

# Process Management Interface for Exascale (PMIx) Standard

Version 2.1 (Draft)

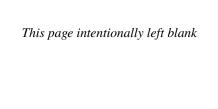
December 2018

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 2.1 (Draft).

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

Copyright © 2018 PMIx Standard Review Board.

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



# **Contents**

1.	Intro	duction		1
	1.1.	Charter		2
	1.2.	PMIx St	andard 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 Aı	rchitecture Overview	5
		1.3.1.	The PMIx Reference Implementation (PRI)	6
		1.3.2.	The PMIx Reference RunTime Environment (PRRTE)	7
	1.4.	Organiza	ation 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
2.	PMIx	Terms a	and Conventions	11
	2.1.	Notation	al Conventions	13
	2.2.	Semantio	cs	14
	2.3.	Naming	Conventions	14
	2.4.	Procedu	re Conventions	15
	2.5.	Standard	l vs Reference Implementation	15
3.	Data	Structu	res and Types	16
	3.1.	Constant	ts	16
		3.1.1.	Error Constants	17
	3.2.	Data Typ	oes	20
		3.2.1.	Key Structure	20
		3.2.2.	Namespace Structure	21
		3 2 3	Rank Structure	22

	3.2.4.	Process Structure	22
	3.2.5.	Process structure support macros	22
	3.2.6.	Process State Structure	24
	3.2.7.	Process Information Structure	25
	3.2.8.	Process Information Structure support macros	26
	3.2.9.	Scope of Put Data	27
	3.2.10.	Range of Published Data	28
	3.2.11.	Data Persistence Structure	28
	3.2.12.	Value Structure	29
	3.2.13.	Value structure support macros	30
	3.2.14.	Load a pmix_value_t structure	31
	3.2.15.	Info and Info Array Structures	32
	3.2.16.	Info structure support macros	33
	3.2.17.	Info Type Directives	36
	3.2.18.	Info Directive support macros	36
	3.2.19.	Job Allocation Directives	37
	3.2.20.	Lookup Returned Data Structure	38
	3.2.21.	Lookup data structure support macros	38
	3.2.22.	Application Structure	41
	3.2.23.	App structure support macros	41
	3.2.24.	Query Structure	43
	3.2.25.	Query structure support macros	43
	3.2.26.	Modex Structure	44
	3.2.27.	Modex data structure support macros	45
3.3.	Data Pa	cking/Unpacking Types and Structures	46
	3.3.1.	Byte Object Type	46
	3.3.2.	Byte object support macros	46
	3.3.3.	Data Buffer Type	48
	3.3.4.	Data buffer support macros	48
	3.3.5.	Data Array Structure	50
	3.3.6.	Generalized Data Types Used for Packing/Unpacking	50
3.4.	Reserve	d attributes	52
	3 4 1	Initialization attributes	52

	3.4.2.	Tool-related attributes	53
	3.4.3.	Identification attributes	53
	3.4.4.	UNIX socket rendezvous socket attributes	54
	3.4.5.	TCP connection attributes	54
	3.4.6.	Global Data Storage (GDS) attributes	55
	3.4.7.	General process-level attributes	55
	3.4.8.	Scratch directory attributes	56
	3.4.9.	Relative Rank Descriptive Attributes	56
	3.4.10.	Information retrieval attributes	57
	3.4.11.	Information storage attributes	58
	3.4.12.	Size information attributes	58
	3.4.13.	Memory information attributes	59
	3.4.14.	Topology information attributes	59
	3.4.15.	Request-related attributes	60
	3.4.16.	Server-to-PMIx library attributes	61
	3.4.17.	Server-to-Client attributes	62
	3.4.18.	Event handler registration and notification attributes	62
	3.4.19.	Fault tolerance attributes	64
	3.4.20.	Spawn attributes	64
	3.4.21.	Query attributes	66
	3.4.22.	Log attributes	67
	3.4.23.	Debugger attributes	68
	3.4.24.	Resource manager attributes	68
	3.4.25.	Environment variable attributes	69
	3.4.26.	Job Allocation attributes	69
	3.4.27.	Job control attributes	70
	3.4.28.	Monitoring attributes	71
3.5.	Callback	K Functions	71
	3.5.1.	Release Callback Function	72
	3.5.2.	Modex Callback Function	72
	3.5.3.	Spawn Callback Function	73
	3.5.4.	Op Callback Function	74
	355	Lookun Callback Function	74

		3.5.6.	Value Callback Function	75
		3.5.7.	Info Callback Function	75
		3.5.8.	Event Handler Registration Callback Function	76
		3.5.9.	Notification Handler Completion Callback Function	77
		3.5.10.	Notification Function	78
		3.5.11.	Server Setup Application Callback Function	80
		3.5.12.	Server Direct Modex Response Callback Function	81
		3.5.13.	pmix_connection_cbfunc_t	82
		3.5.14.	<pre>pmix_tool_connection_cbfunc_t</pre>	83
		3.5.15.	Constant String Functions	83
1	Initia	lization	and Finalization	86
٦.	4.1.			86
	т.1.	4.1.1.	PMIx_Initialized	86
		4.1.2.	PMIx Get version	87
	4.2.		nitialization and Finalization	87
	1.2.	4.2.1.	PMIx Init	87
		4.2.2.	PMIx Finalize	90
	4.3.		tialization and Finalization	91
	1.5.	4.3.1.	PMIx_tool_init	91
		4.3.2.	PMIx tool finalize	94
	4.4.		nitialization and Finalization	95
		4.4.1.	PMIx_server_init	95
		4.4.2.	PMIx_server_finalize	97
5.			anagement	99
	5.1.	Č	and Accessing Key/Value Pairs	99
		5.1.1.	PMIx_Put	99
		5.1.2.	PMIx_Get	100
		5.1.3.	PMIx_Get_nb	103
		5.1.4.	PMIx_Store_internal	106
		5.1.5.	Accessing information: examples	107
	5.2.		ging Key/Value Pairs	112
		5.2.1.	PMIx_Commit	112

		5.2.2.	PMIx_Fence	112
		5.2.3.	PMIx_Fence_nb	114
	5.3.	Publish	and Lookup Data	117
		5.3.1.	PMIx_Publish	117
		5.3.2.	PMIx_Publish_nb	119
		5.3.3.	PMIx_Lookup	121
		5.3.4.	PMIx_Lookup_nb	123
		5.3.5.	PMIx_Unpublish	125
		5.3.6.	PMIx_Unpublish_nb	127
	Duan	M		129
ο.			nagement	
	6.1.			129
	6.0	6.1.1.	PMIx_Abort	129
	6.2.		Creation	130
		6.2.1.	PMIx_Spawn	130
		6.2.2.	PMIx_Spawn_nb	135
	6.3.		ing and Disconnecting Processes	139
		6.3.1.	PMIx_Connect	139
		6.3.2.	PMIx_Connect_nb	142
		6.3.3.	PMIx_Disconnect	144
		6.3.4.	PMIx_Disconnect_nb	146
7.	Job .	Allocation	on Management and Reporting	148
	7.1.			148
		7.1.1.	PMIx_Resolve_peers	149
		7.1.2.	PMIx_Resolve_nodes	149
		7.1.3.	PMIx_Query_info_nb	150
	7.2.	Allocati	on Requests	154
		7.2.1.	PMIx_Allocation_request_nb	155
		7.2.2.	PMIx_Job_control_nb	157
	7.3.	Process	and Job Monitoring	160
		7.3.1.	PMIx_Process_monitor_nb	160
		7.3.2.	PMIx_Heartbeat	163

	7.4.	Logging	3	163
		7.4.1.	PMIx_Log_nb	163
8.	Ever	nt Notific	cation	166
	8.1.	Notifica	tion and Management	166
		8.1.1.	PMIx_Register_event_handler	168
		8.1.2.	PMIx_Deregister_event_handler	171
		8.1.3.	PMIx_Notify_event	172
9.	Data	Packing	g and Unpacking	175
	9.1.	Support	Macros	175
		9.1.1.	PMIX_DATA_BUFFER_CREATE	175
		9.1.2.	PMIX_DATA_BUFFER_RELEASE	176
		9.1.3.	PMIX_DATA_BUFFER_CONSTRUCT	176
		9.1.4.	PMIX_DATA_BUFFER_DESTRUCT	176
		9.1.5.	PMIX_DATA_BUFFER_LOAD	177
		9.1.6.	PMIX_DATA_BUFFER_UNLOAD	177
	9.2.	General	Routines	178
		9.2.1.	PMIx_Data_pack	178
		9.2.2.	PMIx_Data_unpack	180
		9.2.3.	PMIx_Data_copy	182
		9.2.4.	PMIx_Data_print	182
		9.2.5.	PMIx_Data_copy_payload	183
10	.Serv	er-Spec	ific Interfaces	185
	10.1.	Server S	Support Functions	185
		10.1.1.	PMIx_generate_regex	185
		10.1.2.	PMIx_generate_ppn	186
		10.1.3.	PMIx_server_register_nspace	187
		10.1.4.	PMIx_server_deregister_nspace	199
		10.1.5.	PMIx_server_register_client	200
		10.1.6.	PMIx_server_deregister_client	202
		10.1.7.	PMIx_server_setup_fork	203
		10.1.8.	PMIx_server_dmodex_request	203
		10 1 9	PMTx server setup application	205

		10.1.10. PMIx_server_setup_local_support	206
	10.2.	Server Function Pointers	208
		10.2.1. pmix_server_module_t Module	208
		10.2.2. pmix_server_client_connected_fn_t	209
		10.2.3. pmix_server_client_finalized_fn_t	211
		10.2.4. pmix_server_abort_fn_t	212
		10.2.5. pmix_server_fencenb_fn_t	213
		10.2.6. pmix_server_dmodex_req_fn_t	216
		10.2.7. pmix_server_publish_fn_t	217
		10.2.8. pmix_server_lookup_fn_t	219
		10.2.9. pmix_server_unpublish_fn_t	221
		10.2.10. pmix_server_spawn_fn_t	223
		10.2.11. pmix_server_connect_fn_t	228
		10.2.12. pmix_server_disconnect_fn_t	230
		10.2.13. pmix_server_register_events_fn_t	232
		$10.2.14.~{\tt pmix\_server\_deregister\_events\_fn\_t}~\dots~\dots~\dots$	234
		10.2.15. pmix_server_notify_event_fn_t	236
		10.2.16. pmix_server_listener_fn_t	237
		10.2.17. pmix_server_query_fn_t	238
		10.2.18. pmix_server_tool_connection_fn_t	241
		10.2.19. pmix_server_log_fn_t	242
		10.2.20. pmix_server_alloc_fn_t	244
		10.2.21. pmix_server_job_control_fn_t	246
		10.2.22. pmix_server_monitor_fn_t	249
Δ.	Ackr	nowledgements	252
		Version 2.0	252
		Version 1.0	253
	11.2.	TOTOLOGI I.O	233
Bik	oliogr	raphy	255
Ind	lex		256

#### **CHAPTER 1**

# Introduction

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

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

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

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

- 1 c) establish a collaboration between SMS subsystem providers including resource manager, fabric,
  2 file system, and programming library developers to define integration points between the
  3 various subsystems as well as agreed upon definitions for associated APIs, attribute names, and
  4 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

5

6

7

8

13

14

15

16 17

18

19

20

21

- 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.

#### 2 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?

The PMIx Standard informs PMIx clients and RMs of the syntax and semantics of the PMIx APIs.

PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to understand the set of attributes provided by various APIs of the PRI and their intended behavior. Additional information about the rationale for the selection of specific interfaces and attributes is also provided.

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

## 1.2.2 What is defined in the standard?

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

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

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

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

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

If an application (for example) truly relies on the **PMIX\_TIMEOUT** attribute in a call to **PMIx\_Fence**, it should set the required flag in the **pmix\_info\_t** for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not

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

It is therefore critical that users and application implementers:

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

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

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

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

## 1.2.4 General Guidance for PMIx Users and Implementors

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

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

PMIx-enabled RMs may choose to implement a subset of the PMIx standard and/or define attributes beyond those defined herein. PMIx-enabled RMs are strongly advised to define a document itemizing the PMIx interfaces and associated attributes they support, with any annotations about behavior limitations. The PMIx standard cannot define this list for all given PMIx-enabled RMs, 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

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

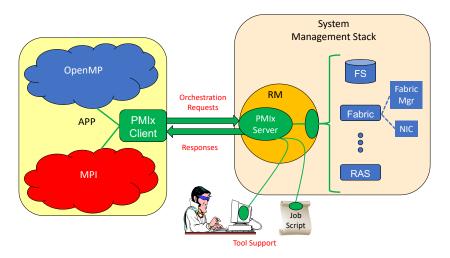


Figure 1.1.: PMIx-SMS Interactions

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

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

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

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

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

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

## 1.3.1 The PMIx Reference Implementation (PRI)

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

Validation of the standard.
 No proposed change and/or extension to the PMIx standard is accepted without an accompanying prototype implementation in the PRI. This ensures that the proposal has undergone at least some 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 <a href="https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/">https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/</a>

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 <a href="https://pmix.org/code/getting-the-reference-implementation/">https://pmix.org/code/getting-the-reference-implementation/</a>. 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.

## 20 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 <a href="https://pmix.org/code/getting-the-pmix-reference-server/">https://pmix.org/code/getting-the-pmix-reference-server/</a>

# 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 11
- Data Structures and Types in Chapter 3 on page 16

- PMIx Initialization and Finalization in Chapter 4 on page 86
- Key/Value Management in Chapter 5 on page 99
- Process Management in Chapter 6 on page 129
  - Job Management in Chapter 7 on page 148
  - Event Notification in Chapter 8 on page 166
    - Data Packing and Unpacking in Chapter 9 on page 175
  - PMIx Server Specific Interfaces in Chapter 10 on page 185

# ₃ 1.5 Version 1.0: June 12, 2015

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

5

6 7

- 13 PMIx\_Init, PMIx\_Initialized, PMIx\_Abort, PMIx\_Finalize
- 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
  - PMIx Disconnect PMIx Disconnect nb
- 23 PMIx\_Resolve\_nodes, PMIx\_Resolve\_peers
- Server APIs
- 25 PMIx\_server\_init, PMIx\_server\_finalize
- 26 PMIx\_generate\_regex, PMIx\_generate\_ppn
- 27 PMIx\_server\_register\_nspace, PMIx\_server\_deregister\_nspace
- 28 PMIx server register client, PMIx server deregister client

The **PMIx Init** API was subsequently modified in the PRI release v1.1.0.

# 7 1.6 Version 2.0: Sept. 2018

6

27

28

29

The following APIs were introduced in v2.0 of the PMIx Standard: 8 • Client APIs 9 10 - PMIx\_Query\_info\_nb, PMIx\_Log\_nb 11 - PMIx\_Allocation\_request\_nb, PMIx\_Job\_control\_nb, PMIx Process monitor nb. PMIx Heartbeat 12 Server APIs 13 14 - PMIx\_server\_setup\_application, PMIx\_server\_setup\_local\_support • Tool APIs 15 16 - PMIx tool init, PMIx tool finalize Common APIs 17 18 - PMIx Register event handler, PMIx Deregister event handler 19 - PMIx Notify event 20 - PMIx Proc state string, PMIx Scope string - PMIx\_Persistence\_string, PMIx\_Data\_range\_string 21 - PMIx\_Info\_directives\_string, PMIx\_Data\_type\_string 22 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

include passing of a pmix\_info\_t array for flexibility and "future-proofing" of the API. In

addition, the PMIx Notify error, PMIx Register errhandler, and

PMIx Deregister errhandler APIs were replaced.

### 1 1.7 Version 2.1: Dec. 2018

2 The v2.1 update includes clarifications and corrections, plus addition of examples: • Clarify description of **PMIx\_Connect** and **PMIx\_Disconnect** APIs. • Explain that values for the PMIX\_COLLECTIVE\_ALGO are environment-dependent 4 • Identify the namespace/rank values required for retrieving attribute-associated information using 5 6 the **PMIx Get** API 7 • Provide definitions for **session**, **job**, **application**, and other terms used throughout the document • Clarify definitions of PMIX UNIV SIZE versus PMIX JOB SIZE 9 • Clarify server module function return values 10 • Provide examples of the use of **PMIx Get** for retrieval of information 11 12 • Clarify the use of PMIx Get versus PMIx Query info nb 13 • Clarify return values for non-blocking APIs and emphasize that callback functions must not be 14 invoked prior to return from the API Provide detailed example for construction of the PMIx\_server\_register\_nspace input 15 information array 16 17 • Define information levels (e.g., **session** vs **job**) and associated attributes for both storing and retrieving values 18 Clarify roles of PMIx server library and host environment for collective operations 19 • Clarify definition of PMIX UNIV SIZE 20

#### **CHAPTER 2**

# **PMIx Terms and Conventions**

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

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

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

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

#### Advice to users -

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

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

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

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

## 10 2.1 Notational Conventions

4

5

6

7

8

12	that applies only to programs for which the base language is C is shown as follows:
	C
13	C specific text
14	<pre>int foo = 42;</pre>
	C —
15	Some text is for information only, and is not part of the normative specification. These take several
16	forms, described in their examples below:
	<b>▼</b>
17	Note: General text
	<u> </u>
	▼
18	Throughout this document, the rationale for the design choices made in the interface specification is
19	set off in this section. Some readers may wish to skip these sections, while readers interested in
20	interface design may want to read them carefully.
	Advice to users
21	Throughout this document, material aimed at users and that illustrates usage is set off in this
22	section. Some readers may wish to skip these sections, while readers interested in programming with the PMIx API may want to read them carefully.

#### Advice to PMIx library implementers —

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

#### Advice to PMIx server hosts -

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

## 2.2 Semantics

1

2

3

4

5

6

7

9 10

11

12

15 16

17

18

19 20

21

22

23

24

25

26

The following terms will be taken to mean:

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

# 3 2.3 Naming Conventions

- The PMIx standard has adopted the following conventions:
  - PMIx constants and attributes are prefixed with **PMIX**.
  - Structures and type definitions are prefixed with pmix.
  - Underscores are used to separate words in a function or variable name.
  - Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the first letter of the word following it. For example, PMIx\_Get\_version.
  - PMIx server and tool APIs are all lower case letters following the prefix e.g.,
     PMIx\_server\_register\_nspace.
  - The **PMIx** prefix is used to denote functions.
  - The **pmix** prefix is used to denote function pointer and type definitions.
  - Users should not use the **PMIX**, **PMIX**, or **pmix** prefixes in their applications or libraries so as to avoid symbol conflicts with current and later versions of the PMIx standard and implementations such as the PRI.

#### 1 2.4 Procedure Conventions

7

8

9

10

12

13

14

15

16

17

- While the current PMIx Reference Implementation (PRI) is solely based on the C programming language, it is not the intent of the PMIx Standard to preclude the use of other languages.

  Accordingly, the procedure specifications in the PMIx Standard are written in a language-independent syntax with the arguments marked as IN, OUT, or INOUT. The meanings of these are:
  - IN: The call may use the input value but does not update the argument from the perspective of the caller at any time during the calls execution,
  - OUT: The call may update the argument but does not use its input value
  - INOUT: The call may both use and update the argument.

# 2.5 Standard vs Reference Implementation

- The *PMIx Standard* is implementation independent. The *PMIx Reference Implementation* (PRI) is one implementation of the Standard and the PMIx community strives to ensure that it fully implements the Standard. Given its role as the community's testbed and its widespread use, this document cites the attributes supported by the PRI for each API where relevant by marking them in red. This is not meant to imply nor confer any special role to the PRI with respect to the Standard itself, but instead to provide a convenience to users of the Standard and PRI.
- Similarly, the *PMIx Reference RunTime Environment* (PRRTE) is provided by the community to enable users operating in non-PMIx environments to develop and execute PMIx-enabled applications and tools. Attributes supported by the PRRTE are marked in green.

#### **CHAPTER 3**

# **Data Structures and Types**

This chapter defines PMIx standard data structures, types, and constants. These apply to all 1 consumers of the PMIx interface. Where necessary for clarification, the description of, for 2 3 example, an attribute may be copied from this chapter into a section where it is used. A PMIx implementation may define additional attributes beyond those specified in this document. 4 Advice to PMIx library implementers – 5 Structures, types, and macros in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent 6 7 definitions in a language-appropriate manner. 8 If a PMIx implementation chooses to define additional attributes they should avoid using the PMIX prefix in their name or starting the attribute string with a pmix prefix. This helps the end user 9 10 distinguish between what is defined by the PMIx standard and what is specific to that PMIx implementation, and avoids potential conflicts with attributes defined by the standard. 11

## 12 3.1 Constants

13

14

15

16 17

18

19

20 21

22

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

PMIX\_MAX\_NSLEN

Maximum namespace string length as an integer.

Advice to PMIx library implementers

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

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

<ul> <li>Advice to PMIx library implemente</li> </ul>	Advice to	<b>PMIx</b>	library im	plementer
---	-----------	-------------	------------	-----------

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

#### 4 3.1.1 Error Constants

1

2

8 9

10

11 12

13

- 5 The **pmix\_status\_t** structure is an **int** type for return status.
- The tables shown in this section define the possible values for **pmix\_status\_t**. PMIx errors are required to always be negative, with 0 reserved for **PMIX\_SUCCESS**.
  - A PMIx implementation must define all of the constants defined in this section, even if they will never return the specific value to the caller.

