [68](#)
 PMIX_WDIR, [138](#), [143](#), [236](#)
 Definition, [72](#)

rank, [116](#), [207](#)
 Definition, [13](#)
 resource manager
 Definition, [14](#)

session, [10](#), [11](#), [64](#), [65](#), [106](#), [114](#), [163](#), [202](#)
 Definition, [13](#)

slot
 Definition, [13](#)

slots
 Definition, [13](#)

workflow
 Definition, [13](#)