#### Advice to users

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

#### 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
                                 Success
              PMIX_SUCCESS
 6
              PMIX ERROR
                              General Error
 7
              PMIX ERR SILENT
                                     Silent error
8
                                                 Error in debugger release
              PMIX ERR DEBUGGER RELEASE
9
                                            Fault tolerance: Error in process restart
              PMIX ERR PROC RESTART
10
              PMIX_ERR_PROC_CHECKPOINT
                                                Fault tolerance: Error in process checkpoint
              PMIX_ERR_PROC_MIGRATE
                                            Fault tolerance: Error in process migration
11
12
              PMIX ERR PROC ABORTED
                                            Process was aborted
                                                      Process is already requested to abort
13
              PMIX_ERR_PROC_REQUESTED_ABORT
14
              PMIX ERR PROC ABORTING
                                             Process is being aborted
15
              PMIX_ERR_SERVER_FAILED_REQUEST
                                                        Failed to connect to the server
                                Requested operation would overwrite an existing value
16
              PMIX EXISTS
17
                                             Invalid security credentials
              PMIX_ERR_INVALID_CRED
              PMIX_ERR_HANDSHAKE_FAILED
                                                 Connection handshake failed
18
19
              PMIX_ERR_READY_FOR_HANDSHAKE
                                                     Ready for handshake
20
                                           Operation would block
              PMIX_ERR_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
              PMIX ERR UNPACK INADEQUATE SPACE
24
                                                          Inadequate space to unpack data
25
              PMIX_ERR_UNPACK_FAILURE
                                               Unpack failed
                                            Pack failed
26
              PMIX_ERR_PACK_FAILURE
27
                                              Pack mismatch
              PMIX_ERR_PACK_MISMATCH
28
              PMIX_ERR_NO_PERMISSIONS
                                               No permissions
29
              PMIX ERR TIMEOUT
                                      Timeout expired
                                      Unreachable
30
              PMIX ERR UNREACH
                                       Error defined in errno
31
              PMIX_ERR_IN_ERRNO
32
              PMIX_ERR_BAD_PARAM
                                         Bad parameter
33
              PMIX_ERR_RESOURCE_BUSY
                                              Resource busy
                                                Resource exhausted
34
              PMIX ERR OUT OF RESOURCE
                                                      Data value not found
35
              PMIX ERR DATA VALUE NOT FOUND
36
              PMIX ERR INIT
                                  Error during initialization
37
              PMIX ERR NOMEM
                                    Out of memory
                                           Invalid argument
38
              PMIX_ERR_INVALID_ARG
39
              PMIX_ERR_INVALID_KEY
                                           Invalid key
40
              PMIX_ERR_INVALID_KEY_LENGTH
                                                    Invalid key length
41
              PMIX_ERR_INVALID_VAL
                                           Invalid value
```

Invalid value length 1 PMIX\_ERR\_INVALID\_VAL\_LENGTH 2 PMIX ERR INVALID LENGTH Invalid argument length Invalid number of arguments 3 PMIX\_ERR\_INVALID\_NUM\_ARGS 4 Invalid arguments PMIX\_ERR\_INVALID\_ARGS 5 PMIX ERR INVALID NUM PARSED Invalid number parsed 6 PMIX ERR INVALID KEYVALP Invalid key/value pair 7 Invalid size PMIX ERR INVALID SIZE 8 PMIX\_ERR\_INVALID\_NAMESPACE Invalid namespace 9 PMIX\_ERR\_SERVER\_NOT\_AVAIL Server is not available 10 PMIX ERR NOT FOUND Not found Not supported 11 PMIX\_ERR\_NOT\_SUPPORTED 12 PMIX ERR NOT IMPLEMENTED Not implemented Communication failure 13 PMIX ERR COMM FAILURE PMIX ERR UNPACK READ PAST END OF BUFFER Unpacking past the end of the buffer 14 15 provided 3.1.1.2 PMIx v2 Error Constants 17 The following list contains constants added in the PMIx v2 standard. 18 PMIX ERR LOST CONNECTION TO SERVER Lost connection to server 19 PMIX\_ERR\_LOST\_PEER\_CONNECTION Lost connection to peer 20 PMIX ERR LOST CONNECTION TO CLIENT Lost connection to client Query partial success (used by query system) PMIX\_QUERY\_PARTIAL\_SUCCESS 21 22 PMIX\_NOTIFY\_ALLOC\_COMPLETE Notify that allocation is complete PMIX\_JCTRL\_CHECKPOINT Job control: Monitored by PMIx client to trigger checkpoint 23 24 operation PMIX JCTRL CHECKPOINT COMPLETE Job control: Sent by PMIx client and monitored 25 by PMIx server to notify that requested checkpoint operation has completed. 26 PMIX JCTRL PREEMPT ALERT Job control: Monitored by PMIx client to detect an RM 27 intending to preempt the job. 28 29 PMIX MONITOR HEARTBEAT ALERT Job monitoring: Heartbeat alert Job monitoring: File alert 30 PMIX\_MONITOR\_FILE\_ALERT PMIX\_PROC\_TERMINATED Process terminated - can be either normal or abnormal 31 32 termination 33 PMIX ERR INVALID TERMINATION Process terminated without calling PMIx\_Finalize, or was a member of an assemblage formed via PMIx\_Connect and 34 terminated or called PMIx\_Finalize without first calling PMIx\_Disconnect (or its 35 non-blocking form) from that assemblage. 36 37 The following list contains operational error constants introduced in the v2 standard. 38 PMIX ERR EVENT REGISTRATION Error in event registration 39 PMIX ERR JOB TERMINATED Error job terminated 40 PMIX ERR UPDATE ENDPOINTS Error updating endpoints

1 2 3 4 5		PMIX_MODEL_DECLARED Model declared  PMIX_GDS_ACTION_COMPLETE The global data storage (GDS) action has completed  PMIX_ERR_INVALID_OPERATION The requested operation is supported by the implementation and host environment, but fails to meet a requirement (e.g., requesting to disconnect from processes without first connecting to them).
6		The following list contains system error constants introduced in the v2 standard.
7 8		PMIX_ERR_NODE_DOWN Node down PMIX_ERR_NODE_OFFLINE Node is marked as offline
9		The following list contains event handler error constants introduced in the v2 standard.
10 11 12 13		PMIX_EVENT_NO_ACTION_TAKEN Event handler: No action taken  PMIX_EVENT_PARTIAL_ACTION_TAKEN Event handler: Partial action taken  PMIX_EVENT_ACTION_DEFERRED Event handler: Action deferred  PMIX_EVENT_ACTION_COMPLETE Event handler: Action complete
14	3.1.1.3	User-Defined Error Constants
15 16 17		PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative values larger than this (and any positive values greater than zero) are guaranteed not to conflict with PMIx values.
18 19 20 21		<b>PMIX_EXTERNAL_ERR_BASE</b> A starting point for user-level defined error constants. Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions should always be based on the <b>PMIX_EXTERNAL_ERR_BASE</b> constant and <i>not</i> a specific value as the value of the constant may change.
22	3.2	Data Types
23		This section defines various data types used by the PMIx APIs.
24	3.2.1	Key Structure
25 26 27		The <b>pmix_key_t</b> structure is a statically defined character array of length <b>PMIX_MAX_KEYLEN</b> +1, thus supporting keys of maximum length <b>PMIX_MAX_KEYLEN</b> while preserving space for a mandatory <b>NULL</b> terminator.
	PMIx v2.0	C
28		<pre>typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];</pre>

1 2		Characters in the key must be standard alphanumeric values supported by common utilities such as <i>strcmp</i> .
		Advice to users
3 4 5 6		References to keys in PMIx v1 rwere defined simply as an array of characters of size <b>PMIX_MAX_KEYLEN+1</b> . The <b>pmix_key_t</b> type definition was introduced in version 2 of the standard. The two definitions are code-compatible and thus do not represent a break in backward compatibility.
7 8 9		Passing a <b>pmix_key_t</b> value to the standard <i>sizeof</i> utility can result in compiler warnings of incorrect returned value. Users are advised to avoid using <i>sizeof(pmix_key_t)</i> and instead rely on the <b>PMIX_MAX_KEYLEN</b> constant.
10	3.2.2	Namespace Structure
11 12 13		The pmix_nspace_t structure is a statically defined character array of length PMIX_MAX_NSLEN +1, thus supporting namespaces of maximum length PMIX_MAX_NSLEN while preserving space for a mandatory NULL terminator.
14	PMIx v2.0	typedef char pmix_nspace_t[PMIX_MAX_NSLEN+1];
15 16		Characters in the namespace must be standard alphanumeric values supported by common utilities such as <i>strcmp</i> .
		Advice to users
17 18 19 20		References to namespace values in PMIx v1 rwere defined simply as an array of characters of size <b>PMIX_MAX_NSLEN+1</b> . The <b>pmix_nspace_t</b> type definition was introduced in version 2 of the standard. The two definitions are code-compatible and thus do not represent a break in backward compatibility.
21 22 23		Passing a <b>pmix_nspace_t</b> value to the standard <i>sizeof</i> utility can result in compiler warnings of incorrect returned value. Users are advised to avoid using <i>sizeof(pmix_nspace_t)</i> and instead rely on the <b>PMIX_MAX_NSLEN</b> constant.

#### 1 3.2.3 Rank Structure

```
The pmix_rank_t structure is a uint32_t type for rank values.
   PMIx v1.0
 3
               typedef uint32_t pmix_rank_t;
               The following constants can be used to set a variable of the type pmix_rank_t. All definitions
 4
 5
               were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at
 6
               zero.
 7
               PMIX RANK UNDEF
                                        A value to request job-level data where the information itself is not
                      associated with any specific rank, or when passing a pmix_proc_t identifier to an
 8
 9
                      operation that only references the namespace field of that structure.
               PMIX RANK WILDCARD
                                            A value to indicate that the user wants the data for the given key
10
11
                      from every rank that posted that key.
12
               PMIX RANK LOCAL NODE
                                              Special rank value used to define groups of ranks for use in
   PMIx v2.0
13
                      collectives. This constant defines the group of all ranks on a local node.
    3.2.4 Process Structure
15
               The pmix_proc_t structure is used to identify a single process in the PMIx universe. It contains
16
               a reference to the namespace and the pmix_rank_t within that namespace.
                                                           C —
   PMIx v1.0
17
               typedef struct pmix_proc {
18
                     pmix nspace t nspace;
19
                     pmix_rank_t rank;
20
                } pmix_proc_t;
```

## 1 3.2.5 Process structure support macros

The following macros are provided to support the **pmix proc** t structure.

#### 3.2.5.1 Initialize the pmix\_proc\_t structure 2 Initialize the **pmix proc t** fields *PMIx v1.0* 3 PMIX PROC CONSTRUCT (m) IN 5 Pointer to the structure to be initialized (pointer to pmix\_proc\_t) 3.2.5.2 Destruct the pmix\_proc\_t structure Clear the **pmix\_proc\_t** fields *PMIx v1.0* 8 PMIX PROC DESTRUCT (m) IN 9 10 Pointer to the structure to be destructed (pointer to pmix\_proc\_t) 11 This macro performs the identical operations as PMIX\_PROC\_CONSTRUCT, but is provided for 12 symmetry in user code. 3.2.5.3 Create a pmix\_proc\_t array Allocate and initialize an array of **pmix\_proc\_t** structures 14 PMIx v1.0 15 PMIX PROC CREATE (m, n) 16 INOUT m Address where the pointer to the array of **pmix\_proc\_t** structures shall be stored (handle) 17 IN 18 Number of structures to be allocated (size t) 19

#### 3.2.5.4 Free a pmix\_proc\_t array 2 Release an array of **pmix proc t** structures PMIx v1.03 PMIX PROC FREE (m, n) IN 4 m Pointer to the array of pmix\_proc\_t structures (handle) 5 6 IN Number of structures in the array (size\_t) 7 3.2.5.5 Load a pmix\_proc\_t structure 9 Load values into a pmix\_proc\_t PMIx v2.0 10 PMIX\_PROC\_LOAD(m, n, r) IN 11 Pointer to the structure to be loaded (pointer to pmix\_proc\_t) 12 IN 13 14 Namespace to be loaded (pmix nspace t) IN 15 16 Rank to be assigned ( pmix\_rank\_t ) **Process State Structure** 3.2.6 The pmix\_proc\_state\_t structure is a uint8\_t type for process state values. The following 18 *PMIx v2.0* constants can be used to set a variable of the type pmix\_proc\_state\_t . All values were 19 20 originally defined in version 2 of the standard unless otherwise marked. Advice to users 21 The fine-grained nature of the following constants may exceed the ability of an RM to provide 22 updated process state values during the process lifetime. This is particularly true of states in the

23

launch process, and for short-lived processes.

```
1
              PMIX_PROC_STATE_UNDEF
                                             Undefined process state
 2
              PMIX_PROC_STATE_PREPPED
                                                Process is ready to be launched
 3
              PMIX_PROC_STATE_LAUNCH_UNDERWAY
                                                          Process launch is underway
 4
              PMIX PROC STATE RESTART
                                                Process is ready for restart
 5
              PMIX PROC STATE TERMINATE
                                                  Process is marked for termination
 6
              PMIX PROC STATE RUNNING
                                                Process has been locally fork'ed by the RM
                                                  Process has connected to PMIx server
 7
              PMIX PROC STATE CONNECTED
8
              PMIX_PROC_STATE_UNTERMINATED
                                                      Define a "boundary" between this constant and
9
                    PMIX_PROC_STATE_CONNECTED so users can easily and quickly determine if a process
10
                    is still running or not. Any value less than this constant means that the process has not
                    terminated.
11
12
              PMIX PROC STATE TERMINATED
                                                    Process has terminated and is no longer running
              PMIX_PROC_STATE_ERROR
                                             Define a boundary so users can easily and quickly determine if
13
14
                    a process abnormally terminated. Any value above this constant means that the process has
15
                    terminated abnormally.
16
              PMIX_PROC_STATE_KILLED_BY_CMD
                                                        Process was killed by a command
                                                Process was aborted by a call to PMIx_Abort
17
              PMIX_PROC_STATE_ABORTED
                                                          Process failed to start
18
              PMIX_PROC_STATE_FAILED_TO_START
19
              PMIX PROC STATE ABORTED BY SIG
                                                         Process aborted by a signal
                                                      Process exited without calling PMIx Finalize
20
              PMIX_PROC_STATE_TERM_WO_SYNC
                                                     Process communication has failed
21
              PMIX_PROC_STATE_COMM_FAILED
22
              PMIX PROC STATE CALLED ABORT
                                                      Process called PMIx Abort
23
              PMIX_PROC_STATE_MIGRATING
                                                  Process failed and is waiting for resources before
24
                    restarting
                                                         Process failed and cannot be restarted
25
              PMIX_PROC_STATE_CANNOT_RESTART
26
              PMIX_PROC_STATE_TERM_NON_ZERO
                                                        Process exited with a non-zero status
27
              PMIX PROC STATE FAILED TO LAUNCH
                                                           Unable to launch process
```

#### 8 3.2.7 Process Information Structure

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

PMIx v2.0

```
1
            typedef struct pmix proc info {
2
                 /** Process structure */
3
                 pmix_proc_t proc;
                 /** Hostname where process resides */
5
                 char *hostname;
6
                 /** Name of the executable */
7
                 char *executable name;
8
                 /** Process ID on the host */
9
                 pid_t pid;
10
                 /** Exit code of the process. Default: 0 */
                 int exit code;
11
                 /** Current state of the process */
12
13
                 pmix_proc_state_t state;
             } pmix_proc_info_t;
14
15 3.2.8 Process Information Structure support macros
16
            The following macros are provided to support the pmix_proc_info_t structure.
            Initialize the pmix_proc_info_t structure
   3.2.8.1
18
            Initialize the pmix_proc_info_t fields
  PMIx v2.0
            PMIX_PROC_INFO_CONSTRUCT (m)
19
            IN
20
21
                 Pointer to the structure to be initialized (pointer to pmix proc info t)
   3.2.8.2
            Destruct the pmix_proc_info_t structure
23
            Destruct the pmix proc info t fields
  PMIx v2.0
            PMIX PROC INFO DESTRUCT (m)
24
25
            IN
```

Pointer to the structure to be destructed (pointer to **pmix\_proc\_info\_t**)

#### 3.2.8.3 Create a pmix\_proc\_info\_t array Allocate and initialize a **pmix proc info t** array 2 PMIx v2.0PMIX PROC INFO\_CREATE(m, n) 3 INOUT m 4 Address where the pointer to the array of **pmix\_proc\_info\_t** structures shall be stored 5 6 (handle) 7 IN 8 Number of structures to be allocated (size\_t) 3.2.8.4 Free a pmix proc info t array Release an array of **pmix\_proc\_info\_t** structures 10 PMIx v2.0PMIX\_PROC\_INFO\_FREE(m, n) 11 12 IN 13 Pointer to the array of **pmix proc info t** structures (handle) IN 14 15 Number of structures in the array (size t) Scope of Put Data 3.2.9 17 *PMIx v1.0* The pmix\_scope\_t structure is a uint8\_t type that defines the scope for data passed to PMIx Put. The following constants can be used to set a variable of the type pmix scope t. 18 All definitions were introduced in version 1 of the standard unless otherwise marked. 19 20 Specific implementations may support different scope values, but all implementations must support 21 at least PMIX GLOBAL. If a scope value is not supported, then the PMIx Put call must return 22 PMIX\_ERR\_NOT\_SUPPORTED. 23 PMIX SCOPE UNDEF Undefined scope PMIX LOCAL The data is intended only for other application processes on the same node. 24 Data marked in this way will not be included in data packages sent to remote requestors — 25 26 i.e., it is only available to processes on the local node. The data is intended solely for applications processes on remote nodes. Data 27 PMIX REMOTE 28 marked in this way will not be shared with other processes on the same node — i.e., it is only 29 available to processes on remote nodes.

1	PMIX_GLOBAL	The data is to be shared with all other requesting processes, regardless of
2	location.	
3 <i>PMIx v2.0</i>	PMIX_INTERNAL	The data is intended solely for this process and is not shared with other
4	processes.	

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

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

- PMIX RANGE RM Data is intended for the host resource manager.
- 13 PMIX\_RANGE\_LOCAL Data is only available to processes on the local node.
- 14 PMIX RANGE NAMESPACE Data is only available to processes in the same namespace.
- 15 PMIX RANGE SESSION Data is only available to all processes in the session.
- 16 Data is available to all processes. PMIX\_RANGE\_GLOBAL
- Range is specified in the **pmix info** t associated with this call. 17 PMIX RANGE CUSTOM
- 18 PMIX RANGE PROC LOCAL Data is only available to this process.

#### Advice to users

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

#### 21 3.2.11 **Data Persistence Structure**

- 22 *PMIx v1.0* The pmix\_persistence\_t structure is a uint8\_t type that defines the policy for data 23 published by clients via the **PMIx Publish** API. The following constants can be used to set a 24 variable of the type pmix persistence t. All definitions were introduced in version 1 of the 25 standard unless otherwise marked.
- 26 PMIX PERSIST INDEF Retain data until specifically deleted.
- PMIX\_PERSIST\_FIRST\_READ Retain data until the first access, then the data is deleted. 27
- 28 Retain data until the publishing process terminates. PMIX PERSIST PROC
- 29 PMIX PERSIST APP Retain data until the application terminates.
- 30 PMIX\_PERSIST\_SESSION Retain data until the session/allocation terminates.

#### 3.2.12 Value Structure

38

39 40

2 The pmix\_value\_t structure is used to represent the value passed to PMIx\_Put and retrieved by PMIx\_Get, as well as many of the other PMIx functions. 3 A collection of values may be specified under a single key by passing a pmix\_value\_t 4 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 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; pid\_t pid; 15 16 int integer; int8\_t int8; 17 int16\_t int16; 18 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 // version 2.0 time\_t time; // version 2.0 30 pmix\_status\_t status; 31 pmix\_rank\_t rank; // version 2.0 32 pmix\_proc\_t \*proc; // version 2.0 pmix\_byte\_object\_t bo; 33 pmix\_persistence\_t persist; // version 2.0 34 // version 2.0 35 pmix scope t scope; 36 pmix\_data\_range\_t range; // version 2.0 37 pmix\_proc\_state\_t state; // version 2.0

pmix\_proc\_info\_t \*pinfo;

void \*ptr;

pmix\_data\_array\_t \*darray;

// version 2.0

// version 2.0

// 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;
   3.2.13
              Value structure support macros
              The following macros are provided to support the pmix_value_t structure.
 8
    3.2.13.1
             Initialize the pmix_value_t structure
10
              Initialize the pmix_value
t fields
   PMIx v1.0
11
              PMIX VALUE CONSTRUCT (m)
              IN
12
13
                   Pointer to the structure to be initialized (pointer to pmix_value_t)
   3.2.13.2 Destruct the pmix_value_t structure
15
              Destruct the pmix value t fields
   PMIx v1.0
16
              PMIX VALUE DESTRUCT (m)
              IN
17
                   Pointer to the structure to be destructed (pointer to pmix value t)
18
   3.2.13.3 Create a pmix_value_t array
20
              Allocate and initialize an array of pmix value t structures
   PMIx v1.0
21
              PMIX VALUE CREATE (m, n)
              INOUT m
22
                   Address where the pointer to the array of pmix_value_t structures shall be stored
23
                   (handle)
24
              IN
                  n
25
                   Number of structures to be allocated (size_t)
26
```

#### 3.2.13.4 Free a pmix\_value\_t array 2 Release an array of **pmix value** t structures PMIx v1.03 PMIX VALUE FREE (m, n) IN 4 Pointer to the array of **pmix\_value\_t** structures (handle) 5 IN 6 Number of structures in the array (size\_t) 3.2.14 Load a pmix value t structure Summary 9 10 Load data into a **pmix value** t structure. PMIx v2.0 11 PMIX VALUE LOAD (v, d, t); IN 12 The pmix\_value\_t into which the data is to be loaded (pointer to pmix\_value\_t) 13 IN 14 15 Pointer to the data value to be loaded (handle) IN 16 Type of the provided data value ( pmix\_data\_type\_t ) 17 **Description** 18 19 This macro simplifies the loading of data into a **pmix\_value\_t** by correctly assigning values to the structure's fields. 20 Advice to users -The data will be copied into the **pmix\_value\_t** - thus, any data stored in the source value can be 21 22 modified or free'd without affecting the copied data once the macro has completed.

#### 3.2.14.1 Transfer data between pmix\_value\_t structures

```
Summary
2
 3
               Transfer the data value between two pmix_value_t structures.
   PMIx v2.0
 4
               PMIX_VALUE_XFER(r, d, s);
               OUT r
5
6
                     Status code indicating success or failure of the transfer (pmix status t)
7
               IN
                     Pointer to the pmix value t destination (handle)
8
               IN
9
10
                     Pointer to the pmix value t source (handle)
               Description
11
12
               This macro simplifies the transfer of data between two pmix_value_t structures, ensuring that
               all fields are properly copied.
13
                                                Advice to users -
14
               The data will be copied into the destination pmix_value_t - thus, any data stored in the source
15
               value can be modified or free'd without affecting the copied data once the macro has completed.
               Info and Info Array Structures
   3.2.15
17
               The pmix info t structure defines a key/value pair with associated directive. All fields were
               defined in version 1.0 unless otherwise marked.
18
   PMIx v1.0
19
               typedef struct pmix info t {
20
                    pmix_key_t key;
                    pmix_info_directives_t flags; // version 2.0
21
22
                    pmix_value_t value;
23
               } 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
2
              versions of the PMIx Standard.
   PMIx v1.0
3
              typedef struct pmix_info_array {
4
                  size_t size;
 5
                  pmix_info_t *array;
              } pmix_info_array_t;
6
   3.2.16
              Info structure support macros
8
              The following macros are provided to support the pmix_info_t structure.
             Initialize the pmix_info_t structure
   3.2.16.1
10
              Initialize the pmix_info_t fields
   PMIx v1.0
11
             PMIX INFO CONSTRUCT (m)
              IN
12
13
                   Pointer to the structure to be initialized (pointer to pmix_info_t)
   3.2.16.2 Destruct the pmix_info_t structure
              Destruct the pmix_info_t fields
15
   PMIx v1.0
             PMIX INFO DESTRUCT (m)
16
              IN
17
```

Pointer to the structure to be destructed (pointer to **pmix\_info\_t**)

#### 3.2.16.3 Create a pmix\_info\_t array Allocate and initialize an array of **pmix info t** structures 2 PMIx v1.03 PMIX INFO CREATE (m, n) INOUT m 4 Address where the pointer to the array of **pmix\_info\_t** structures shall be stored (handle) 5 IN 6 Number of structures to be allocated (size\_t) 7 3.2.16.4 Free a pmix\_info\_t array 9 Release an array of **pmix\_info\_t** structures *PMIx v1.0* 10 PMIX\_INFO\_FREE(m, n) IN 11 12 Pointer to the array of **pmix info t** structures (handle) 13 IN Number of structures in the array (size t) 14 3.2.16.5 Load key and value data into a pmix\_info\_t 15 *PMIx v1.0* 16 PMIX INFO LOAD (v, k, d, t); IN 17 Pointer to the pmix\_info\_t into which the key and data are to be loaded (pointer to 18 pmix\_info\_t) 19 IN 20 String key to be loaded - must be less than or equal to PMIX MAX KEYLEN in length 21 22 (handle) 23 IN Pointer to the data value to be loaded (handle) 24 25 IN Type of the provided data value (pmix data type t) 26 This macro simplifies the loading of key and data into a **pmix\_info\_t** by correctly assigning 27 values to the structure's fields. 28

		Advice to users		
1 2 3		Both key and data will be copied into the <code>pmix_info_t</code> - thus, the key and any data stored in the source value can be modified or free'd without affecting the copied data once the macro has completed.		
4	3.2.16.6	Copy data between pmix_info_t structures		
5		Copy all data (including key, value, and directives) between two pmix_info_t structures.		
	PMIx v2.0	▼ C		
6		PMIX_INFO_XFER(d, s);		
7		IN d		
8		Pointer to the destination pmix_info_t (pointer to pmix_info_t)		
9		<pre>IN s     Pointer to the source pmix_info_t (pointer to pmix_info_t)</pre>		
1		This macro simplifies the transfer of data between two pmix_info_t structures.		
		Advice to users		
2 3 4		All data (including key, value, and directives) will be copied into the destination <code>pmix_info_t</code> - thus, the source <code>pmix_info_t</code> may be free'd without affecting the copied data once the macro has completed.		
5	3.2.16.7	Test a boolean pmix_info_t		
6		A special macro for checking if a boolean <b>pmix_info_t</b> is <b>true</b>		
	PMIx v2.0	▼ C		
7		PMIX_INFO_TRUE (m)		
8 9		IN m Pointer to a pmix_info_t structure (handle)		
0		A pmix_info_t structure is considered to be of type PMIX_BOOL and value true if:		
1		• the structure reports a type of <b>PMIX_UNDEF</b> , or		
2		• the structure reports a type of PMTX_BOOT, and the data flag is true		

# 1 3.2.17 Info Type Directives

The pmix\_info\_directives\_t structure is a uint32\_t type that defines the behavior of command directives via pmix\_info\_t arrays. By default, the values in the pmix\_info\_t array passed to a PMIx are optional.

Advice to users

A PMIx implementation or PMIx-enabled RM may ignore any <code>pmix\_info\_t</code> value passed to a PMIx API if it is not explicitly marked as <code>PMIX\_INFO\_REQD</code>. This is because the values specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if the user is relying on the behavior specified by the <code>pmix\_info\_t</code> value. If the user relies on the behavior defined by the <code>pmix\_info\_t</code> then they must set the <code>PMIX\_INFO\_REQD</code> flag using the <code>PMIX\_INFO\_REQUIRED</code> macro.

Advice to PMIx library implementers

The top 16-bits of the <code>pmix\_info\_directives\_t</code> are reserved for internal use by PMIx library implementers - the PMIx standard will *not* specify their intent, leaving them for customized use by implementers. Implementers are advised to use the provided <code>PMIX\_INFO\_IS\_REQUIRED</code> macro for testing this flag, and must return <code>PMIX\_ERR\_NOT\_SUPPORTED</code> as soon as possible to the caller if the required behavior is not supported.

The following constants were introduced in version 2.0 (unless otherwise marked) and can be used to set a variable of the type <code>pmix\_info\_directives\_t</code>.

**PMIX\_INFO\_REQD** The behavior defined in the **pmix\_info\_t** array is required, and not optional. This is a bit-mask value.

#### Advice to PMIx server hosts -

Host environments are advised to use the provided **PMIX\_INFO\_IS\_REQUIRED** macro for testing this flag and must return **PMIX\_ERR\_NOT\_SUPPORTED** as soon as possible to the caller if the required behavior is not supported.

## 23 3.2.18 Info Directive support macros

The following macros are provided to support the setting and testing of **pmix\_info\_t** directives.

# 1 3.2.18.1 Mark an info structure as required

2	Summary				
3	Set the PMIX_INFO_REQD flag in a pmix_info_t structure.				
PMIx v2.0	▼ C -				
4	PMIX_INFO_REQUIRED(info);				
	C				
5 6	<pre>IN info     Pointer to the pmix_info_t (pointer to pmix_info_t)</pre>				
7 8 <b>3.2.18.2</b>	This macro simplifies the setting of the PMIX_INFO_REQD flag in pmix_info_t structures.  Test an info structure for required directive				
9	Summary				
0	Test the PMIX_INFO_REQD flag in a pmix_info_t structure, returning true if the flag is set.				
PMIx v2.0	▼ C -				
1	PMIX_INFO_IS_REQUIRED(info);				
2	IN info				
3	Pointer to the <b>pmix_info_t</b> (pointer to <b>pmix_info_t</b> )				
4	This macro simplifies the testing of the required flag in <b>pmix_info_t</b> structures.				
5 <b>3.2.19</b>	Job Allocation Directives				
6 <i>PMIx v2.0</i> 7 8 9	The <code>pmix_alloc_directive_t</code> structure is a <code>uint8_t</code> type that defines the behavior of allocation requests. The following constants can be used to set a variable of the type <code>pmix_alloc_directive_t</code> . All definitions were introduced in version 2 of the standard unless otherwise marked.				
20 21 22	PMIX_ALLOC_NEW A new allocation is being requested. The resulting allocation will be disjoint (i.e., not connected in a job sense) from the requesting allocation.  PMIX_ALLOC_EXTEND Extend the existing allocation, either in time or as additional				
23	resources.				
24 25	<b>PMIX_ALLOC_RELEASE</b> Release part of the existing allocation. Attributes in the accompanying <b>pmix_info_t</b> array may be used to specify permanent release of the				
26	identified resources, or "lending" of those resources for some period of time.				
27	PMIX_ALLOC_REAQUIRE Reacquire resources that were previously "lent" back to the				
!8 !9	scheduler.  PMIX_ALLOC_EXTERNAL A value boundary above which implementers are free to define				
10	their own directive values				

### 1 3.2.20 Lookup Returned Data Structure

```
2
             The pmix_pdata_t structure is used by PMIx_Lookup to describe the data being accessed.
                                                 C _____
  PMIx v1.0
3
             typedef struct pmix_pdata {
4
                 pmix_proc_t proc;
                 pmix_key_t key;
5
                 pmix_value_t value;
6
             } pmix_pdata_t;
7
   3.2.21
             Lookup data structure support macros
             The following macros are provided to support the pmix pdata t structure.
9
   3.2.21.1 Initialize the pmix_pdata_t structure
11
             Initialize the pmix_pdata_t fields
  PMIx v1.0
12
             PMIX PDATA CONSTRUCT (m)
13
             IN
                  Pointer to the structure to be initialized (pointer to pmix_pdata_t)
14
   3.2.21.2
              Destruct the pmix_pdata_t structure
             Destruct the pmix_pdata_t fields
16
  PMIx v1.0
17
             PMIX_PDATA_DESTRUCT (m)
             IN
18
```

Pointer to the structure to be destructed (pointer to pmix\_pdata\_t)

#### Allocate and initialize an array of pmix\_pdata\_t structures 2 PMIx v1.03 PMIX PDATA CREATE (m, n) INOUT m 4 Address where the pointer to the array of pmix\_pdata\_t structures shall be stored 5 6 (handle) 7 IN 8 Number of structures to be allocated (size\_t) 3.2.21.4 Free a pmix\_pdata\_t array Release an array of **pmix\_pdata\_t** structures 10 PMIx v1.0PMIX\_PDATA\_FREE(m, n) 11 12 IN 13 Pointer to the array of **pmix pdata t** structures (handle) IN 14 Number of structures in the array (size\_t) 15 3.2.21.5 Load a lookup data structure Summary 17 Load key, process identifier, and data value into a pmix\_pdata\_t structure. 18 PMIx v1.0 PMIX\_PDATA\_LOAD(m, p, k, d, t); 19 IN 20 Pointer to the pmix\_pdata\_t structure into which the key and data are to be loaded 21 (pointer to pmix pdata t) 22 23 IN Pointer to the pmix\_proc\_t structure containing the identifier of the process being 24 referenced (pointer to pmix\_proc\_t) 25 IN 26 27 String key to be loaded - must be less than or equal to PMIX MAX KEYLEN in length 28 (handle)

3.2.21.3 Create a pmix\_pdata\_t array

```
IN
 1
                    d
 2
                     Pointer to the data value to be loaded (handle)
 3
               IN
 4
                     Type of the provided data value ( pmix_data_type_t )
 5
               This macro simplifies the loading of key, process identifier, and data into a pmix_proc_t by
               correctly assigning values to the structure's fields.
 6
                                                 Advice to users
               Key, process identifier, and data will all be copied into the pmix pdata t - thus, the source
 7
               information can be modified or free'd without affecting the copied data once the macro has
 8
 9
               completed.
    3.2.21.6 Transfer a lookup data structure
               Summary
11
12
               Transfer key, process identifier, and data value between two pmix_pdata_t structures.
   PMIx v2.0
13
               PMIX PDATA XFER(d, s);
               IN
                    d
14
                     Pointer to the destination pmix_pdata_t (pointer to pmix_pdata_t)
15
               IN
16
17
                     Pointer to the source pmix_pdata_t (pointer to pmix_pdata_t)
               This macro simplifies the transfer of key and data between two pmix pdata t structures.
18
                            Advice to users
19
               Key, process identifier, and data will all be copied into the destination pmix pdata t - thus, the
               source pmix pdata t may free'd without affecting the copied data once the macro has
20
               completed.
21
```

## 3.2.22 Application Structure

```
2
            The pmix_app_t structure describes the application context for the PMIx_Spawn and
3
            PMIx Spawn nb operations.
  PMIx v1.0
4
            typedef struct pmix_app {
5
                 /** Executable */
6
                 char *cmd;
7
                 /** Argument set, NULL terminated */
8
                 char **argv;
9
                 /** Environment set, NULL terminated */
10
                char **env;
                /** Current working directory */
11
12
                char *cwd;
13
                /** Maximum processes with this profile */
                int maxprocs;
14
                 /** Array of info keys describing this application*/
15
                pmix_info_t *info;
16
                 /** Number of info keys in 'info' array */
17
18
                 size t ninfo;
            } pmix_app_t;
19
```

## 3.2.23 App structure support macros

The following macros are provided to support the pmix app t structure.

### 22 3.2.23.1 Initialize the pmix\_app\_t structure

Initialize the pmix\_app\_t fields

PMIx v1.0

PMIX\_APP\_CONSTRUCT (m)

IN m

Pointer to the structure to be initialized (pointer to pmix\_app\_t)

```
3.2.23.2 Destruct the pmix_app_t structure
2
              Destruct the pmix_app_t fields
   PMIx v1.0
3
              PMIX APP DESTRUCT (m)
              IN
 5
                    Pointer to the structure to be destructed (pointer to pmix_app_t)
    3.2.23.3 Create a pmix_app_t array
              Allocate and initialize an array of pmix_app_t structures
   PMIx v1.0
8
              PMIX_APP_CREATE(m, n)
              INOUT m
9
10
                    Address where the pointer to the array of pmix_app_t structures shall be stored (handle)
              IN
11
                    Number of structures to be allocated (size_t)
12
    3.2.23.4 Free a pmix_app_t array
14
              Release an array of pmix app t structures
   PMIx v1.0
15
              PMIX APP FREE (m, n)
              IN
16
                    Pointer to the array of pmix_app_t structures (handle)
17
18
              IN
                    Number of structures in the array (size_t)
19
```

### 1 3.2.24 Query Structure

IN

19 20

```
2
             The pmix_query_t structure is used by PMIx_Query_info_nb to describe a single query
3
             operation.
   PMIx v2.0
4
             typedef struct pmix_query {
5
                  char **keys;
6
                  pmix_info_t *qualifiers;
7
                  size_t nqual;
8
             } pmix_query_t;
   3.2.25
            Query structure support macros
             The following macros are provided to support the pmix query t structure.
10
   3.2.25.1 Initialize the pmix_query_t structure
             Initialize the pmix_query_t fields
12
   PMIx v2.0
13
             PMIX QUERY CONSTRUCT (m)
             IN
14
                   Pointer to the structure to be initialized (pointer to pmix_query_t)
15
   3.2.25.2 Destruct the pmix_query_t structure
             Destruct the pmix_query_t fields
17
   PMIx v2.0
             PMIX QUERY DESTRUCT (m)
18
```

Pointer to the structure to be destructed (pointer to pmix query t)

```
3.2.25.3 Create a pmix_query_t array
              Allocate and initialize an array of pmix_query_t structures
2
   PMIx v2.0
              PMIX_QUERY_CREATE(m, n)
 3
              INOUT m
4
5
                    Address where the pointer to the array of pmix query t structures shall be stored
6
                    (handle)
 7
              IN
8
                    Number of structures to be allocated (size_t)
    3.2.25.4 Free a pmix query t array
10
              Release an array of pmix_query_t structures
   PMIx v2.0
              PMIX QUERY FREE (m, n)
11
              IN
12
13
                    Pointer to the array of pmix_query_t structures (handle)
              IN
14
15
                    Number of structures in the array (size t)
   3.2.26 Modex Structure
17
              The pmix modex data t structure describes the business card exchange (BCX) information.
              Note: This structure and its supporting macros have been deprecated and will be removed in future
18
              versions of the PMIx Standard.
19
   PMIx v1.0
20
              typedef struct pmix_modex_data {
21
                   pmix_nspace_t nspace;
22
                   int rank;
                   uint8_t *blob;
23
24
                   size t size;
25
               } pmix_modex_data_t;
```

## 3.2.27 Modex data structure support macros

The following macros are provided to support the pmix modex data t structure. 3.2.27.1 Initialize the pmix\_modex\_data\_t structure Initialize the pmix\_modex\_data\_t fields PMIx v1.0 5 PMIX MODEX CONSTRUCT (m) IN 6 m Pointer to the structure to be initialized (pointer to pmix\_modex\_data\_t) 3.2.27.2 Destruct the pmix\_modex\_data\_t structure Destruct the pmix\_modex\_data\_t fields PMIx v1.010 PMIX MODEX DESTRUCT (m) IN 11 12 Pointer to the structure to be destructed (pointer to pmix\_modex\_data\_t) 3.2.27.3 Create a pmix\_modex\_data\_t array Allocate and initialize an array of pmix\_modex\_data\_t structures 14 *PMIx v1.0* PMIX\_MODEX\_CREATE(m, n) 15 16 INOUT m 17 Address where the pointer to the array of pmix modex\_data\_t structures shall be stored (handle) 18 IN 19 Number of structures to be allocated (size t) 20

#### 3.2.27.4 Free a pmix\_modex\_data\_t array Release an array of **pmix modex data t** structures 2 PMIx v1.03 PMIX MODEX FREE (m, n) IN 4 Pointer to the array of pmix\_modex\_data\_t structures (handle) 5 6 IN 7 Number of structures in the array (size\_t) Data Packing/Unpacking Types and Structures 9 This section defines types and structures used to pack and unpack data passed through the PMIx 10 API. 3.3.1 Byte Object Type 12 The pmix byte object t structure describes a raw byte sequence. *PMIx v1.0* 13 typedef struct pmix byte object { 14 char \*bytes; size t size; 15 16 } pmix\_byte\_object\_t; Byte object support macros 3.3.2 The following macros support the pmix\_byte\_object\_t structure. 18 Initialize the pmix\_byte\_object\_t structure 3.3.2.1 20 Initialize the pmix byte object t fields PMIx v2.0 21 PMIX\_BYTE\_OBJECT\_CONSTRUCT (m) IN 22 Pointer to the structure to be initialized (pointer to pmix byte object t) 23

#### 3.3.2.2 Destruct the pmix\_byte\_object\_t structure Clear the pmix\_byte\_object\_t fields 2 PMIx v2.03 PMIX BYTE OBJECT DESTRUCT (m) IN 5 Pointer to the structure to be destructed (pointer to pmix\_byte\_object\_t) 3.3.2.3 Create a pmix\_byte\_object\_t structure Allocate and intitialize an array of pmix\_byte\_object\_t structures PMIx v2.08 PMIX\_BYTE\_OBJECT\_CREATE(m, n) INOUT m 9 Address where the pointer to the array of pmix\_byte\_object\_t structures shall be 10 stored (handle) 11 IN 12 13 Number of structures to be allocated (size t) 3.3.2.4 Free a pmix\_byte\_object\_t array 15 Release an array of pmix\_byte\_object\_t structures PMIx v2.016 PMIX BYTE OBJECT FREE (m, n) IN 17 18 Pointer to the array of pmix\_byte\_object\_t structures (handle) IN 19 Number of structures in the array (size\_t) 20

#### 3.3.2.5 Load a pmix\_byte\_object\_t structure

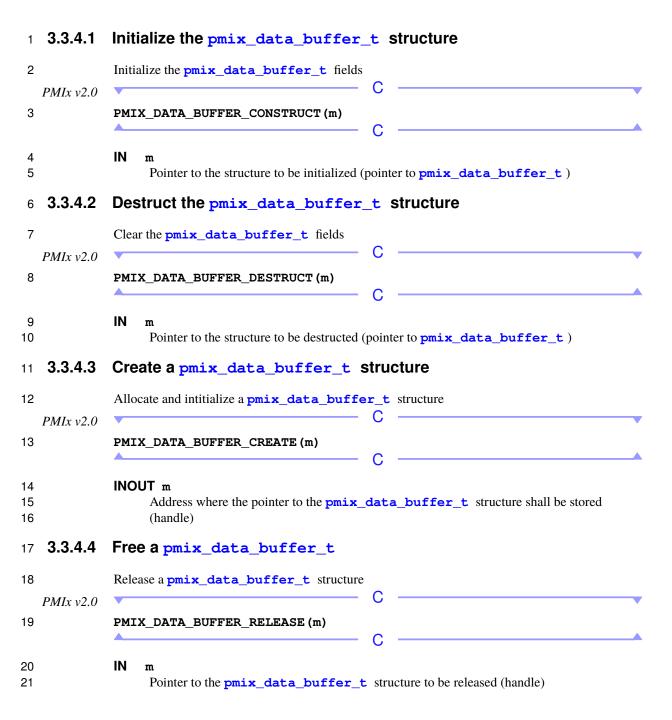
```
2
              Load values into a pmix_byte_object_t
  PMIx v2.0
3
              PMIX BYTE OBJECT LOAD (b, d, s)
              IN
4
5
                    Pointer to the structure to be loaded (pointer to pmix_byte_object_t)
6
              IN
7
                    Pointer to the data to be loaded (char*)
8
              IN
9
                    Number of bytes in the data array (size_t)
```

# 10 3.3.3 Data Buffer Type

```
The pmix_data_buffer_t structure describes a data buffer used for packing and unpacking.
11
  PMIx v2.0
12
            typedef struct pmix_data_buffer {
13
                 /** Start of my memory */
                char *base ptr;
14
15
                /** Where the next data will be packed to (within the allocated
16
                     memory starting at base_ptr) */
17
                char *pack_ptr;
18
                /** Where the next data will be unpacked from (within the
                     allocated memory starting as base ptr) */
19
                char *unpack ptr;
20
21
                /** Number of bytes allocated (starting at base_ptr) */
22
                size_t bytes_allocated;
23
                /** Number of bytes used by the buffer (i.e., amount of data --
                     including overhead -- packed in the buffer) */
24
25
                size_t bytes_used;
            } pmix data buffer t;
26
```

## 3.3.4 Data buffer support macros

The following macros support the **pmix\_data\_buffer\_t** structure.



## 1 3.3.5 Data Array Structure

The pmix\_data\_array\_t structure defines an array data structure.

PMIx v2.0

typedef struct pmix\_data\_array {
 pmix\_data\_type\_t type;
 size\_t size;
 void \*array;
 pmix\_data\_array\_t;

pmix\_data\_array\_t;

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

The **pmix\_data\_type\_t** structure is a **uint16\_t** type for identifying the data type for packing/unpacking purposes.

— Advice to PMIx library implementers -

The following constants can be used to set a variable of the type <code>pmix\_data\_type\_t</code>. Data types in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner. Additionally, a PMIx implementation may choose to add additional types.

#### 6 3.3.6.1 PMIx v1 Data Types

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

```
18
              PMIX UNDEF
                               Undefined
19
              PMIX BOOL
                              Boolean (converted to/from native true/false) (bool)
20
              PMIX BYTE
                              A byte of data (uint8 t)
              PMIX_STRING
21
                                NULL terminated string (char*)
22
              PMIX SIZE
                             Size size t
23
                            Operating process identifier (PID) (pid_t)
              PMIX_PID
24
              PMIX_INT
                            Integer (int)
25
              PMIX INT8
                             8-byte integer (int8 t)
                               16-byte integer (int16 t)
26
              PMIX INT16
27
              PMIX INT32
                               32-byte integer (int32_t)
              PMIX_INT64
28
                               64-byte integer (int64 t)
                              Unsigned integer (unsigned int)
29
              PMIX_UINT
                               Unsigned 8-byte integer (uint8_t)
30
              PMIX UINT8
```

9

10

11

12

13

14

15

```
Unsigned 16-byte integer (uint16_t)
 1
              PMIX_UINT16
2
                               Unsigned 32-byte integer (uint32_t)
              PMIX UINT32
 3
              PMIX_UINT64
                               Unsigned 64-byte integer (uint64_t)
 4
                             Float (float)
              PMIX_FLOAT
 5
              PMIX DOUBLE
                               Double (double)
6
                                Time value (struct timeval)
              PMIX TIMEVAL
 7
              PMIX TIME
                            Time (time t)
8
                             Value ( pmix_value_t )
              PMIX_VALUE
9
              PMIX PROC
                            Process ( pmix_proc_t )
10
                           Application context
              PMIX APP
                            Info object
11
              PMIX_INFO
12
              PMIX PDATA
                             Pointer to data
13
              PMIX BUFFER
                               Buffer
14
              PMIX BYTE OBJECT
                                     Byte object ( pmix_byte_object_t )
15
                            Key/value pair
              PMIX KVAL
16
              PMIX_MODEX (Deprecated in PMIx 2.0)
                                                     Modex
17
              PMIX PERSIST
                                Persistance (pmix_persistence_t)
              PMIX_INFO_ARRAY (Deprecated in PMIx 2.0)
18
                                                           Info array
   3.3.6.2
             PMIx v2 Data Types
19
20
              The following types were introduced in version 2 of the PMIx Standard.
21
              PMIX STATUS
                               Status (pmix_status_t)
22
              PMIX POINTER
                                Pointer (void*)
23
              PMIX SCOPE
                             Scope (pmix scope t)
24
              PMIX DATA RANGE
                                    Data range (pmix data range t)
25
              PMIX COMMAND
                                Command
26
              PMIX_INFO_DIRECTIVES
                                          Info directives
27
              PMIX DATA TYPE
                                  Data type
28
              PMIX_PROC_STATE
                                    Process state ( pmix_proc_state_t )
                                  Process info (pmix proc info t)
29
              PMIX_PROC_INFO
30
              PMIX DATA ARRAY
                                    Data array (pmix data array t)
31
              PMIX PROC RANK
                                  Process rank (pmix rank t)
32
              PMIX QUERY
                             Query
33
              PMIX_COMPRESSED_STRING
                                            Compressed string (with zlib)
                                          Allocation directive ( pmix_alloc_directive t )
34
              PMIX_ALLOC_DIRECTIVE
35
              PMIX_DATA_TYPE_MAX
                                       A boundary for implementers above which they can add their own
```

36

data types.

### 1 3.4 Reserved attributes

5

6 7

8

9

10

11 12

13 14

15

16

18

19

20

21

22

23 24

25

26

27

28

29

30 31

32

33

34

35

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

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

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

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

#### PMIX ATTR UNDEF NULL (NULL)

Constant representing an undefined attribute.

#### 7 3.4.1 Initialization attributes

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

```
PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
```

Pointer to libevent **base** to use in place of the internal progress thread.

```
PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)
```

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

```
PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)
```

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

```
PMIX SERVER SYSTEM SUPPORT "pmix.srvr.sys" (bool)
```

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

```
PMIX SERVER TMPDIR "pmix.srvr.tmpdir" (char*)
```

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

```
PMIX SYSTEM TMPDIR "pmix.sys.tmpdir" (char*)
```

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

<sup>&</sup>lt;sup>1</sup>http://libevent.org/

```
1
             PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
2
                  Registration is for the namespace only. Do not copy job data.
3
             PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)
4
                  Enable PMIx internal monitoring by the PMIx server.
5
             PMIX SERVER NSPACE "pmix.srv.nspace" (char*)
6
                  Name of the namespace to use for this PMIx server.
             PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
7
8
                  Rank of this PMIx server
```

#### 3.4.2 Tool-related attributes

```
10
              These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by
              passing them into the PMIx_tool_init API - thus, they are not typically accessed via the
11
              PMIx Get API.
12
              PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
13
14
                    Name of the namespace to use for this tool.
15
              PMIX TOOL RANK "pmix.tool.rank" (uint32 t)
16
                    Rank of this tool.
17
              PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
18
                    PID of the target PMIx server for a tool.
              PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
19
20
                    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)
21
22
                    Preferentially, look for a system-level PMIx server first.
              PMIX SERVER URI "pmix.srvr.uri" (char*)
23
24
                    uniform resource identifier (URI) of the PMIx server to be contacted.
25
              PMIX SERVER HOSTNAME "pmix.srvr.host" (char*)
                    Host where target PMIx server is located.
26
              PMIX_CONNECT_MAX_RETRIES "pmix.tool.mretries" (uint32_t)
27
28
                    Maximum number of times to try to connect to PMIx server.
29
              PMIX CONNECT RETRY DELAY "pmix.tool.retry" (uint32 t)
30
                    Time in seconds between connection attempts to a PMIx server.
31
              PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
32
                    The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
```

### 33 3.4.3 Identification attributes

34

35

36

37

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

```
PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.
```

```
1
              PMIX_GRPID "pmix.egid" (uint32_t)
2
                   Effective group id.
3
              PMIX_DSTPATH "pmix.dstpath" (char*)
4
                   Path to shared memory data storage (dstore) files.
              PMIX VERSION INFO "pmix.version" (char*)
5
6
                   PMIx version of contractor.
              PMIX PROGRAMMING MODEL "pmix.pgm.model" (char*)
7
8
                   Programming model being initialized (e.g., "MPI" or "OpenMP")
              PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)
9
10
                   Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
              PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
11
12
                   Programming model version string (e.g., "2.1.1")
              PMIX THREADING MODEL "pmix.threads" (char*)
13
14
                   Threading model used (e.g., "pthreads")
15
              PMIX REQUESTOR IS TOOL "pmix.req.tool" (bool)
16
                   The requesting process is a PMIx tool.
17
              PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
                   The requesting process is a PMIx client.
18
```

#### UNIX socket rendezvous socket attributes 19 3.4.4

20 These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing 21 them into the relevant initialization API - thus, they are not typically accessed via the PMIx Get API. 22

```
PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)
     Disable legacy UNIX socket (usock) support
PMIX SOCKET MODE "pmix.sockmode" (uint32 t)
     POSIX mode_t (9 bits valid)
PMIX SINGLE LISTENER "pmix.sing.listnr" (bool)
```

Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

## 30 3.4.5 TCP connection attributes

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

```
34
               PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
                    If provided, directs that the TCP URI be reported and indicates the desired method of
35
                    reporting: '-' for stdout, '+' for stderr, or filename.
36
37
               PMIX_TCP_URI "pmix.tcp.uri" (char*)
```

23

24

25 26

27

28 29

31 32

```
1
                    The URI of the PMIx server to connect to, or a file name containing it in the form of
2
                    file: <name of file containing it>.
3
              PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)
4
                    Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to
                    include when establishing the TCP connection.
5
              PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*)
6
                    Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
7
                    TCP connection.
8
9
              PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
                    The IPv4 port to be used.
10
              PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
11
12
                    The IPv6 port to be used.
              PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)
13
                    Set to true to disable IPv4 family of addresses.
14
              PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
15
                    Set to true to disable IPv6 family of addresses.
16
17 3.4.6 Global Data Storage (GDS) attributes
18
              These attributes are used to define the behavior of the GDS used to manage key/value pairs by
              passing them into the relevant initialization API - thus, they are not typically accessed via the
19
20
              PMIx Get API.
21
              PMIX GDS MODULE "pmix.gds.mod" (char*)
22
                    Comma-delimited string of desired modules.
             General process-level attributes
23 3.4.7
24
```

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

```
PMIX CPUSET "pmix.cpuset" (char*)
     hwloc<sup>2</sup> bitmap to be applied to the process upon launch.
PMIX CREDENTIAL "pmix.cred" (char*)
     Security credential assigned to the process.
PMIX_SPAWNED "pmix.spawned" (bool)
```

true if this process resulted from a call to PMIx Spawn.

PMIX\_ARCH "pmix.arch" (uint32\_t)

Architecture flag.

25

26 27

28

29

30

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

### 1 3.4.8 Scratch directory attributes

```
2
               These attributes are used to define an application scratch directory and are referenced using the
 3
               PMIX RANK WILDCARD rank.
               PMIX_TMPDIR "pmix.tmpdir" (char*)
 4
                    Full path to the top-level temporary directory assigned to the session.
5
               PMIX NSDIR "pmix.nsdir" (char*)
6
7
                    Full path to the temporary directory assigned to the namespace, under PMIX_TMPDIR.
8
               PMIX PROCDIR "pmix.pdir" (char*)
9
                    Full path to the subdirectory under PMIX NSDIR assigned to the process.
10
               PMIX TDIR RMCLEAN "pmix.tdir.rmclean" (bool)
                    Resource Manager will clean session directories
11
12 3.4.9
            Relative Rank Descriptive Attributes
13
               These attributes are used to describe information about relative ranks as assigned by the RM, and
               thus are referenced using the process rank except where noted.
14
15
               PMIX_PROCID "pmix.procid" (pmix_proc_t)
                    Process identifier
16
               PMIX_NSPACE "pmix.nspace" (char*)
17
18
```

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

Job identifier assigned by the scheduler.

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

Application number within the job.

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

Process rank within the job.

PMIX GLOBAL RANK "pmix.grank" (pmix rank t)

Process rank spanning across all jobs in this session.

PMIX APP RANK "pmix.apprank" (pmix rank t)

Process rank within this application.

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

Starting global rank of this job - referenced using PMIX\_RANK\_WILDCARD.

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

Local rank on this node within this job.

PMIX NODE RANK "pmix.nrank" (uint16 t)

Process rank on this node spanning all jobs.

PMIX LOCALLDR "pmix.lldr" (pmix rank t)

Lowest rank on this node within this job - referenced using PMIX\_RANK\_WILDCARD.

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

Lowest rank in this application within this job - referenced using PMIX\_RANK\_WILDCARD.

PMIX PROC PID "pmix.ppid" (pid t)

19

20

21

22

23 24

25

26 27

28

29

30 31

32

33

34

35

36 37

38

```
1
                     PID of specified process.
2
               PMIX_SESSION_ID "pmix.session.id" (uint32_t)
                     Session identifier - referenced using PMIX_RANK_WILDCARD.
3
4
               PMIX_NODE_LIST "pmix.nlist" (char*)
5
                     Comma-delimited list of nodes running processes for the specified namespace - referenced
6
                     using PMIX RANK WILDCARD.
7
               PMIX ALLOCATED NODELIST "pmix.alist" (char*)
                     Comma-delimited list of all nodes in this allocation regardless of whether or not they
8
9
                     currently host processes - referenced using PMIX RANK WILDCARD.
               PMIX HOSTNAME "pmix.hname" (char*)
10
                     Name of the host where the specified process is running.
11
               PMIX_NODEID "pmix.nodeid" (uint32_t)
12
                     Node identifier where the specified process is located, expressed as the node's index
13
                     (beginning at zero) in the array resulting from expansion of the PMIX NODE MAP regular
14
                     expression for the job
15
               PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
16
                     Comma-delimited list of ranks on this node within the specified namespace - referenced
17
18
                     using PMIX_RANK_WILDCARD.
19
               PMIX LOCAL PROCS "pmix.lprocs" (pmix proc t array)
                     Array of pmix_proc_t of all processes on the specified node - referenced using
20
                     PMIX RANK WILDCARD.
21
22
               PMIX LOCAL CPUSETS "pmix.lcpus" (char*)
23
                     Colon-delimited cpusets of local peers within the specified namespace - referenced using
                     PMIX RANK WILDCARD.
24
               PMIX_PROC_URI "pmix.puri" (char*)
25
26
                     URI containing contact information for a given process.
27
               PMIX_LOCALITY "pmix.loc" (uint16_t)
                     Relative locality of the specified process to the requestor.
28
29
               PMIX PARENT ID "pmix.parent" (pmix proc t)
30
                     Process identifier of the parent process of the calling process.
   3.4.10 Information retrieval attributes
32
               The following attributes are used to specify the level of information (e.g., session, job, or
33
               application) being requested where ambiguity may exist - see 5.1.5 for examples of their use.
34
               PMIX SESSION INFO "pmix.ssn.info" (bool)
                     Return information about the specified session. If information about a session other than the
35
36
                     one containing the requesting process is desired, then the attribute array must contain a
37
                     PMIX SESSION ID attribute identifying the desired target.
38
               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 <code>PMIX\_JOBID</code> or <code>PMIX\_NSPACE</code> 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)

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

# 3.4.11 Information storage attributes

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

PMIX\_SESSION\_INFO\_ARRAY "pmix.ssn.arr" (pmix\_data\_array\_t)

Provide an array of pmix\_info\_t containing session-level information. The

PMIX\_SESSION\_ID attribute is required to be included in the array.

PMIX\_JOB\_INFO\_ARRAY "pmix.job.arr" (pmix\_data\_array\_t)

Provide an array of **pmix\_info\_t** containing job-level information. Information is registered one job (aka namespace) at a time via the **PMIx\_server\_register\_nspace** API. Thus, there is no requirement that the array contain either the **PMIX\_NSPACE** or **PMIX\_JOBID** attributes, though either or both of them *may* be included.

PMIX\_APP\_INFO\_ARRAY "pmix.app.arr" (pmix\_data\_array\_t)

Provide an array of **pmix\_info\_t** containing app-level information. The **PMIX\_NSPACE** or **PMIX\_JOBID** attributes of the **job** containing the appplication, plus its **PMIX\_APPNUM** attribute, are *required* to be included in the array.

PMIX\_NODE\_INFO\_ARRAY "pmix.node.arr" (pmix\_data\_array\_t)

Provide an array of **pmix\_info\_t** containing node-level information. At a minimum, either the **PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute is *required* to be included in the array, though both *may* be included.

#### 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
1
2
                     executing process. Note that this attribute is the equivalent to the combination of
                     PMIX SESSION INFO ARRAY with the PMIX NUM SLOTS entry in the array - it is
 3
4
                     included in the Standard for historical reasons.
5
               PMIX_JOB_SIZE "pmix.job.size" (uint32_t)
6
                     Total number of processes in this job across all contained applications
7
               PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)
                     Number of applications in this job.
8
9
               PMIX APP SIZE "pmix.app.size" (uint32 t)
                     Number of processes in this application.
10
               PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)
11
12
                     Number of processes in this job on this node.
               PMIX_NODE_SIZE "pmix.node.size" (uint32_t)
13
                     Number of processes across all jobs on this node.
14
               PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
15
                     Maximum number of processes for this job.
16
17
               PMIX NUM NODES "pmix.num.nodes" (uint32 t)
                     Number of nodes in this session or namespace.
18
19
               PMIX NUM SLOTS "pmix.num.slots" (uint32 t)
20
                     Number of slots allocated to the session, namespace, or application.
```

## 3.4.13 Memory information attributes

25

26 27

28

29

33

34

35

36 37

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

PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
```

```
Total available physical memory on this node.
```

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

Megabytes of memory currently used by the RM daemon.

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

Average Megabytes of memory used by client processes.

## 30 3.4.14 Topology information attributes

These attributes are used to describe topology information in the PMIx universe - all are referenced using PMIX\_RANK\_WILDCARD except where noted.

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

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

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

XML representation of local node topology.

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

```
1
                     Comma-delimited list of nodes running processes for this job.
2
               PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
3
                     Pointer to the PMIx client's internal hwloc topology object.
 4
               PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
5
                     Topology signature string.
6
               PMIX LOCALITY STRING "pmix.locstr" (char*)
7
                     String describing a process's bound location - referenced using the process's rank. The string
                     is of the form:
8
9
                     NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s
10
                     Where the \$s is replaced with an integer index or inclusive range for hwloc. NM identifies
                     the numa node(s). SK identifies the socket(s). L3 identifies the L3 cache(s). L2 identifies the
11
                     L2 cache(s). L1 identifies the L1 cache(s). CR identifies the cores(s). HT identifies the
12
13
                     hardware thread(s). If your architecture does not have the specified hardware designation
14
                     then it can be omitted from the signature.
15
                     For example: NM0: SK0: L30-4: L20-4: L10-4: CR0-4: HT0-39.
16
                     This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches
                     0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.
17
               PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
18
                     Address of the hwloc shared memory segment.
19
20
               PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
21
                     Size of the hwloc shared memory segment.
22
               PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
23
                     Path to the hwloc shared memory file.
               PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
24
                     XML representation of local topology using hwloc's v1.x format.
25
26
               PMIX HWLOC XML V2 "pmix.hwlocxml2" (char*)
27
                     XML representation of local topology using hwloc's v2.x format.
               Request-related attributes
   3.4.15
29
               These attributes are used to influence the behavior of various PMIx operations - they do not
30
               represent values accessed using the PMIx Get API.
31
```

```
PMIX COLLECT DATA "pmix.collect" (bool)
```

Collect data and return it at the end of the 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_IMMEDIATE "pmix.immediate" (bool)
```

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.

```
PMIX_WAIT "pmix.wait" (int)
```

32

33

34

35

36

37 38

39

```
1
                     Caller requests that the PMIx server wait until at least the specified number of values are
2
                     found (0 indicates all and is the default).
3
               PMIX COLLECTIVE ALGO "pmix.calgo" (char*)
4
                     Comma-delimited list of algorithms to use for the collective operation. PMIx does not
                     impose any requirements on a host environment's collective algorithms. Thus, the
5
                     acceptable values for this attribute will be environment-dependent - users are encouraged to
6
7
                     check their host environment for supported values.
8
               PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
9
                     If true, indicates that the requested choice of algorithm is mandatory.
               PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
10
                     Notify the parent process upon termination of child job.
11
12
               PMIX RANGE "pmix.range" (pmix data range t)
                     Value for calls to publish/lookup/unpublish or for monitoring event notifications.
13
               PMIX PERSISTENCE "pmix.persist" (pmix persistence t)
14
                     Value for calls to PMIx_Publish.
15
               PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
16
                     Scope of the data to be found in a PMIx_Get call.
17
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.
20
21
               PMIX EMBED BARRIER "pmix.embed.barrier" (bool)
22
                     Execute a blocking fence operation before executing the specified operation. For example,
23
                     PMIx Finalize does not include an internal barrier operation by default. This attribute
                     would direct PMIx Finalize to execute a barrier as part of the finalize operation.
24
               PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)
25
26
                     Status to be returned upon job termination.
27
               PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)
                     Process state
28
               Server-to-PMIx library attributes
   3.4.16
30
               Attributes used by the host environment to pass data to its PMIx server library. The data will then
               be parsed and provided to the local PMIx clients. These attributes are all referenced using
31
               PMIX RANK WILDCARD except where noted.
32
33
               PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
                     Registration is for this namespace only, do not copy job data - this attribute is not accessed
34
                     using the PMIx Get
35
36
               PMIX PROC DATA "pmix.pdata" (pmix data array t)
37
                     Array of process data. Starts with rank, then contains more data.
38
               PMIX_NODE_MAP "pmix.nmap" (char*)
                     Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.
39
               PMIX_PROC_MAP "pmix.pmap" (char*)
40
```

```
1
                    Regular expression describing processes on each node - see 10.1.3.1 for an explanation of its
2
                    generation.
               PMIX ANL MAP "pmix.anlmap" (char*)
 3
 4
                    Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
               PMIX_APP_MAP_TYPE "pmix.apmap.type" (char*)
5
6
                    Type of mapping used to layout the application (e.g., cyclic).
7
               PMIX APP MAP REGEX "pmix.apmap.regex" (char*)
8
                    Regular expression describing the result of the process mapping.
   3.4.17 Server-to-Client attributes
               Attributes used internally to communicate data from the PMIx server to the PMIx client - they do
10
11
               not represent values accessed using the PMIx_Get API.
12
               PMIX_PROC_BLOB "pmix.pblob" (pmix_byte_object_t)
                    Packed blob of process data.
13
14
               PMIX MAP BLOB "pmix.mblob" (pmix byte object t)
15
                    Packed blob of process location.
16 3.4.18 Event handler registration and notification attributes
17
               Attributes to support event registration and notification - they are values passed to the event
               registration and notification APIs and therefore are not accessed using the PMIx_Get API.
18

    Advice to users

19
               The event handler subsystem defined in the PMIx ad hoc version 1 Standard was completely
               overhauled in version 2 to resolve design flaws. Deprecated attributes shown below were therefore
20
               removed in the version 2 Standard.
21
22
               PMIX ERROR NAME "pmix.errname" (pmix status t)
23
                    Specific error to be notified
               PMIX_ERROR_GROUP_COMM "pmix.errgroup.comm" (bool)
24
25
                    Set true to get comm errors notification
26
               PMIX_ERROR_GROUP_ABORT "pmix.errgroup.abort" (bool)
27
                    Set true to get abort errors notification
               PMIX_ERROR_GROUP_MIGRATE "pmix.errgroup.migrate" (bool)
28
                    Set true to get migrate errors notification
29
               PMIX ERROR GROUP RESOURCE "pmix.errgroup.resource" (bool)
30
                    Set true to get resource errors notification
31
32
               PMIX_ERROR_GROUP_SPAWN "pmix.errgroup.spawn" (bool)
33
                    Set true to get spawn errors notification
```

PMIX\_ERROR\_GROUP\_NODE "pmix.errgroup.node" (bool)

```
1
                    Set true to get node status notification
              PMIX ERROR_GROUP_LOCAL "pmix.errgroup.local" (bool)
2
                    Set true to get local errors notification
3
4
              PMIX_ERROR_GROUP_GENERAL "pmix.errgroup.gen" (bool)
                    Set true to get notified of generic errors
5
              PMIX_ERROR_HANDLER_ID "pmix.errhandler.id" (int)
6
7
                    Errhandler reference id of notification being reported
8
              PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)
                    String name identifying this handler.
9
              PMIX EVENT HDLR FIRST "pmix.evfirst" (bool)
10
                    Invoke this event handler before any other handlers.
11
              PMIX EVENT HDLR LAST "pmix.evlast" (bool)
12
                    Invoke this event handler after all other handlers have been called.
13
14
              PMIX EVENT HDLR FIRST IN CATEGORY "pmix.evfirstcat" (bool)
                    Invoke this event handler before any other handlers in this category.
15
              PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
16
17
                    Invoke this event handler after all other handlers in this category have been called.
              PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)
18
                    Put this event handler immediately before the one specified in the (char*) value.
19
              PMIX EVENT HDLR AFTER "pmix.evafter" (char*)
20
21
                    Put this event handler immediately after the one specified in the (char*) value.
22
              PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)
                    Prepend this handler to the precedence list within its category.
23
              PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)
24
                    Append this handler to the precedence list within its category.
25
              PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)
26
                    Array of pmix proc t defining range of event notification.
27
28
              PMIX EVENT AFFECTED PROC "pmix.evproc" (pmix proc t)
29
                    The single process that was affected.
30
              PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)
                    Array of pmix_proc_t defining affected processes.
31
              PMIX_EVENT_NON_DEFAULT "pmix.evnondef" (bool)
32
                    Event is not to be delivered to default event handlers.
33
              PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
34
35
                    Object to be returned whenever the registered callback function cbfunc is invoked. The
36
                    object will only be returned to the process that registered it.
              PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
37
                    Instruct the PMIx server not to cache the event.
38
              PMIX EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
39
                    Do not generate an event when this job normally terminates.
40
```

### 1 3.4.19 Fault tolerance attributes

```
2
               Attributes to support fault tolerance behaviors - they are values passed to the event notification API
 3
               and therefore are not accessed using the PMIx Get API.
 4
               PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)
5
                    The RM intends to terminate this session.
6
               PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
7
                    The RM intends to terminate this job.
8
               PMIX EVENT TERMINATE NODE "pmix.evterm.node" (bool)
9
                    The RM intends to terminate all processes on this node.
10
               PMIX EVENT TERMINATE PROC "pmix.evterm.proc" (bool)
                    The RM intends to terminate just this process.
11
12
               PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)
13
                    The time in seconds before the RM will execute error response.
14
               PMIX EVENT NO TERMINATION "pmix.evnoterm" (bool)
15
                    Indicates that the handler has satisfactorily handled the event and believes termination of the
16
                    application is not required.
17
               PMIX EVENT WANT TERMINATION "pmix.evterm" (bool)
18
                    Indicates that the handler has determined that the application should be terminated
```

## 19 3.4.20 Spawn attributes

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

```
27
              PMIX PERSONALITY "pmix.pers" (char*)
28
                    Name of personality to use.
29
              PMIX_HOST "pmix.host" (char*)
                    Comma-delimited list of hosts to use for spawned processes.
30
31
              PMIX_HOSTFILE "pmix.hostfile" (char*)
32
                    Hostfile to use for spawned processes.
              PMIX ADD HOST "pmix.addhost" (char*)
33
                    Comma-delimited list of hosts to add to the allocation.
34
              PMIX ADD HOSTFILE "pmix.addhostfile" (char*)
35
                    Hostfile listing hosts to add to existing allocation.
36
37
              PMIX_PREFIX "pmix.prefix" (char*)
38
                    Prefix to use for starting spawned processes.
39
              PMIX WDIR "pmix.wdir" (char*)
```

20 21

22

23

24 25

1	Working directory for spawned processes.
2	<pre>PMIX_MAPPER "pmix.mapper" (char*)</pre>
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 <b>PMIx_Get</b> , 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 <b>PMIx_Get</b> , 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 <b>PMIx_Get</b> , use the
20	PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
21	provided namespace
22	PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
23	Preload binaries onto nodes.
24	<pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)</pre>
25	Comma-delimited list of files to pre-position on nodes.
26	PMIX_NON_PMI "pmix.nonpmi" (bool)
27	Spawned processes will not call <b>PMIx_Init</b> .
28	PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
29	Spawned process rank that is to receive <b>stdin</b> .
30	PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)
31	Forward this process's <b>stdin</b> to the designated process.
32	PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
33	Forward <b>stdout</b> from spawned processes to this process.
34	PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
35	Forward <b>stderr</b> from spawned processes to this process.
36	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
37	Spawned application consists of debugger daemons.
38	PMIX_COSPAWN_APP "pmix.cospawn" (bool)
39	Designated application is to be spawned as a disconnected job. Meaning that it is not part of
40	the "comm_world" of the parent process.
41	PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)

```
1
                    Set the application's current working directory to the session working directory assigned by
2
                    the RM - when accessed using PMIx Get, use the PMIX RANK WILDCARD value for
                    the rank to discover the session working directory assigned to the provided namespace
3
4
              PMIX TAG OUTPUT "pmix.tagout" (bool)
                    Tag application output with the identity of the source process.
5
              PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
6
7
                    Timestamp output from applications.
8
              PMIX MERGE STDERR STDOUT "pmix.mergeerrout" (bool)
9
                    Merge stdout and stderr streams from application processes.
              PMIX OUTPUT TO FILE "pmix.outfile" (char*)
10
                    Output application output to the specified file.
11
              PMIX_INDEX_ARGV "pmix.indxargv" (bool)
12
                    Mark the argv with the rank of the process.
13
              PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
14
                    Number of cpus to assign to each rank - when accessed using PMIx Get, use the
15
                    PMIX RANK WILDCARD value for the rank to discover the cpus/process assigned to the
16
17
                    provided namespace
              PMIX NO PROCS ON HEAD "pmix.nolocal" (bool)
18
19
                    Do not place processes on the head node.
              PMIX NO OVERSUBSCRIBE "pmix.noover" (bool)
20
21
                    Do not oversubscribe the cpus.
22
              PMIX REPORT BINDINGS "pmix.repbind" (bool)
                    Report bindings of the individual processes.
23
              PMIX_CPU_LIST "pmix.cpulist" (char*)
24
25
                    List of cpus to use for this job - when accessed using PMIx Get, use the
26
                    PMIX RANK WILDCARD value for the rank to discover the cpu list used for the provided
27
                    namespace
              PMIX JOB RECOVERABLE "pmix.recover" (bool)
28
29
                    Application supports recoverable operations.
30
              PMIX JOB CONTINUOUS "pmix.continuous" (bool)
                    Application is continuous, all failed processes should be immediately restarted.
31
              PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
32
33
                    Maximum number of times to restart a job - when accessed using PMIx_Get, use the
                    PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
34
35
                    namespace
               Query attributes
36 3.4.21
37
              Attributes used to describe PMIx Query_info_nb behavior - these are values passed to the
              PMIx_Query_info_nb API and therefore are not passed to the PMIx_Get API.
38
39
              PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
40
                    Retrieve updated information from server.
```

```
1
              PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)
2
                    Request a comma-delimited list of active namespaces.
              PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)
3
4
                    Status of a specified, currently executing job.
              PMIX QUERY QUEUE LIST "pmix.qry.qlst" (char*)
5
6
                    Request a comma-delimited list of scheduler queues.
7
              PMIX QUERY QUEUE STATUS "pmix.gry.gst" (TBD)
8
                    Status of a specified scheduler queue.
              PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
9
                    Input namespace of the job whose information is being requested returns (
10
                    pmix_data_array_t ) an array of pmix_proc_info_t .
11
              PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
12
                    Input namespace of the job whose information is being requested returns (
13
14
                    pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same
                    node.
15
              PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
16
17
                    Constrain the query to local information only.
18
              PMIX QUERY AUTHORIZATIONS "pmix.gry.auths" (bool)
19
                    Return operations the PMIx tool is authorized to perform.
              PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
20
                    Return a comma-delimited list of supported spawn attributes.
21
22
              PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
23
                    Return a comma-delimited list of supported debug attributes.
              PMIX QUERY MEMORY USAGE "pmix.gry.mem" (bool)
24
25
                    Return information on memory usage for the processes indicated in the qualifiers.
26
              PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
27
                    Report average values.
              PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
28
                    Report minimum and maximum values.
29
              PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
30
                    String identifier of the allocation whose status is being requested.
31
32
              PMIX TIME REMAINING "pmix.time.remaining" (char*)
33
                    Query number of seconds (uint32 t) remaining in allocation for the specified namespace.
34 3.4.22 Log attributes
              Attributes used to describe PMIx_Log_nb behavior - these are values passed to the
35
              PMIx Log nb API and therefore are not accessed using the PMIx Get API.
36
              PMIX_LOG_STDERR "pmix.log.stderr" (char*)
37
38
                    Log string to stderr.
              PMIX LOG STDOUT "pmix.log.stdout" (char*)
39
```

40

Log string to **stdout**.

```
1
               PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
2
                    Log data to syslog. Defaults to ERROR priority.
 3
               PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
                    Message blob to be sent somewhere.
4
5
               PMIX LOG EMAIL "pmix.log.email" (pmix data array t)
6
                    Log via email based on pmix info t containing directives.
7
               PMIX LOG EMAIL ADDR "pmix.log.emaddr" (char*)
8
                    Comma-delimited list of email addresses that are to receive the message.
9
               PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
10
                    Subject line for email.
11
               PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
12
                    Message to be included in email.
   3.4.23 Debugger attributes
13
               Attributes used to assist debuggers - these are values that can be passed to the PMIx_Spawn or
14
15
               PMIx Init APIs. Some may be accessed using the PMIx Get API with the
16
               PMIX RANK WILDCARD rank.
17
               ====== PMIX DEBUG STOP ON EXEC "pmix.dbg.exec" (bool)
                    Passed to PMIx Spawn to indicate that the specified application is being spawned under
18
19
                    debugger, and that the launcher is to pause the resulting application processes on first
20
                    instruction for debugger attach.
21
               PMIX DEBUG STOP IN INIT "pmix.dbg.init" (bool)
22
                    Passed to PMIx_Spawn to indicate that the specified application is being spawned under
23
                    debugger, and that the PMIx client library is to pause the resulting application processes
                    during PMIx Init until debugger attach and release.
24
25
               PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool)
                    Passed to PMIx_Spawn to indicate that the specified application is being spawned under
26
27
                    debugger, and that the resulting application processes are to pause at some
                    application-determined location until debugger attach and release.
28
               PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
29
                    Namespace of the job to be debugged - provided to the debugger upon launch.
30
```

PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY "pmix.dbg.waiting" (bool)

# 3.4.24 Resource manager attributes

PMIx Get API.

Attributes used to describe the RM - these are values assigned by the host environment and accessed using the <code>PMIx\_Get</code> API. The value of the provided namespace is unimportant but should be given as the namespace of the requesting process and a rank of <code>PMIX\_RANK\_WILDCARD</code> used to indicate that the information will be found with the job-level information.

Job to be debugged is waiting for a release - this is not a value accessed using the

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

31 32

33

35

36

37

38

```
1
                    String name of the RM.
2
              PMIX_RM_VERSION "pmix.rm.version" (char*)
                    RM version string.
 3
4 3.4.25 Environment variable attributes
              Attributes used to adjust environment variables - these are values passed to the PMIx Spawn API
5
6
              and are not accessed using the PMIx Get API.
              PMIX SET_ENVAR "pmix.set.envar" (char*)
7
8
                    String "key=value" value shall be put into the environment.
9
              PMIX UNSET ENVAR "pmix.unset.envar" (char*)
10
                    Unset the environment variable specified in the string.
11 3.4.26 Job Allocation attributes
12
              Attributes used to describe the job allocation - these are values passed to the
13
              PMIx Allocation request nb API and are not accessed using the PMIx Get API
14
              PMIX_ALLOC_ID "pmix.alloc.id" (char*)
15
                    Provide a string identifier for this allocation request which can later be used to query status
                    of the request.
16
              PMIX ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
17
                    The number of nodes.
18
19
              PMIX ALLOC NODE LIST "pmix.alloc.nlist" (char*)
20
                    Regular expression of the specific nodes.
              PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
21
22
                    Number of cpus.
23
              PMIX ALLOC NUM CPU LIST "pmix.alloc.ncpulist" (char*)
                    Regular expression of the number of cpus for each node.
24
              PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
25
26
                    Regular expression of the specific cpus indicating the cpus involved.
27
              PMIX ALLOC MEM SIZE "pmix.alloc.msize" (float)
                    Number of Megabytes.
28
              PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
29
                    Array of pmix info t describing requested network resources. If not given as part of an
30
31
                    pmix info t struct that identifies the involved nodes, then the description will be
32
                    applied across all nodes in the requestor's allocation.
              PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
33
34
                    Name of the network.
              PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
35
```

PMIX ALLOC NETWORK QOS "pmix.alloc.netgos" (char\*)

36

37

Mbits/sec.

```
1 Quality of service level.
2 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
3 Time in seconds.
```

### 4 3.4.27 Job control attributes

```
Attributes used to request control operations on an executing application - these are values passed
5
6
              to the PMIx_Job_control_nb API and are not accessed using the PMIx_Get API.
7
              PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
8
                    Provide a string identifier for this request.
9
              PMIX JOB CTRL PAUSE "pmix.jctrl.pause" (bool)
10
                    Pause the specified processes.
              PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
11
                    Resume ("un-pause") the specified processes.
12
13
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
                    Cancel the specified request (NULL implies cancel all requests from this requestor).
14
              PMIX JOB CTRL KILL "pmix.jctrl.kill" (bool)
15
16
                    Forcibly terminate the specified processes and cleanup.
              PMIX JOB CTRL RESTART "pmix.jctrl.restart" (char*)
17
                    Restart the specified processes using the given checkpoint ID.
18
              PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
19
                    Checkpoint the specified processes and assign the given ID to it.
20
              PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
21
22
                    Use event notification to trigger a process checkpoint.
              PMIX JOB CTRL CHECKPOINT SIGNAL "pmix.jctrl.ckptsig" (int)
23
24
                    Use the given signal to trigger a process checkpoint.
              PMIX JOB CTRL CHECKPOINT TIMEOUT "pmix.jctrl.ckptsig" (int)
25
26
                    Time in seconds to wait for a checkpoint to complete.
              PMIX JOB CTRL CHECKPOINT METHOD
27
              "pmix.jctrl.ckmethod" (pmix_data_array_t)
28
                    Array of pmix_info_t declaring each method and value supported by this application.
29
              PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
30
31
                    Send given signal to specified processes.
32
              PMIX JOB CTRL PROVISION "pmix.jctrl.pvn" (char*)
                    Regular expression identifying nodes that are to be provisioned.
33
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
34
35
                    Name of the image that is to be provisioned.
              PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
36
                    Indicate that the job can be pre-empted.
37
38
              PMIX JOB CTRL TERMINATE "pmix.jctrl.term" (bool)
39
                    Politely terminate the specified processes.
```

### 1 3.4.28 Monitoring attributes

```
2
              Attributes used to control monitoring of an executing application- these are values passed to the
 3
              PMIx_Process_monitor_nb API and are not accessed using the PMIx_Get API.
              PMIX_MONITOR_ID "pmix.monitor.id" (char*)
 4
5
                    Provide a string identifier for this request.
6
              PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
7
                    Identifier to be canceled (NULL means cancel all monitoring for this process).
8
              PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
9
                    The application desires to control the response to a monitoring event.
10
              PMIX MONITOR HEARTBEAT "pmix.monitor.mbeat" (void)
11
                    Register to have the PMIx server monitor the requestor for heartbeats.
              PMIX SEND_HEARTBEAT "pmix.monitor.beat" (void)
12
13
                    Send heartbeat to local PMIx server.
14
              PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
15
                    Time in seconds before declaring heartbeat missed.
16
              PMIX MONITOR HEARTBEAT DROPS "pmix.monitor.bdrop" (uint32 t)
17
                    Number of heartbeats that can be missed before generating the event.
18
              PMIX MONITOR FILE "pmix.monitor.fmon" (char*)
                    Register to monitor file for signs of life.
19
              PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
20
21
                    Monitor size of given file is growing to determine if the application is running.
22
              PMIX MONITOR FILE ACCESS "pmix.monitor.faccess" (char*)
                    Monitor time since last access of given file to determine if the application is running.
23
              PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
24
25
                    Monitor time since last modified of given file to determine if the application is running.
26
              PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
27
                    Time in seconds between checking the file.
28
              PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
29
                    Number of file checks that can be missed before generating the event.
```

### 30 3.5 Callback Functions

31

32 33 PMIx provides blocking and nonblocking versions of most APIs. In the nonblocking versions, a callback is activated upon completion of the the operation. This section describes many of those callbacks.

### 3.5.1 Release Callback Function

```
Summary
2
 3
              The pmix release cbfunc t is used by the pmix modex cbfunc t and
 4
              pmix info cbfunc t operations to indicate that the callback data may be reclaimed/freed by
 5
              the caller.
              Format
6
   PMIx v1.0
 7
              typedef void (*pmix_release_cbfunc_t)
8
                   (void *cbdata)
9
              INOUT cbdata
                    Callback data passed to original API call (memory reference)
10
              Description
11
              Since the data is "owned" by the host server, provide a callback function to notify the host server
12
              that we are done with the data so it can be released.
13
             Modex Callback Function
14 3.5.2
15
              Summary
16
              The pmix_modex_cbfunc_t is used by the pmix_server_fencenb_fn_t and
17
              pmix_server_dmodex_req_fn_t PMIx server operations to return modex BCX data.
   PMIx v1.0
18
              typedef void (*pmix_modex_cbfunc_t)
19
                   (pmix_status_t status,
                    const char *data, size_t ndata,
20
                    void *cbdata,
21
22
                    pmix_release_cbfunc_t release_fn,
                    void *release cbdata)
23
              IN
                   status
24
25
                    Status associated with the operation (handle)
              IN
26
                   data
27
                    Data to be passed (pointer)
```

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

### Description

9

10 11

12

14

15

16

20

21

22

23

24

25

26 27

28

29

30

31 32 A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to "fence" and "get" operations. The returned blob contains the data collected from each server participating in the operation.

### 3 3.5.3 Spawn Callback Function

#### Summary

The pmix\_spawn\_cbfunc\_t is used on the PMIx client side by PMIx\_Spawn\_nb and on the PMIx server side by pmix\_server\_spawn\_fn\_t.

```
PMIx v1.0

typedef void (*pmix_spawn_cbfunc_t)

(pmix_status_t status,
pmix_nspace_t nspace, void *cbdata);
```

IN statusStatus associated with the operation (handle)IN nspace

Namespace string ( pmix\_nspace\_t )

IN cbdata

Callback data passed to original API call (memory reference)

### Description

The callback will be executed upon launch of the specified applications in **PMIx\_Spawn\_nb**, or upon failure to launch any of them.

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

## 3.5.4 Op Callback Function

```
Summary
2
              The pmix op cbfunc t is used by operations that simply return a status.
 3
   PMIx v1.0
4
              typedef void (*pmix op cbfunc t)
5
                   (pmix status t status, void *cbdata);
                       ______ C
              IN
                  status
6
 7
                   Status associated with the operation (handle)
8
              IN
                   cbdata
9
                   Callback data passed to original API call (memory reference)
10
              Description
11
              Used by a wide range of PMIx API's including PMIx_Fence_nb,
12
              pmix_server_client_connected_fn_t, PMIx_server_register_nspace. This
              callback function is used to return a status to an often nonblocking operation.
13
            Lookup Callback Function
   3.5.5
15
              Summary
              The pmix lookup cbfunc t is used by PMIx Lookup nb to return data.
16
   PMIx v1.0
17
              typedef void (*pmix lookup cbfunc t)
                   (pmix_status_t status,
18
19
                    pmix_pdata_t data[], size_t ndata,
                    void *cbdata);
20
              IN
21
                  status
22
                   Status associated with the operation (handle)
              IN
23
                   Array of data returned ( pmix_pdata_t )
24
              IN
                   ndata
25
26
                   Number of elements in the data array (size_t)
              IN
27
                   Callback data passed to original API call (memory reference)
28
```

### Description

1 2

3

4

5

6

7

9

14

15

16 17

18

19

20 21

22

23

25

26

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.

\_\_\_\_ C \_\_\_\_

#### Value Callback Function 3.5.6

### Summary

10 The pmix\_value\_cbfunc\_t is used by PMIx\_Get\_nb to return data.

```
PMIx v1.0
11
            typedef void (*pmix_value_cbfunc_t)
12
                 (pmix_status_t status,
                 pmix_value_t *kv, void *cbdata);
13
```

IN status Status associated with the operation (handle) IN

Key/value pair representing the data (pmix value t)

IN cbdata

Callback data passed to original API call (memory reference)

### **Description**

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

#### Info Callback Function 3.5.7

### Summary

The **pmix** info **cbfunc** t is a general information callback used by various APIs.

PMIx v2.0

```
1
               typedef void (*pmix info cbfunc t)
2
                    (pmix_status_t status,
 3
                     pmix_info_t info[], size_t ninfo,
 4
                     void *cbdata,
 5
                     pmix_release_cbfunc_t release_fn,
                     void *release cbdata);
6
7
               IN
                    status
                    Status associated with the operation ( pmix_status_t )
8
9
               IN
                    Array of pmix info t returned by the operation (pointer)
10
               IN
11
                    Number of elements in the info array (size t)
12
13
               IN
                    cbdata
14
                    Callback data passed to original API call (memory reference)
               IN
                    release fn
15
                    Function to be called when done with the info data (function pointer)
16
               IN
                    release_cbdata
17
18
                     Callback data to be passed to release fn (memory reference)
               Description
19
20
               The status indicates if requested data was found or not. An array of pmix info t will contain
21
               the key/value pairs.
             Event Handler Registration Callback Function
    3.5.8
23
               The pmix evhdlr reg cbfunc t callback function.
                                                Advice to users -
               The PMIx ad hoc v1.0 Standard defined an error handler registration callback function with a
24
               compatible signature, but with a different type definition function name
25
               (pmix errhandler reg cbfunc t). It was removed from the v2.0 Standard and is not included in this
26
               document to avoid confusion.
27
```

PMIx v2.0

```
1
              typedef void (*pmix evhdlr reg cbfunc t)
 2
                   (pmix_status_t status,
 3
                    size_t evhdlr_ref,
                    void *cbdata)
 4
              IN
                 status
5
                   Status indicates if the request was successful or not ( pmix_status_t )
6
7
              IN
                   evhdlr ref
                   Reference assigned to the event handler by PMIx — this reference * must be used to
8
                   deregister the err handler (size_t)
9
              IN
                   cbdata
10
                   Callback data passed to original API call (memory reference)
11
              Description
12
              Define a callback function for calls to PMIx_Register_event_handler
13
   3.5.9
             Notification Handler Completion Callback Function
15
              Summary
16
              The pmix_event_notification_cbfunc_fn_t is called by event handlers to indicate
17
              completion of their operations.
   PMIx v2.0
18
              typedef void (*pmix_event_notification_cbfunc_fn_t)
                   (pmix_status_t status,
19
                    pmix_info_t *results, size_t nresults,
20
                    pmix_op_cbfunc_t cbfunc, void *thiscbdata,
21
                    void *notification cbdata);
22
              IN status
23
                   Status returned by the event handler's operation ( pmix_status_t )
24
              IN
25
26
                   Results from this event handler's operation on the event (pmix info t)
27
              IN
                   nresults
28
                   Number of elements in the results array (size t)
              IN
29
30
                   pmix op cbfunc t function to be executed when PMIx completes processing the
31
                   callback (function reference)
```

IN 1 thischdata 2 Callback data that was passed in to the handler (memory reference) 3 IN cbdata Callback data to be returned when PMIx executes cbfunc (memory reference) 4 5 Description 6 Define a callback by which an event handler can notify the PMIx library that it has completed its 7 response to the notification. The handler is required to execute this callback so the library can determine if additional handlers need to be called. The handler shall return 8 9 PMIX ERR EVENT COMPLETE if no further action is required. The return status of each event 10 handler and any returned **pmix info** t structures will be added to the *results* array of 11 pmix\_info\_t passed to any subsequent event handlers to help guide their operation. 12 If non-NULL, the provided callback function will be called to allow the event handler to release the 13 provided info array and execute any other required cleanup operations. 3.5.10 Notification Function Summary 15 16 The pmix notification fn t is called by PMIx to deliver notification of an event. Advice to users The PMIx ad hoc v1.0 Standard defined an error notification function with an identical name, but 17 different signature than the v2.0 Standard described below. The ad hoc v1.0 version was removed 18 from the v2.0 Standard is not included in this document to avoid confusion. 19 PMIx v2.0 20 typedef void (\*pmix\_notification\_fn\_t) (size t evhdlr registration id, 21 22 pmix status t status,

23

24

25

26 27 const pmix\_proc\_t \*source,

void \*cbdata);

pmix\_info\_t info[], size\_t ninfo,

pmix\_info\_t results[], size\_t nresults,

pmix\_event\_notification\_cbfunc\_fn\_t cbfunc,

	1	-	٦
	١		J
		Ī	

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

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.

## 12 3.5.11 Server Setup Application Callback Function

13 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.

#### **Format** 1 PMIx v2.0 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) 7 IN status 8 returned status of the request ( **pmix\_status\_t** ) 9 IN 10 Array of info structures (array of handles) 11 IN ninfo Number of elements in the *info* array (integer) 12 13 IN provided cbdata 14 Data originally passed to call to PMIx server setup application (memory reference) 15 IN cbfunc 16 17 pmix\_op\_cbfunc\_t function to be called when processing completed (function 18 reference) IN cbdata 19 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. 3.5.12 Server Direct Modex Response Callback Function 26 The PMIx server dmodex request callback function. Summary 27

Provide a function by which the local PMIx server library can return connection and other data

posted by local application processes to the host resource manager.

28

```
Format
 1
   PMIx v1.0
 2
               typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
 3
                                              char *data, size t sz,
 4
                                              void *cbdata);
               IN
 5
                    status
 6
                    Returned status of the request ( pmix_status_t )
 7
               IN
                    data
 8
                    Pointer to a data "blob" containing the requested information (handle)
 9
               IN
10
                    Number of bytes in the data blob (integer)
11
               IN
                    Data passed into the initial call to PMIx server dmodex request (memory
12
13
               Description
14
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
               from the host RM. The returned data blob is owned by the PMIx server library and will be free'd
17
18
               upon return from the function.
    3.5.13
               pmix connection cbfunc t
               Summary
20
21
               Callback function for incoming connection request from a local client
               Format
22
   PMIx v1.0
23
               typedef void (*pmix connection cbfunc t) (
                                                    int incoming_sd, void *cbdata)
24
25
               IN
                    incoming_sd
26
                    (integer)
27
               IN
                    cbdata
28
                    (memory reference)
```

### 1 Description

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

### 3.5.14 pmix tool connection cbfunc t

#### 5 Summary

Callback function for incoming tool connections.

#### Format

7

11

12

13

14

15

16

17

18 19

20

21

23

24

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

#### Description

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

### ------ Advice to PMIx server hosts ------

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

## 22 3.5.15 Constant String Functions

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

```
Summary
1
 2
             String representation of a pmix status t.
   PMIx v1.0
 3
             const char*
4
             PMIx_Error_string(pmix_status_t status);
             Summary
5
             String representation of a pmix_proc_state_t.
   PMIx v2.0
7
             const char*
             PMIx_Proc_state_string(pmix_proc_state_t state);
8
             Summary
9
             String representation of a pmix_scope_t.
10
   PMIx v2.0
11
             const char*
12
             PMIx_Scope_string(pmix_scope_t scope);
             Summary
13
             String representation of a pmix_persistence_t.
14
   PMIx v2.0
15
             const char*
             PMIx_Persistence_string(pmix_persistence_t persist);
16
17
             Summary
             String representation of a pmix_data_range_t.
18
   PMIx v2.0
19
             const char*
             PMIx_Data_range_string(pmix_data_range_t range);
20
```

```
Summary
1
2
           String representation of a pmix_info_directives_t.
  PMIx v2.0
3
           const char*
4
           PMIx_Info_directives_string(pmix_info_directives_t directives);
           Summary
5
           String representation of a pmix_data_type_t.
  PMIx v2.0
7
           const char*
8
           PMIx_Data_type_string(pmix_data_type_t type);
           Summary
9
10
           String representation of a pmix_alloc_directive_t.
  PMIx v2.0
11
           const char*
12
           PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);
```

#### **CHAPTER 4**

# Initialization and Finalization

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

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

#### Advice to users —

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

# **4.1 Query**

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

### 17 4.1.1 PMIx Initialized

Format

PMIx v1.0

int PMIx\_Initialized(void)

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

Rationale

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

### **Description** 1 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. 4.1.2 PMIx Get version Summary 5 Get the PMIx version information. **Format** 7 PMIx v1.0 8 const char\* PMIx Get version(void) **Description** 9 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 The PMIx ad hoc v1.0 Standard defined the PMIx\_Init function, but modified the function 14 signature in the v1.2 version. The ad hoc v1.0 version is not included in this document to avoid 15 confusion. 16 17 **4.2.1** PMIx Init 18 Summary

Initialize the PMIx client library

#### **Format** 1 *PMIx* v1.2 2 pmix status t 3 PMIx Init(pmix proc t \*proc, pmix\_info\_t info[], size\_t ninfo) 4 INOUT proc 5 6 proc structure (handle) 7 IN info Array of pmix info t structures (array of handles) 8 9 IN Number of element in the *info* array (size\_t) 10 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. 11 Optional Attributes The following attributes are optional for implementers of PMIx libraries: 12 13 PMIX\_USOCK\_DISABLE "pmix.usock.disable" (bool) 14 Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it. 15 16 PMIX\_SOCKET\_MODE "pmix.sockmode" (uint32\_t) POSIX mode t (9 bits valid) If the library supports socket connections, this attribute may 17 be supported for setting the socket mode. 18 PMIX SINGLE\_LISTENER "pmix.sing.listnr" (bool) 19 Use only one rendezvous socket, letting priorities and/or environment parameters select the 20 active transport. If the library supports multiple methods for clients to connect to servers, 21 22 this attribute may be supported for disabling all but one of them. 23 PMIX\_TCP\_REPORT\_URI "pmix.tcp.repuri" (char\*) If provided, directs that the TCP URI be reported and indicates the desired method of 24 reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket 25 26 connections, this attribute may be supported for reporting the URI. 27 PMIX TCP IF INCLUDE "pmix.tcp.ifinclude" (char\*) Comma-delimited list of devices and/or CIDR notation to include when establishing the 28 TCP connection. If the library supports TCP socket connections, this attribute may be 29 supported for specifying the interfaces to be used. 30 31 PMIX\_TCP\_IF\_EXCLUDE "pmix.tcp.ifexclude" (char\*) Comma-delimited list of devices and/or CIDR notation to exclude when establishing the 32 TCP connection. If the library supports TCP socket connections, this attribute may be 33 34 supported for specifying the interfaces that are *not* to be used.

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

### **Description**

19 20

21 22

23

24 25

26

27 28

29

30

31

32

33

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

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

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

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

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

Each call to **PMIx\_Init** may contain an array of **pmix\_info\_t** structures passing directives to 1 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. 4.2.2 PMIx Finalize Summary 6 7 Finalize the PMIx client library. **Format** 8 PMIx v1.0 9 pmix status t PMIx\_Finalize(const pmix\_info\_t info[], size\_t ninfo) 10 C IN info 11 12 Array of pmix\_info\_t structures (array of handles) ninfo 13 IN 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 The following attributes are optional for implementers of PMIx libraries: 16 PMIX\_EMBED\_BARRIER "pmix.embed.barrier" (bool) 17 Execute a blocking fence operation before executing the specified operation. For example, 18 PMIx Finalize does not include an internal barrier operation by default. This attribute 19 20 would direct **PMIx Finalize** to execute a barrier as part of the finalize operation. Description 21 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

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

```
Summary
              Initialize the PMIx library for operating as a tool.
 5
              Format
6
   PMIx v2.0
7
              pmix_status_t
              PMIx_tool_init(pmix_proc_t *proc,
8
9
                                 pmix_info_t info[], size_t ninfo)
                                             _____ C
              INOUT proc
10
                    pmix_proc_t structure (handle)
11
              IN
12
                   info
13
                    Array of pmix info t structures (array of handles)
              IN
14
15
                    Number of element in the info array (size t)
              Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
16
                                             Required Attributes
17
              The following attributes are required to be supported by all PMIx libraries:
              PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
18
19
                    Name of the namespace to use for this tool.
              PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
20
                    Rank of this tool.
21
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
1	The following attributes are optional for implementers of PMIx libraries:
2	PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)  The requestor requires that a connection be made only to a local, system-level PMIx server.
4 5	<pre>PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool) Preferentially, look for a system-level PMIx server first.</pre>
6 7	<pre>PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t) PID of the target PMIx server for a tool.</pre>
8 9 10	<pre>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 containing="" file="" it="" of="">.</name></pre>
11 12	<pre>PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t) Time in seconds between connection attempts to a PMIx server.</pre>
13 14	<pre>PMIX_CONNECT_MAX_RETRIES "pmix.tool.mretries" (uint32_t) Maximum number of times to try to connect to PMIx server.</pre>
15 16 17	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.
18 19 20 21	<pre>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.</pre>
22 23 24 25	<pre>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.</pre>
26 27 28 29	<pre>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.</pre>
30 31 32	<pre>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.</pre>
33 34	PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)  The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be

35

supported for specifying the port to be used.

#### PMIX\_TCP\_DISABLE\_IPV4 "pmix.tcp.disipv4" (bool) Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it. PMIX\_TCP\_DISABLE\_IPV6 "pmix.tcp.disipv6" (bool) Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it. PMIX\_EVENT\_BASE "pmix.evbase" (struct event\_base \*) Pointer to libevent<sup>2</sup> event base to use in place of the internal progress thread. PMIX\_GDS\_MODULE "pmix.gds.mod" (char\*) Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

#### Description

Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided **pmix\_proc\_t** struct. The *info* array is used to pass user requests pertaining to the init and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives are desired.

If called with the PMIX\_TOOL\_DO\_NOT\_CONNECT attribute, the PMIx tool library will fully initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to according to the following precedence chain:

- if PMIX\_SERVER\_URI or PMIX\_TCP\_URI is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified.
   PMIX\_SERVER\_URI is the preferred method as it is more generalized PMIX\_TCP\_URI is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if it isn't available or cannot succeed. The PMIx library will return an error if connection fails it will not proceed to check for other connection options as the user specified a particular one to use
- if PMIX\_SERVER\_PIDINFO was provided, then the tool will search under the directory provided by the PMIX\_SERVER\_TMPDIR environmental variable for a rendezvous file created by the process corresponding to that PID. The PMIx library will return an error if the rendezvous file cannot be found, or the connection is refused by the server

<sup>&</sup>lt;sup>2</sup>http://libevent.org/

- if PMIX\_CONNECT\_TO\_SYSTEM is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX\_SYSTEM\_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. An error is returned if the rendezvous file cannot be found or the connection is refused.
  - if PMIX\_CONNECT\_SYSTEM\_FIRST is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX\_SYSTEM\_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused the PMIx library will silently continue to the next option
  - by default, the tool will search the directory tree under the directory provided by the PMIX\_SERVER\_TMPDIR environmental variable for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error.

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

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

### 4.3.2 PMIx tool finalize

### Summary

Finalize the PMIx library for a tool connection.

#### Format

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

#### Description

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

### 1 4.4 Server Initialization and Finalization

2 Initialization and finalization routines for PMIx servers.

28

```
4.4.1
             PMIx server init
              Summary
 4
              Initialize the PMIx server.
 5
              Format
6
   PMIx v1.0
7
              pmix_status_t
              PMIx_server_init(pmix_server_module_t *module,
8
9
                                    pmix_info_t info[], size_t ninfo)
                                                        C —
              INOUT module
10
                    pmix server module t structure (handle)
11
              IN
                    info
12
13
                    Array of pmix info t structures (array of handles)
              IN
14
                   ninfo
15
                    Number of elements in the info array (size t)
16
              Returns PMIX SUCCESS or a negative value corresponding to a PMIx error constant.
                                              Required Attributes
17
              The following attributes are required to be supported by all PMIx libraries:
              PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
18
                    Name of the namespace to use for this PMIx server.
19
              PMIX SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
20
                    Rank of this PMIx server
21
22
              PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
23
                    Top-level temporary directory for all client processes connected to this server, and where the
                    PMIx server will place its tool rendezvous point and contact information.
24
25
              PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
                    Temporary directory for this system, and where a PMIx server that declares itself to be a
26
27
                    system-level server will place a tool rendezvous point and contact information.
```

PMIX SERVER\_TOOL\_SUPPORT "pmix.srvr.tool" (bool)

1	The host RM wants to declare itself as willing to accept tool connection requests.
2 3 4	PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)  The host RM wants to declare itself as being the local system server for PMIx connection requests.
	▼ Optional Attributes
5	The following attributes are optional for implementers of PMIx libraries:
6 7 8	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.
9 10 11	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.
12 13 14 15	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.
16 17 18 19	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.
20 21 22 23	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 <i>not</i> to be used.
24 25 26	<pre>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.</pre>
27 28 29	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.
30 31 32	PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)  Set to true to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.
33 34 35	PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)  Set to true to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX\_SERVER\_REMOTE\_CONNECTIONS "pmix.srvr.remote" (bool) 1 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 be supported for enabling or disabling it. 4 PMIX EVENT BASE "pmix.evbase" (struct event base \*) 5 Pointer to libevent<sup>3</sup> event base to use in place of the internal progress thread. 6 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 for interacting with the server is performed dynamically during the connection process. 10 Description 11 12 Initialize the PMIx server support library, and provide a pointer to a pmix server module t structure containing the caller's callback functions. The array of pmix info t structs is used to 13 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 -Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module* 17 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. 4.4.2 PMIx server finalize Summary 21 22 Finalize the PMIx server library. 23 **Format** PMIx v1.024 pmix\_status\_t PMIx server finalize(void) 25 26 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. <sup>3</sup>http://libevent.org/

## 1 Description

Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.

#### **CHAPTER 5**

1

3

4

5

6 7

8

9

10

14

15

26

# **Key/Value Management**

Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of the PMIx community is to eliminate collective operations, it is recognized that the traditional method of publishing/exchanging data must be supported until that objective can be met. This method relies on processes to discover and publish their local information which is collected by the local PMIx server library. Global exchange of the published information is then executed via a collective operation performed by the host SMS servers.

Keys are required to be unique within a specific level of informarion as defined in 3.4.10. For example, a value for **PMIX\_NUM\_NODES** can be specified for each of the **session**, **job**, and **application** levels. However, subsequently specifying another value for that attribute in the **session** level will overwrite the prior value.

# 5.1 Setting and Accessing Key/Value Pairs

### 2 5.1.1 PMIx Put

#### 13 Summary

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

#### Format

```
PMIx v1.0
16
              pmix status t
              PMIx_Put(pmix_scope_t scope,
17
18
                         const pmix_key_t key,
                         pmix_value_t *val)
19
              IN
20
                   Distribution scope of the provided value (handle)
21
22
              IN
23
                   key(pmix key t)
24
              IN
                   value
                    Reference to a pmix_value_t structure (handle)
25
```

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 <code>pmix\_scope\_t</code> values are defined in Section 3.2.9 on page 27. Specific implementations may support different scope values, but all implementations must support at least **PMIX\_GLOBAL**.

The <code>pmix\_value\_t</code> 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 <code>value</code> into internal memory.

# Advice to PMIx library implementers —

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

## Advice to users -

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

Note that keys starting with a string of "pmix" are exclusively reserved for the PMIx standard and must not be used in calls to  $pmix_put$ . Thus, applications should never use a defined "PMIX\_" attribute as the key in a call to  $pmix_put$ .

# **5.1.2** PMIx\_Get

# **Summary**

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

```
1
   PMIx v1.0
2
               pmix_status_t
 3
               PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
                          const pmix_info_t info[], size_t ninfo,
 4
5
                          pmix_value_t **val)
6
               IN
                    proc
7
                    process reference (handle)
8
               IN
9
                    key to retrieve ( pmix key t )
               IN
                    info
10
11
                    Array of info structures (array of handles)
12
               IN
                    ninfo
13
                    Number of element in the info array (integer)
14
               OUT val
                     value (handle)
15
16
               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:
17
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.
20
21
               PMIX_IMMEDIATE "pmix.immediate" (bool)
22
                     Specified operation should immediately return an error from the PMIx server if the requested
23
                     data cannot be found - do not request it from the host RM.
24
               PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
                     Scope of the data to be found in a PMIx Get call.
25
26
               PMIX SESSION INFO "pmix.ssn.info" (bool)
27
                     Return information about the specified session. If information about a session other than the
28
                     one containing the requesting process is desired, then the attribute array must contain a
                     PMIX_SESSION_ID attribute identifying the desired target.
29
               PMIX_JOB_INFO "pmix.job.info" (bool)
30
```

**Format** 

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\_NSPACE 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)

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

# 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.

# —— Advice to PMIx library implementers ————

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

## Description

Retrieve information for the specified *key* as published by the process identified in the given <code>pmix\_proc\_t</code>, 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 <code>PMIX\_RANK\_WILDCARD</code>. The list of data referenced in this way is maintained on the PMIx web site at <a href="https://pmix.org/support/faq/wildcard-rank-access/">https://pmix.org/support/faq/wildcard-rank-access/</a> but includes items such as the number of processes in the namespace ( <code>PMIX\_JOB\_SIZE</code> ), total available slots in the allocation ( <code>PMIX\_UNIV\_SIZE</code> ), and the number of nodes in the allocation ( <code>PMIX\_NUM\_NODES</code> ).

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 PMIx v1.0 2 pmix status t 3 PMIx\_Get\_nb(const pmix\_proc\_t \*proc, const char key[], const pmix\_info\_t info[], size\_t ninfo, 4 5 pmix\_value\_cbfunc\_t cbfunc, void \*cbdata) 6 IN proc 7 process reference (handle) IN 8 key 9 key to retrieve (string) IN info 10 11 Array of info structures (array of handles) 12 IN ninfo 13 Number of elements in the *info* array (integer) 14 IN cbfunc Callback function (function reference) 15 IN cbdata 16 17 Data to be passed to the callback function (memory reference) 18 Function returns either: 19 • PMIX\_SUCCESS indicating that the request has been accepted for processing and the provided 20 callback function will be executed upon completion of the operation 21 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this 22 case, the provided callback function will not be executed 23 If executed, the status returned in the provided callback function will be one of the following 24 constants: 25 • PMIX SUCCESS The requested data has been returned 26 • PMIX ERR NOT FOUND The requested data was not available 27 • a non-zero PMIx error constant indicating a reason for the request's failure Required Attributes 28 The following attributes are required to be supported by all PMIx libraries: 29 PMIX\_OPTIONAL "pmix.optional" (bool) 30 Look only in the client's local data store for the requested value - do not request data from 31 the PMIx server if not found. 32 PMIX IMMEDIATE "pmix.immediate" (bool)

**Format** 

Specified operation should immediately return an error from the PMIx server if the requested 1 2 data cannot be found - do not request it from the host RM. 3 PMIX\_DATA\_SCOPE "pmix.scope" (pmix\_scope\_t) Scope of the data to be found in a PMIx Get call. 4 5 PMIX\_SESSION\_INFO "pmix.ssn.info" (bool) 6 Return information about the specified session. If information about a session other than the 7 one containing the requesting process is desired, then the attribute array must contain a 8 **PMIX SESSION ID** attribute identifying the desired target. 9 PMIX\_JOB\_INFO "pmix.job.info" (bool) 10 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 11 12 array must contain a PMIX JOBID or PMIX NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than 13 the one containing the requesting process, then an attribute identifying the target session 14 15 must be provided. 16 PMIX APP INFO "pmix.app.info" (bool) 17 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 18 contain a PMIX APPNUM attribute identifying the desired target. Similarly, if information is 19 20 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. 21 PMIX\_NODE\_INFO "pmix.node.info" (bool) 22 Return information about the specified node. If information about a node other than the one 23 24 containing the requesting process is desired, then the attribute array must contain either the PMIX NODEID or PMIX\_HOSTNAME attribute identifying the desired target. 25 Optional Attributes 26 The following attributes are optional for host environments that support this operation: PMIX\_TIMEOUT "pmix.timeout" (int) 27 Time in seconds before the specified operation should time out (0 indicating infinite) in 28 29 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. 30

<ul> <li>Advice to PMIx library</li> </ul>	y impl	lementers
--	--------	-----------

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 <code>PMIX\_RANK\_WILDCARD</code>. The list of data referenced in this way is maintained on the PMIx web site at <a href="https://pmix.org/support/faq/wildcard-rank-access/">https://pmix.org/support/faq/wildcard-rank-access/</a> but includes items such as the number of processes in the namespace (<a href="mailto:PMIX\_JOB\_SIZE">PMIX\_JOB\_SIZE</a>), total available slots in the allocation (<a href="mailto:PMIX\_UNIV\_SIZE">PMIX\_UNIV\_SIZE</a>), and the number of nodes in the allocation (<a href="mailto:PMIX\_NUM\_NODES">PMIX\_NUM\_NODES</a>).

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.

# 20 5.1.4 PMIx\_Store\_internal

## Summary

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

#### **Format** 1 PMIx v1.0 2 pmix\_status\_t 3 PMIx\_Store\_internal(const pmix\_proc\_t \*proc, const pmix\_key\_t key, 4 5 pmix\_value\_t \*val); 6 IN proc 7 process reference (handle) IN 8 9 key to retrieve (string) IN 10 val 11 Value to store (handle) 12 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 13 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. 5.1.5 Accessing information: examples 17 This section provides examples illustrating methods for accessing information at various levels. The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate 18 19 how PMIx\_Get can be used to obtain information on a session, job, application, 20 process, and node. 5.1.5.1 **Session-level information** 21 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 obtained by the following methods: 24 • for session-level attributes (e.g., PMIX\_UNIV\_SIZE ), specifying the requestor's namespace 25 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:

```
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 */
             PMIx_Init(&myproc, NULL, 0);
7
8
9
             /* get the #slots in our session */
10
             PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
             rc = PMIx Get(&wildcard, PMIX UNIV SIZE, NULL, 0, &value);
11
12
13
             /* get the #nodes in our session */
             PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
14
             rc = PMIx Get(&wildcard, PMIX NUM NODES, &info, 1, &value);
15
16
             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
17
18
             PMIX_SESSION_ID attribute identifying the target session. In the latter case, the proc argument
19
             to PMIx_Get will be ignored:
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);
```

## 1 5.1.5.2 Job-level information

3

4 5

6

7 8

25 26

27 28

29

30

31

33

34

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

- for job-level attributes (e.g., PMIX\_JOB\_SIZE or PMIX\_JOB\_NUM\_APPS), specifying the namespace of the job and a rank of PMIX\_RANK\_WILDCARD for the proc argument to PMIx\_Get; or
- for non-specific attributes (e.g., **PMIX\_NUM\_NODES**), including the **PMIX\_JOB\_INFO** attribute to indicate that the job-level information for that attribute is being requested

Example requests are shown below:

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

# 5.1.5.3 Application-level information

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

- for application-level attributes (e.g., **PMIX\_APP\_SIZE** ), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., PMIX\_APP\_SIZE), including the PMIX\_APPNUM
   attribute specifying the application whose information is being requested. In this case, the
   namespace field of the *proc* argument is used to reference the job containing the application the rank field is ignored;
- or application-level attributes (e.g., PMIX\_APP\_SIZE), including the PMIX\_APPNUM and PMIX\_NSPACE or PMIX\_JOBID attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;

• for non-specific attributes (e.g., **PMIX\_NUM\_NODES**), including the **PMIX\_APP\_INFO** attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

1

3

4 5

6

7

8

9

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);
```

## 5.1.5.4 Process-level information

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

## 5 5.1.5.5 Node-level information

2

3

6 7

8 9

10 11

12

13 14

32

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

- for node-level attributes (e.g., **PMIX\_NODE\_SIZE** ), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., PMIX\_NODE\_SIZE), including the PMIX\_NODEID or
   PMIX\_HOSTNAME attribute specifying the node whose information is being requested. In this case, the *proc* argument's values are ignored; or
- for non-specific attributes (e.g., **PMIX\_NUM\_SLOTS**), including the **PMIX\_NODE\_INFO** attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

```
15
           pmix_info_t info[2];
16
           pmix value t *value;
17
           pmix status t rc;
           pmix_proc_t myproc, otherproc;
18
           uint32 t nodeid;
19
20
21
           /* initialize the client library */
22
           PMIx Init(&myproc, NULL, 0);
23
24
           /* get the #procs on our node */
           rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);
25
26
27
           /* get the #slots on another node */
           PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
28
           PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
29
30
           rc = PMIx_Get(&myproc, PMIX_NUM_SLOTS, info, 2, &value);
31
                   Advice to users
```

An explanation of the use of **PMIx Get** versus **PMIx Query info nb** is provided in 7.1.3.1.

# 5.2 Exchanging Key/Value Pairs

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

# 4 5.2.1 PMIx\_Commit

## 5 Summary

6

7

10

13

14

15

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

## Format

PMIx v1.0

pmix\_status\_t PMIx\_Commit (void)

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

## Description

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

# Advice to users -

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

# 16 5.2.2 PMIx\_Fence

# 17 **Summary**

Execute a blocking barrier across the processes identified in the specified array, collecting information posted via **PMIx\_Put** as directed.

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx\_Fence(const pmix\_proc\_t procs[], size\_t nprocs, const pmix\_info\_t info[], size\_t ninfo) 4 C IN 5 procs 6 Array of pmix proc t structures (array of handles) 7 IN nprocs Number of element in the *procs* array (integer) 8 9 IN Array of info structures (array of handles) 10 IN ninfo 11 Number of element in the *info* array (integer) 12 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. 13 \_\_\_\_\_ Required Attributes 14 The following attributes are required to be supported by all PMIx libraries: 15 PMIX COLLECT DATA "pmix.collect" (bool) Collect data and return it at the end of the operation. 16 **▲**-----**-**Optional Attributes -----\_\_\_\_\_\_ 17 The following attributes are optional for host environments: 18 PMIX TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in 19 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 20 21 the target process from ever exposing its data. 22 PMIX\_COLLECTIVE\_ALGO "pmix.calgo" (char\*) 23 Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the 24 25 acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values. 26 27 PMIX\_COLLECTIVE\_ALGO\_REQD "pmix.calreqd" (bool) If **true**, indicates that the requested choice of algorithm is mandatory. 28

<ul> <li>Advice to PMIx library</li> </ul>	ımpı	iementers
--	------	-----------

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

## Description

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

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

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

# Advice to PMIx library implementers -

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

## Advice to PMIx server hosts —

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

# **5.2.3 PMIx\_Fence\_nb**

## Summary

Execute a nonblocking **PMIx\_Fence** across the processes identified in the specified array of processes, collecting information posted via **PMIx\_Put** as directed.

#### **Format** 1 PMIx v1.0 2 pmix\_status\_t 3 PMIx\_Fence\_nb(const pmix\_proc\_t procs[], size\_t nprocs, const pmix\_info\_t info[], size\_t ninfo, 4 5 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 6 IN procs 7 Array of pmix proc t structures (array of handles) IN 8 9 Number of element in the *procs* array (integer) IN info 10 11 Array of info structures (array of handles) ninfo 12 IN 13 Number of element in the *info* array (integer) 14 IN cbfunc Callback function (function reference) 15 IN cbdata 16 17 Data to be passed to the callback function (memory reference) Returns one of the following: 18 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 20 function prior to returning from the API. 21 22 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called Required Attributes \_\_\_\_\_ 23 The following attributes are required to be supported by all PMIx libraries: PMIX\_COLLECT\_DATA "pmix.collect" (bool) 24 Collect data and return it at the end of the operation. 25

# 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 ( $\theta$  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.

# 5.3 Publish and Lookup Data

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

# 7 5.3.1 PMIx Publish

# 8 Summary

9

Publish data for later access via **PMIx\_Lookup**.

#### **Format** 1 PMIx v1.0 2 pmix status t PMIx\_Publish(const pmix\_info\_t info[], size\_t ninfo) 3 C IN info 4 5 Array of info structures (array of handles) 6 IN ninfo 7 Number of element in the *info* array (integer) 8 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. Required Attributes PMIx libraries are not required to directly support any attributes for this function. However, any 9 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 10 required to add the PMIX USERID and the PMIX GRPID attributes of the client process that 11 published the info. 12 **▲**-----**-**Optional Attributes 13 The following attributes are optional for host environments that support this operation: 14 PMIX\_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in 15 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 16 the target process from ever exposing its data. 17 PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t) 18 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 19 20 PMIX PERSISTENCE "pmix.persist" (pmix persistence t) Value for calls to PMIx Publish. 21 Advice to PMIx library implementers We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host 22 23 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 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 27 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 <code>PMIX\_EXTERNAL\_ERR\_BASE</code> 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.

# o 5.3.2 PMIx Publish nb

# 21 Summary

Nonblocking **PMIx\_Publish** routine.

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx\_Publish\_nb(const pmix\_info\_t info[], size\_t ninfo, pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 4 IN info 5 6 Array of info structures (array of handles) 7 IN ninfo Number of element in the *info* array (integer) 8 9 IN Callback function pmix\_op\_cbfunc\_t (function reference) 10 IN cbdata 11 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 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback 15 function prior to returning from the API. 16 17 • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called 18 19 • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called 20 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 22 23 required to add the PMIX USERID and the PMIX GRPID attributes of the client process that 24 published the info.

		▼ Optional Attributes
1		The following attributes are optional for host environments that support this operation:
2 3 4 5		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.
6 7		PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.
8 9		PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.
		Advice to PMIx library implementers
10 11 12 13 14 15		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.
16		Description
17 18		Nonblocking <b>PMIx_Publish</b> routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.
19 20		Note that the function will return an error if a <b>NULL</b> callback function is given, and that the <i>info</i> array must be maintained until the callback is provided.
21	5.3.3	PMIx_Lookup
22		Summary

Lookup information published by this or another process with  ${\tt PMIx\_Publish}$  or

23

24

PMIx\_Publish\_nb.

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx Lookup(pmix pdata t data[], size t ndata, const pmix\_info\_t info[], size\_t ninfo) 4 \_\_\_\_\_ C INOUT data 5 6 Array of publishable data structures (array of handles) 7 IN ndata Number of elements in the *data* array (integer) 8 9 IN Array of info structures (array of handles) 10 IN ninfo 11 Number of elements in the *info* array (integer) 12 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 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 15 16 required to add the PMIX USERID and the PMIX GRPID attributes of the client process that is requesting the info. 17 **A**-----Optional Attributes -----The following attributes are optional for host environments that support this operation: 18 19 PMIX\_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in 20 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 21 the target process from ever exposing its data. 22 23 PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications. 24 25 PMIX WAIT "pmix.wait" (int) Caller requests that the PMIx server wait until at least the specified number of values are 26 found (0 indicates all and is the default). 27

\_\_\_\_\_\_

## 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 <code>pmix\_info\_t</code> directive.

The *data* parameter consists of an array of <code>pmix\_pdata\_t</code> 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 <code>PMIX\_UNDEF</code>. The function will return <code>PMIX\_SUCCESS</code> 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.

# 27 5.3.4 PMIx\_Lookup\_nb

## Summary

Nonblocking version of **PMIx\_Lookup**.

#### **Format** 1 PMIx v1.0 2 pmix status t PMIx Lookup\_nb(char \*\*keys, 3 const pmix\_info\_t info[], size\_t ninfo, 4 5 pmix\_lookup\_cbfunc\_t cbfunc, void \*cbdata) 6 IN keys 7 Array to be provided to the callback (array of strings) IN 8 9 Array of info structures (array of handles) IN 10 ninfo 11 Number of element in the *info* array (integer) 12 IN cbfunc 13 Callback function (handle) IN cbdata 14 15 Callback data to be provided to the callback function (pointer) Returns one of the following: 16 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 18 function prior to returning from the API. 19 20 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called \_\_\_\_\_\_ Required Attributes PMIx libraries are not required to directly support any attributes for this function. However, any 21 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 22 required to add the PMIX USERID and the PMIX GRPID attributes of the client process that is 23 24 requesting the info. **▲**-----**-**Optional Attributes 25 The following attributes are optional for host environments that support this operation: 26 PMIX\_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in 27 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 28 29 the target process from ever exposing its data. PMIX RANGE "pmix.range" (pmix data range t) 30 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 31 32 PMIX\_WAIT "pmix.wait" (int)

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

# Advice to PMIx library implementers —

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

## Description

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

# **5.3.5 PMIx\_Unpublish**

# 16 Summary

Unpublish data posted by this process using the given keys.

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx\_Unpublish(char \*\*keys, 4 const pmix\_info\_t info[], size\_t ninfo) — C IN 5 info 6 Array of info structures (array of handles) 7 IN ninfo Number of element in the *info* array (integer) 8 9 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. \_\_\_\_\_ Required Attributes 10 PMIx libraries are not required to directly support any attributes for this function. However, any 11 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the PMIX USERID and the PMIX GRPID attributes of the client process that is 12 requesting the operation. 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 ( $\theta$ indicating infinite) in 16 17 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 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 Advice to PMIx library implementers We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host 21 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 created. 26

## 1 Description

2

3

4

5

6

9

10

16

17

18 19

20

21

22

23 24

25

26

27 28

29

30 31

32

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)

C

IN kevs
```

# 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

# Description

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

## **CHAPTER 6**

# **Process Management**

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

# 3 6.1 Abort

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

# 6 6.1.1 PMIx Abort

## 7 Summary

Abort the specified processes

#### 9 **Format**

8

20

21

Error code to return to invoking environment (integer)

IN msg

String message to be returned to user (string)

IN procs

Array of pmix\_proc\_t structures (array of handles)

IN nprocs

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

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

## Description

1 2

3

4

5

6

7

8

9

10

11 12

13

14

15

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.

# 16 6.2 Process Creation

- 17 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.

# 20 6.2.1 PMIx\_Spawn

# 21 **Summary**

Spawn a new job.

#### PMIx v1.0 2 pmix status t 3 PMIx\_Spawn(const pmix\_info\_t job\_info[], size\_t ninfo, 4 const pmix\_app\_t apps[], size\_t napps, 5 char nspace[]) 6 IN job info 7 Array of info structures (array of handles) 8 IN 9 Number of elements in the *job info* array (integer) IN 10 apps 11 Array of **pmix app t** structures (array of handles) 12 IN napps 13 Number of elements in the *apps* array (integer) 14 OUT nspace 15 Namespace of the new job (string) 16 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. Required Attributes PMIx libraries are not required to directly support any attributes for this function. However, any 17 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 18 19 required to add the following attributes to those provided before passing the request to the host: 20 PMIX SPAWNED "pmix.spawned" (bool) true if this process resulted from a call to PMIx\_Spawn. 21 22 PMIX\_PARENT\_ID "pmix.parent" (pmix\_proc\_t) 23 Process identifier of the parent process of the calling process. 24 PMIX REQUESTOR IS CLIENT "pmix.req.client" (bool) The requesting process is a PMIx client. 25 26 PMIX REQUESTOR IS TOOL "pmix.req.tool" (bool) 27 The requesting process is a PMIx tool. Host environments that implement support for PMIx\_Spawn are required to pass the 28 PMIX SPAWNED and PMIX PARENT ID attributes to all PMIx servers launching new child 29 30 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 31 the *info* array of an element of the *apps* array: 32 33 PMIX\_WDIR "pmix.wdir" (char\*) 34 Working directory for spawned processes.

**Format** 

1

```
PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
1
2
                    Set the application's current working directory to the session working directory assigned by
 3
                    the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for
                    the rank to discover the session working directory assigned to the provided namespace
 4
               PMIX_PREFIX "pmix.prefix" (char*)
 5
                    Prefix to use for starting spawned processes.
 6
 7
               PMIX_HOST "pmix.host" (char*)
8
                    Comma-delimited list of hosts to use for spawned processes.
9
               PMIX_HOSTFILE "pmix.hostfile" (char*)
10
                    Hostfile to use for spawned processes.
                                               Optional Attributes
11
               The following attributes are optional for host environments that support this operation:
               PMIX ADD HOSTFILE "pmix.addhostfile" (char*)
12
13
                    Hostfile listing hosts to add to existing allocation.
               PMIX_ADD_HOST "pmix.addhost" (char*)
14
15
                    Comma-delimited list of hosts to add to the allocation.
               PMIX PRELOAD_BIN "pmix.preloadbin" (bool)
16
17
                    Preload binaries onto nodes.
18
               PMIX PRELOAD FILES "pmix.preloadfiles" (char*)
                    Comma-delimited list of files to pre-position on nodes.
19
               PMIX_PERSONALITY "pmix.pers" (char*)
20
21
                    Name of personality to use.
22
               PMIX MAPPER "pmix.mapper" (char*)
                    Mapping mechanism to use for placing spawned processes - when accessed using
23
                    PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping
24
                    mechanism used for the provided namespace.
25
26
               PMIX DISPLAY MAP "pmix.dispmap" (bool)
                    Display process mapping upon spawn.
27
               PMIX_PPR "pmix.ppr" (char*)
28
                    Number of processes to spawn on each identified resource.
29
30
               PMIX_MAPBY "pmix.mapby" (char*)
                    Process mapping policy - when accessed using PMIx_Get, use the
31
                    PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
32
                    provided namespace
33
               PMIX_RANKBY "pmix.rankby" (char*)
34
```

1 2 3	Process ranking policy - when accessed using <b>PMIx_Get</b> , use the <b>PMIX_RANK_WILDCARD</b> value for the rank to discover the ranking algorithm used for the provided namespace
4 5 6 7	<pre>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</pre>
8 9	PMIX_NON_PMI "pmix.nonpmi" (bool)  Spawned processes will not call PMIx_Init.
10 11	PMIX_STDIN_TGT "pmix.stdin" (uint32_t)  Spawned process rank that is to receive stdin.
12 13	<pre>PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) Forward this process's stdin to the designated process.</pre>
14 15	<pre>PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool) Forward stdout from spawned processes to this process.</pre>
16 17	<pre>PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.</pre>
18 19	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.
20 21	PMIX_TAG_OUTPUT "pmix.tagout" (bool)  Tag application output with the identity of the source process.
22 23	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)  Timestamp output from applications.
24 25	<pre>PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  Merge stdout and stderr streams from application processes.</pre>
26 27	PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)  Output application output to the specified file.
28 29	PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.
30 31 32 33	PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)  Number of cpus to assign to each rank - when accessed using PMIx_Get, use the  PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace
34 35	PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)  Do not place processes on the head node.
36	PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)

Do not oversubscribe the cpus. 1 PMIX REPORT BINDINGS "pmix.repbind" (bool) 2 Report bindings of the individual processes. 3 PMIX CPU LIST "pmix.cpulist" (char\*) 4 List of cpus to use for this job - when accessed using PMIx\_Get, use the 5 PMIX RANK WILDCARD value for the rank to discover the cpu list used for the provided 6 7 namespace PMIX JOB RECOVERABLE "pmix.recover" (bool) 8 Application supports recoverable operations. 9 PMIX JOB CONTINUOUS "pmix.continuous" (bool) 10 Application is continuous, all failed processes should be immediately restarted. 11 PMIX\_MAX\_RESTARTS "pmix.maxrestarts" (uint32\_t) 12 13 Maximum number of times to restart a job - when accessed using **PMIx Get**, use the 14 PMIX RANK WILDCARD value for the rank to discover the max restarts for the provided 15 namespace

## **Description**

16 17

18

19 20

21 22

23

24

25

26 27

28

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

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

## — Advice to users

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

# 1 6.2.2 PMIx Spawn nb

```
Summary
 2
 3
               Nonblocking version of the PMIx_Spawn routine.
               Format
 4
   PMIx v1.0
 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,
                                  pmix spawn cbfunc t cbfunc, void *cbdata)
 8
                                                        - C
 9
               IN
                     job_info
                     Array of info structures (array of handles)
10
               IN
                     ninfo
11
12
                     Number of elements in the job_info array (integer)
               IN
13
                     apps
                     Array of pmix_app_t structures (array of handles)
14
               IN
                     cbfunc
15
                     Callback function pmix_spawn_cbfunc_t (function reference)
16
17
               IN
                     cbdata
                     Data to be passed to the callback function (memory reference)
18
               Returns one of the following:
19
20
               • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
                  will be returned in the provided cbfunc. Note that the library must not invoke the callback
21
                  function prior to returning from the API.
22
23
               • a PMIx error constant indicating an error in the request - the cbfunc will not be called
                                                Required Attributes
               PMIx libraries are not required to directly support any attributes for this function. However, any
24
               provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
25
26
               required to add the following attributes to those provided before passing the request to the host:
27
               PMIX_SPAWNED "pmix.spawned" (bool)
                     true if this process resulted from a call to PMIx Spawn.
28
               PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
29
                     Process identifier of the parent process of the calling process.
30
               PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
31
                     The requesting process is a PMIx client.
32
```

```
PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
 1
2
                    The requesting process is a PMIx tool.
               Host environments that implement support for PMIx_Spawn are required to pass the
 3
               PMIX SPAWNED and PMIX PARENT ID attributes to all PMIx servers launching new child
 4
 5
               processes so those values can be returned to clients upon connection to the PMIx server. In
6
               addition, they are required to support the following attributes when present in either the job info or
 7
               the info array of an element of the apps array:
8
               PMIX WDIR "pmix.wdir" (char*)
                    Working directory for spawned processes.
9
10
               PMIX SET SESSION CWD "pmix.ssncwd" (bool)
                    Set the application's current working directory to the session working directory assigned by
11
12
                    the RM - when accessed using PMIx Get, use the PMIX RANK WILDCARD value for
                    the rank to discover the session working directory assigned to the provided namespace
13
               PMIX PREFIX "pmix.prefix" (char*)
14
                    Prefix to use for starting spawned processes.
15
               PMIX HOST "pmix.host" (char*)
16
                    Comma-delimited list of hosts to use for spawned processes.
17
18
               PMIX HOSTFILE "pmix.hostfile" (char*)
19
                    Hostfile to use for spawned processes.

    ▼------ Optional Attributes ------

20
               The following attributes are optional for host environments that support this operation:
21
               PMIX ADD HOSTFILE "pmix.addhostfile" (char*)
22
                    Hostfile listing hosts to add to existing allocation.
23
               PMIX ADD HOST "pmix.addhost" (char*)
                    Comma-delimited list of hosts to add to the allocation.
24
25
               PMIX PRELOAD BIN "pmix.preloadbin" (bool)
26
                    Preload binaries onto nodes.
27
               PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
                    Comma-delimited list of files to pre-position on nodes.
28
               PMIX PERSONALITY "pmix.pers" (char*)
29
30
                    Name of personality to use.
31
               PMIX_MAPPER "pmix.mapper" (char*)
32
                    Mapping mechanism to use for placing spawned processes - when accessed using
                    PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping
33
34
                    mechanism used for the provided namespace.
```

1 2	PMIX_DISPLAY_MAP "pmix.dispmap" (bool)  Display process mapping upon spawn.
3 4	PMIX_PPR "pmix.ppr" (char*)  Number of processes to spawn on each identified resource.
5 6 7 8	<pre>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</pre>
9  0  1  2	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
13 14 15 16	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
17 18	PMIX_NON_PMI "pmix.nonpmi" (bool)  Spawned processes will not call PMIx_Init.
19 20	PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive stdin.
21 22	PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)  Forward this process's stdin to the designated process.
23 24	PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool) Forward stdout from spawned processes to this process.
25 26	PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.
27 28	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.
29 30	PMIX_TAG_OUTPUT "pmix.tagout" (bool)  Tag application output with the identity of the source process.
31 32	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.
33 34	PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  Merge stdout and stderr streams from application processes.
35 36	PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Output application output to the specified file

1 2	PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.
3 4 5 6	PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)  Number of cpus to assign to each rank - when accessed using PMIx_Get , use the  PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace
7 8	PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)  Do not place processes on the head node.
9 10	PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)  Do not oversubscribe the cpus.
l1 l2	PMIX_REPORT_BINDINGS "pmix.repbind" (bool) Report bindings of the individual processes.
13 14 15 16	PMIX_CPU_LIST "pmix.cpulist" (char*)  List of cpus to use for this job - when accessed using PMIx_Get , use the  PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided namespace
17 18	PMIX_JOB_RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations.
19 20	PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) Application is continuous, all failed processes should be immediately restarted.
21 22 23 24	PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)  Maximum number of times to restart a job - when accessed using PMIx_Get, use the  PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided namespace
25	Description
-5 26 27	Nonblocking version of the <b>PMIx_Spawn</b> routine. The provided callback function will be executed upon successful start of <i>all</i> specified application processes.
	Advice to users
28 29 30	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 <i>all</i> 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 <code>PMIx\_Query\_info\_nb</code>). The host environment is required to generate a <code>PMIX\_ERR\_INVALID\_TERMINATION</code> event should any process in the assemblage terminate or call <code>PMIx\_Finalize</code> without first disconnecting from the assemblage.

### Advice to users -

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

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

# 27 6.3.1 PMIx Connect

# Summary

Connect namespaces.

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx\_Connect(const pmix\_proc\_t procs[], size\_t nprocs, const pmix\_info\_t info[], size\_t ninfo) 4 C IN 5 procs 6 Array of proc structures (array of handles) 7 IN nprocs Number of elements in the *procs* array (integer) 8 IN 9 Array of info structures (array of handles) 10 IN ninfo 11 Number of elements in the *info* array (integer) 12 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. **A**-----**A** Optional Attributes -----The following attributes are optional for host environments that support this operation: 16 17 PMIX TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in 18 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 19 the target process from ever exposing its data. 20 21 PMIX\_COLLECTIVE\_ALGO "pmix.calgo" (char\*) 22 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 23 acceptable values for this attribute will be environment-dependent - users are encouraged to 24 25 check their host environment for supported values. 26 PMIX\_COLLECTIVE\_ALGO\_REQD "pmix.calreqd" (bool) If **true**, indicates that the requested choice of algorithm is mandatory. 27

# 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.

1 2

3

4

5

6

7 8

9

10

11

12

13

14 15

16

17

18

19

20

21

# 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

# 11 Summary

4

5

6 7

8

9

12

13

18

19

20

21

22

23

24

25

26 27

28

29 30 PMIx v1.0

Nonblocking **PMIx\_Connect\_nb** routine.

### Format

IN

IN

procs

```
pmix_status_t

PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,

const pmix_info_t info[], size_t ninfo,

pmix_op_cbfunc_t cbfunc, void *cbdata)

C
```

Array of proc structures (array of handles)

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

IN info
 Array of info structures (array of handles)

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

IN cbfunc

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

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

Returns one of the following:

1 2 3	• <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.
4 5	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
6 7	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
	▼
8 9	PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.
	▼ Optional Attributes
0	The following attributes are optional for host environments that support this operation:
1 2 3 4	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.
5 6 7 8 9	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.
20 21	PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.
	Advice to PMIx library implementers
22 23 24 25 26	We recommend that implementation of the <b>PMIX_TIMEOUT</b> 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 <b>PMIX_TIMEOUT</b> directly in the PMIx server library must take care to resolve the race condition and should avoid passing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not created.

1

3

4

5

6

8

9

10

22

23

24

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.

# 7 6.3.3 PMIx Disconnect

### Summary

Disconnect a previously connected set of processes.

### Format

```
PMIx v1.0
11
              pmix status t
12
              PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
                                   const pmix_info_t info[], size_t ninfo);
13
14
              IN
                   procs
                    Array of proc structures (array of handles)
15
              IN
16
                   nprocs
                    Number of elements in the procs array (integer)
17
              IN
18
19
                    Array of info structures (array of handles)
              IN
                   ninfo
20
                    Number of element in the info array (integer)
21
```

### Required Attributes

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

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

# Optional Attributes The following attributes are optional for host environments that support this operation: PMIX TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. Advice to PMIx library implementers We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not created. **Description** 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.

1 2

3

4

5

6 7

8

9

10

11

12

13

14

15

16 17

18

19 20

21

22

23

### Advice to PMIx server hosts -

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

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

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

#### 6.3.4 PMIx Disconnect nb

### Summary

1

2

3 4

5

6

7

8 9

10

12 13

14

19

29

30

PMIx v1.0

Nonblocking **PMIx Disconnect** routine.

### **Format**

IN

IN

cbdata

```
pmix_status_t
15
            PMIx Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,
16
17
                               const pmix info t info[], size t ninfo,
                               pmix op cbfunc t cbfunc, void *cbdata);
18
```

```
procs
                      Array of proc structures (array of handles)
20
21
                IN
                     nprocs
                      Number of elements in the procs array (integer)
22
23
                IN
                      info
                      Array of info structures (array of handles)
24
                IN
25
                     ninfo
26
                      Number of element in the info array (integer)
27
                IN
                      cbfunc
28
                      Callback function pmix_op_cbfunc_t (function reference)
```

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

	e
W	MIX_SUCCESS, indicating that the request is being processed by the host environment - result ill be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback notion prior to returning from the API.
	${f MIX\_OPERATION\_SUCCEEDED}$ , indicating that the request was immediately processed and turned $success$ - the $cbfunc$ will $not$ be called
	PMIx error constant indicating either an error in the input or that the request was immediately ocessed and failed - the <i>cbfunc</i> will <i>not</i> be called
<b>~</b> -	Required Attributes
prov	x libraries are not required to directly support any attributes for this function. However, any ided 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:
PMI	X_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.
<b>-</b>	Advice to PMIx library implementers —
envii inter direc	recommend that implementation of the <b>PMIX_TIMEOUT</b> attribute be left to the host ronment due to race condition considerations between completion of the operation versus nal timeout in the PMIx server library. Implementers that choose to support <b>PMIX_TIMEOUT</b> ctly in the PMIx server library must take care to resolve the race condition and should avoid ing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not seed.

Returns one of the following:

Nonblocking **PMIx\_Disconnect** routine. The callback function is called once all processes identified in *procs* have called either **PMIx\_Disconnect\_nb** or its blocking version, *and* the host environment has completed any required supporting operations. See the advice provided in the description for **PMIx\_Disconnect** for more information.

#### **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

PMIx Process monitor nb APIs.

# 5 7.1 Query

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

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

PMIx supports this use-case by defining an attribute key ( PMIX\_TIME\_REMAINING ) that can be used with the PMIx\_Query\_info\_nb interface to obtain the number of seconds remaining in the current job allocation. Note that one could alternatively use the PMIx\_Register\_event\_handler API to register for an event indicating incipient job termination, and then use the PMIx\_Job\_control\_nb API to request that the host SMS generate an event a specified amount of time prior to reaching the maximum run time. PMIx provides such alternate methods as a means of maximizing the probability of a host system supporting at least one method by which the application can obtain the desired service.

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

# 1 7.1.1 PMIx\_Resolve\_peers

### 2 Summary

3

4

17

18

19 20

21

22

23

26

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

#### Format

```
PMIx v1.0
5
             pmix status t
6
              PMIx_Resolve_peers(const char *nodename,
 7
                                     const pmix nspace t nspace,
8
                                     pmix proc t **procs, size t *nprocs)
9
              IN
                  nodename
                   Name of the node to query (string)
10
              IN
11
                  nspace
12
                   namespace (string)
              OUT procs
13
                   Array of process structures (array of handles)
14
```

15 **OUT** nprocs
16 Number of eler

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

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

# Description

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

# 4 7.1.2 PMIx Resolve nodes

# 25 Summary

Return a list of nodes hosting processes within the given namespace.

```
Format
1
   PMIx v1.0
2
              pmix_status_t
 3
               PMIx_Resolve_nodes(const char *nspace, char **nodelist)
                                                        C
4
               IN
                    nspace
 5
                    Namespace (string)
6
               OUT nodelist
 7
                    Comma-delimited list of nodenames (string)
8
               Returns PMIX SUCCESS or a negative value corresponding to a PMIx error constant.
               Description
9
10
               Given a nspace, return the list of nodes hosting processes within that namespace. The returned
               string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the
11
12
               string when done with it.
   7.1.3
             PMIx_Query_info_nb
14
               Summary
               Query information about the system in general.
15
16
               Format
   PMIx v2.0
17
               pmix status t
               PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,
18
                                       pmix_info_cbfunc_t cbfunc, void *cbdata)
19
20
               IN
                    queries
                    Array of query structures (array of handles)
21
22
               IN
                    nqueries
23
                    Number of elements in the queries array (integer)
               IN
                    cbfunc
24
25
                    Callback function pmix_info_cbfunc_t (function reference)
               IN
26
                    cbdata
                    Data to be passed to the callback function (memory reference)
27
28
               Function returns either:
```

• PMIX\_SUCCESS indicating that the request has been accepted for processing and the provided 1 2 callback function will be executed upon completion of the operation 3 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this 4 case, the provided callback function will not be executed 5 If executed, the status returned in the provided callback function will be one of the following 6 constants: 7 • PMIX SUCCESS All data has been returned • PMIX\_ERR\_NOT\_FOUND None of the requested data was available 8 9 • PMIX\_ERR\_PARTIAL\_SUCCESS Some of the data has been returned 10 • 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 11 \_\_\_\_\_ Required Attributes 12 PMIx libraries that support this API are required to support the following attributes: 13 PMIX\_QUERY\_REFRESH\_CACHE "pmix.qry.rfsh" (bool) 14 Retrieve updated information from server. 15 PMIX SESSION INFO "pmix.ssn.info" (bool) 16 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 17 **PMIX SESSION ID** attribute identifying the desired target. 18 19 PMIX JOB INFO "pmix.job.info" (bool) 20 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 21 array must contain a PMIX\_JOBID or PMIX\_NSPACE attribute identifying the desired 22 target. Similarly, if information is requested about a job or namespace in a session other than 23 24 the one containing the requesting process, then an attribute identifying the target session 25 must be provided. 26 PMIX\_APP\_INFO "pmix.app.info" (bool) Return information about the specified application. If information about an application other 27 28 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 29 requested about an application in a job or session other than the one containing the requesting 30 process, then attributes identifying the target job and/or session must be provided. 31 32 PMIX NODE INFO "pmix.node.info" (bool) Return information about the specified node. If information about a node other than the one 33

containing the requesting process is desired, then the attribute array must contain either the

**PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute identifying the desired target.

34

2 3 4	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.
	▼ Optional Attributes
5	The following attributes are optional for host environments that support this operation:
6 7	PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces.
8 9	<pre>PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job.</pre>
10 11	<pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues.</pre>
12 13	PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue.
14 15 16	<pre>PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns (     pmix_data_array_t) an array of pmix_proc_info_t.</pre>
17 18 19 20	<pre>PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)     Input namespace of the job whose information is being requested returns (     pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same node.</pre>
21 22	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)  Return a comma-delimited list of supported spawn attributes.
23 24	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)  Return a comma-delimited list of supported debug attributes.
25 26	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)  Return information on memory usage for the processes indicated in the qualifiers.
27 28	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool) Report average values.
29 30	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values.
31 32	<pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested.</pre>
33	<pre>PMIX_TIME_REMAINING "pmix.time.remaining" (char*)</pre>

1 2	Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
3	Description
4 5 6 7	Query information about the system in general. This can include a list of active namespaces, network topology, etc. Also can be used to query node-specific info such as the list of peers executing on a given node. We assume that the host RM will exercise appropriate access control on the information.
8 9	NOTE: There is no blocking form of this API as the structures passed to query info differ from those for receiving the results.
10 11 12 13	The <i>status</i> argument to the callback function indicates if requested data was found or not. An array of <code>pmix_info_t</code> will contain each key that was provided and the corresponding value that was found. Requests for keys that are not found will return the key paired with a value of type <code>PMIX_UNDEF</code> .
	Advice to users
14 15 16	The desire to query a list of attributes supported by the implementation and/or the host environment has been expressed and noted. The PMIx community is exploring the possibility and it will likely become available in a future release
	Advice to PMIx library implementers
17 18 19 20 21	Information returned from PMIx_Query_info_nb shall be locally cached so that retrieval by subsequent calls to PMIx_Get or PMIx_Query_info_nb can succeed with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or the host environment. Queries that include the PMIX_QUERY_REFRESH_CACHE attribute shall bypass the local cache and retrieve a new value for the query, refreshing the values in the cache upon return.

# 7.1.3.1 Using PMIx\_Get VS PMIx\_Query\_info\_nb

- Both **PMIx\_Get** and **PMIx\_Query\_info\_nb** can be used to retrieve information about the system. In general, the *get* operation should be used to retrieve:
- information provided by the host environment at time of job start. This includes information on the number of processes in the job, their location, and possibly their communication endpoints
- information posted by processes via the PMIx\_Put function

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

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

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

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

# 7.2 Allocation Requests

- This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:
- Evolving applications that dynamically request and return resources as they execute

- Malleable environments where the scheduler redirects resources away from executing
   applications for higher priority jobs or load balancing
  - Resilient applications that need to request replacement resources in the face of failures
  - *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing

PMIx attempts to address this range of use-cases with a single, flexible API.

# 7.2.1 PMIx Allocation request nb

### Summary

3

4 5

6

9

10

26

27 28

29

30 31

32

Request an allocation operation from the host resource manager.

#### Format

```
PMIx v2.0
11
              pmix_status_t
12
              PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
13
                                                pmix_info_t info[], size_t ninfo,
                                                pmix info cbfunc t cbfunc, void *cbdata);
14
              IN
15
                   directive
16
                    Allocation directive (handle)
              IN
17
                   info
                    Array of pmix_info_t structures (array of handles)
18
19
              IN
                   ninfo
                    Number of elements in the info array (integer)
20
              IN
21
                   cbfunc
                    Callback function pmix_info_cbfunc_t (function reference)
22
                   cbdata
23
              IN
24
                    Data to be passed to the callback function (memory reference)
25
              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

	▼
1 2 3 4	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.
5 6	Host environments that implement support for this operation are required to support the following attributes:
7 8 9	<pre>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.</pre>
10 11	<pre>PMIX_ALLOC_NUM_NODES</pre>
12 13	<pre>PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t) Number of cpus.</pre>
14 15	PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t) Time in seconds.
	▼Optional Attributes
16	The following attributes are optional for host environments that support this operation:
17 18	<pre>PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*) Regular expression of the specific nodes.</pre>
19 20	<pre>PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*) Regular expression of the number of cpus for each node.</pre>
21 22	<pre>PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*) Regular expression of the specific cpus indicating the cpus involved.</pre>
23 24	<pre>PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float) Number of Megabytes.</pre>
25 26 27 28	<pre>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.</pre>
29 30	<pre>PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*) Name of the network.</pre>
31 32	<pre>PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)</pre>

PMIX_ALLOC_NETWORK_QOS	"pmix.alloc.netqos"	$(\mathtt{char} \star)$
Quality of service level		

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

- Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and
  priorities. This includes extending the time limit on current resources, and/or requesting
  additional resources be allocated to the requesting job. Any additional allocated resources will be
  considered as part of the current allocation, and thus will be released at the same time.
- Return no-longer-required resources to the scheduler. This includes the "loan" of resources back to the scheduler with a promise to return them upon subsequent request.

# 7.2.2 PMIx\_Job\_control\_nb

The PMIx\_Job\_control\_nb API enables the application and SMS to coordinate the response to failures and other events. This can include requesting termination of the entire job or a subset of processes within a job, but can also be used in combination with other PMIx capabilities (e.g., allocation support and event notification) for more nuanced responses. For example, an application notified of an incipient over-temperature condition on a node could use the PMIx\_Allocation\_request\_nb interface to request replacement nodes while simultaneously using the PMIx\_Job\_control\_nb interface to direct that a checkpoint event be delivered to all processes in the application. If replacement resources are not available, the application might use the PMIx\_Job\_control\_nb interface to request that the job continue at a lower power setting, perhaps sufficient to avoid the over-temperature failure.

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

#### Summary 1 2 Request a job control action. 3 Format PMIx v2.04 pmix status t 5 PMIx Job control nb(const pmix proc t targets[], size t ntargets, const pmix\_info\_t directives[], size\_t ndirs, 6 7 pmix\_info\_cbfunc\_t cbfunc, void \*cbdata) IN 8 targets 9 Array of proc structures (array of handles) 10 IN ntargets 11 Number of element in the *targets* array (integer) IN directives 12 13 Array of info structures (array of handles) 14 IN ndirs 15 Number of element in the *directives* array (integer) IN cbfunc 16 Callback function **pmix** info **cbfunc** t (function reference) 17 IN 18 cbdata Data to be passed to the callback function (memory reference) 19 20 Returns one of the following: 21 • PMIX SUCCESS, indicating that the request is being processed by the host environment - result 22 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API. 23 24 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called 25 26 • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called 27 Required Attributes PMIx libraries are not required to directly support any attributes for this function. However, any 28 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 29 required to add the PMIX USERID and the PMIX GRPID attributes of the client process making 30 31 the request. Host environments that implement support for this operation are required to support the following 32 33 attributes: 34 PMIX JOB CTRL ID "pmix.jctrl.id" (char\*)

```
Provide a string identifier for this request.
1
              PMIX JOB CTRL PAUSE "pmix.jctrl.pause" (bool)
 2
                   Pause the specified processes.
 3
              PMIX JOB CTRL RESUME "pmix.jctrl.resume" (bool)
 4
5
                   Resume ("un-pause") the specified processes.
6
              PMIX JOB CTRL KILL "pmix.jctrl.kill" (bool)
                   Forcibly terminate the specified processes and cleanup.
 7
              PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
8
9
                   Send given signal to specified processes.
              PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
10
                   Politely terminate the specified processes.
11

→ - - - Optional Attributes

12
              The following attributes are optional for host environments that support this operation:
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
13
14
                   Cancel the specified request (NULL implies cancel all requests from this requestor).
              PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
15
                   Restart the specified processes using the given checkpoint ID.
16
17
              PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
18
                   Checkpoint the specified processes and assign the given ID to it.
19
              PMIX JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
                   Use event notification to trigger a process checkpoint.
20
21
              PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
22
                   Use the given signal to trigger a process checkpoint.
23
              PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
24
                   Time in seconds to wait for a checkpoint to complete.
25
              PMIX JOB CTRL CHECKPOINT METHOD
              "pmix.jctrl.ckmethod" (pmix_data_array_t)
26
                   Array of pmix_info_t declaring each method and value supported by this application.
27
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
28
29
                   Regular expression identifying nodes that are to be provisioned.
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
30
31
                   Name of the image that is to be provisioned.
              PMIX JOB CTRL PREEMPTIBLE "pmix.jctrl.preempt" (bool)
32
```

### Description

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

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

# 7.3 Process and Job Monitoring

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

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

# 4 7.3.1 PMIx Process monitor nb

# Summary

Request that application processes be monitored.

#### **Format** 1 PMIx v2.0 2 pmix\_status\_t 3 PMIx\_Process\_monitor\_nb(const pmix\_info\_t \*monitor, pmix\_status\_t error, 4 const pmix\_info\_t directives[], size\_t ndirs, 5 pmix\_info\_cbfunc\_t cbfunc, void \*cbdata) 6 IN monitor 7 info (handle) IN 8 error 9 status (integer) IN directives 10 11 Array of info structures (array of handles) 12 IN ndirs 13 Number of elements in the *directives* array (integer) 14 IN cbfunc 15 Callback function **pmix\_info\_cbfunc\_t** (function reference) IN cbdata 16 17 Data to be passed to the callback function (memory reference) Returns one of the following: 18 19 • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result 20 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API. 21 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 Optional Attributes 26 The following attributes may be implemented by a PMIx library or by the host environment. If 27 supported by the PMIx server library, then the library must not pass the supported attributes to the 28 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 29 PMIX USERID and the PMIX GRPID attributes of the requesting process: 30 PMIX MONITOR ID "pmix.monitor.id" (char\*) 31 Provide a string identifier for this request. 32 33 PMIX MONITOR CANCEL "pmix.monitor.cancel" (char\*) 34 Identifier to be canceled (**NULL** means cancel all monitoring for this process).

1 2	<pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>
3 4	<pre>PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void) Register to have the PMIx server monitor the requestor for heartbeats.</pre>
5 6	<pre>PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.</pre>
7 8	<pre>PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t) Number of heartbeats that can be missed before generating the event.</pre>
9 10	<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>
11 12	<pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)     Monitor size of given file is growing to determine if the application is running.</pre>
13 14	<pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre>
15 16	<pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)  Monitor time since last modified of given file to determine if the application is running.</pre>
17 18	<pre>PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t) Time in seconds between checking the file.</pre>
19 20	<pre>PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)     Number of file checks that can be missed before generating the event.</pre>

Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not become "wedged". When a monitor detects the specified alarm condition, it will generate an event notification using the provided error code and passing along any available relevant information. It is up to the caller to register a corresponding event handler.

The *monitor* argument is an attribute indicating the type of monitor being requested. For example, **PMIX\_MONITOR\_FILE** to indicate that the requestor is asking that a file be monitored.

The *error* argument is the status code to be used when generating an event notification alerting that the monitor has been triggered. The range of the notification defaults to **PMIX\_RANGE\_NAMESPACE**. This can be changed by providing a **PMIX\_RANGE** directive.

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

1 The *cbfunc* function provides a *status* to indicate whether or not the request was granted, and to 2 provide some information as to the reason for any denial in the pmix info cbfunc t array of 3 pmix info t structures. 7.3.2 PMIx\_Heartbeat Summary 5 Send a heartbeat to the PMIx server library 6 **Format** 7 PMIx v2.0 8 void PMIx Heartbeat(void) **Description** 9 A simplified macro wrapping PMIx\_Process\_monitor\_nb that sends a heartbeat to the 10 11 PMIx server library. Logging 12 **7.4** 13 The logging interface supports posting information by applications and SMS elements to persistent storage. This function is not intended for output of computational results, but rather for reporting 14 15 status and saving state information such as inserting computation progress reports into the 16 application's SMS job log or error reports to the local syslog. 17 **7.4.1** PMIx Log nb

# 18 **Summary**

Log data to a data service.

#### PMIx v2.0 2 pmix status t 3 PMIx\_Log\_nb(const pmix\_info\_t data[], size\_t ndata, const pmix\_info\_t directives[], size\_t ndirs, 4 5 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 6 IN data 7 Array of info structures (array of handles) IN 8 9 Number of elements in the *data* array (size t) IN directives 10 11 Array of info structures (array of handles) 12 IN ndirs 13 Number of elements in the *directives* array (size\_t) 14 IN cbfunc 15 Callback function **pmix\_op\_cbfunc\_t** (function reference) IN cbdata 16 17 Data to be passed to the callback function (memory reference) Return codes are one of the following: 18 19 **PMIX\_SUCCESS** The logging request is valid and is being processed. The resulting status from the operation will be provided in the callback function. Note that the library must not invoke 20 the callback function prior to returning from the API. 21 PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and 22 returned success - the cbfunc will not be called 23 PMIX ERR BAD PARAM The logging request contains at least one incorrect entry that prevents 24 25 it from being processed. The callback function will *not* be called. PMIX ERR NOT SUPPORTED The PMIx implementation does not support this function. The 26 callback function will not be called. 27 Required Attributes If the PMIx library does not itself perform this operation, then it is required to pass any attributes 28 provided by the client to the host environment for processing. In addition, it must include the 29 following attributes in the passed *info* array: 30 PMIX\_USERID "pmix.euid" (uint32\_t) 31 Effective user id. 32 PMIX\_GRPID "pmix.egid" (uint32\_t) 33 34 Effective group id.

**Format** 

```
Host environments that implement support for this operation are required to support the following
 1
 2
               attributes:
 3
               PMIX LOG STDERR "pmix.log.stderr" (char*)
                     Log string to stderr.
 4
               PMIX LOG STDOUT "pmix.log.stdout" (char*)
 5
 6
                     Log string to stdout.
 7
               PMIX LOG SYSLOG "pmix.log.syslog" (char*)
                     Log data to syslog. Defaults to ERROR priority.
 8
                                               Optional Attributes ------
                 -----
9
               The following attributes are optional for host environments that support this operation:
               PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
10
                     Message blob to be sent somewhere.
11
               PMIX LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
12
                     Log via email based on pmix_info_t containing directives.
13
14
               PMIX LOG EMAIL ADDR "pmix.log.emaddr" (char*)
                     Comma-delimited list of email addresses that are to receive the message.
15
               PMIX LOG EMAIL_SUBJECT "pmix.log.emsub" (char*)
16
                     Subject line for email.
17
18
               PMIX LOG EMAIL MSG "pmix.log.emmsg" (char*)
                     Message to be included in email.
19
               Description
20
21
               Log data subject to the services offered by the host environment. The data to be logged is provided
22
               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
23
24
               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
25
26
               data as it is not a "performant" transport and can perturb the application since it involves the local
27
               PMIx server and host SMS daemon.
```

### **CHAPTER 8**

# **Event Notification**

This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

# 8.1 Notification and Management

PMIx event notification provides an asynchronous out-of-band mechanism for communicating events between application processes and/or elements of the SMS. Its uses span a wide range that includes fault notification, coordination between multiple programming libraries within a single process, and workflow orchestration for non-synchronous programming models. Events can be divided into two distinct classes:

- *Job-specific events* directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.
- Environment events indirectly relate to a job but do not specifically target the job itself. This category includes SMS-generated events such as Error Check and Correction (ECC) errors, temperature excursions, and other non-job conditions that might directly affect a session's resources, but would never include an event generated by an application process. Note that although these do potentially impact the session's jobs, they are not directly tied to those jobs. Thus, events in this category are to be delivered to the PMIx server library only upon request.

Both SMS elements and applications can register for events of either type.

# Advice to PMIx library implementers -

Race conditions can cause the registration to come after events of possible interest (e.g., a memory ECC event that occurs after start of execution but prior to registration, or an application process generating an event prior to another process registering to receive it). SMS vendors are *requested* to cache environment events for some time to mitigate this situation, but are not *required* to do so. However, PMIx implementers are *required* to cache all events received by the PMIx server library and to deliver them to registering clients in the same order in which they were received

6

10

11

23

31

Applications must be aware that they may not receive environment events that occur prior to registration, depending upon the capabilities of the host SMS.

The generator of an event can specify the target range for delivery of that event. Thus, the generator can choose to limit notification to processes on the local node, processes within the same job as the generator, processes within the same allocation, other threads within the same process, only the SMS (i.e., not to any application processes), all application processes, or to a custom range based on specific process identifiers. Only processes within the given range that register for the provided event code will be notified. In addition, the generator can use attributes to direct that the event not be delivered to any default event handlers, or to any multi-code handler (as defined below).

Event notifications provide the process identifier of the source of the event plus the event code and any additional information provided by the generator. When an event notification is received by a process, the registered handlers are scanned for their event code(s), with matching handlers assembled into an event chain for servicing. Note that users can also specify a source range when registering an event (using the same range designators described above) to further limit when they are to be invoked. When assembled, PMIx event chains are ordered based on both the specificity of the event handler and user directives at time of handler registration. By default, handlers are grouped into three categories based on the number of event codes that can trigger the callback:

- single-code handlers are serviced first as they are the most specific. These are handlers that are registered against one specific event code.
- multi-code handlers are serviced once all single-code handlers have completed. The handler will be included in the chain upon receipt of an event matching any of the provided codes.
- default handlers are serviced once all multi-code handlers have completed. These handlers are always included in the chain unless the generator specifically excludes them.

Users can specify the callback order of a handler within its category at the time of registration. Ordering can be specified either by providing the relevant returned event handler registration ID or using event handler names, if the user specified an event handler name when registering the corresponding event. Thus, users can specify that a given handler be executed before or after another handler should both handlers appear in an event chain (the ordering is ignored if the other handler isn't included). Note that ordering does not imply immediate relationships. For example, multiple handlers registered to be serviced after event handler A will all be executed after A, but are not guaranteed to be executed in any particular order amongst themselves.

In addition, one event handler can be declared as the *first* handler to be executed in the chain. This handler will always be called prior to any other handler, regardless of category, provided the incoming event matches both the specified range and event code. Only one handler can be so designated — attempts to designate additional handlers as first will return an error. Deregistration of the declared *first* handler will re-open the position for subsequent assignment.

Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This handler will *always* be called after all other handlers have executed, regardless of category, provided the incoming event matches both the specified range and event code. Note that this handler will not be called if the chain is terminated by an earlier handler. Only one handler can be designated as *last* — attempts to designate additional handlers as *last* will return an error. Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

#### Advice to users

Note that the *last* handler is called *after* all registered default handlers that match the specified range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

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. PMIx automatically aggregates the status and any results of each handler (as provided in the completion callback) with status from all prior handlers so that each step in the chain has full knowledge of what preceded it. An event handler can terminate all further progress along the chain by passing the **PMIX EVENT ACTION COMPLETE** status to the completion callback function.

# 7 8.1.1 PMIx\_Register\_event\_handler

# 18 Summary

Register an event handler

#### 20 Format

PMIx v2.0

 void

1	IN codes
2	Array of status codes (array of <b>pmix_status_t</b> )
3	IN ncodes
4	Number of elements in the <i>codes</i> array (size_t)
5	IN info
6	Array of info structures (array of handles)
7	IN ninfo
8	Number of elements in the <i>info</i> array (size_t)
9	IN evhdlr
10	Event handler to be called <b>pmix_notification_fn_t</b> (function reference)
11	IN cbfunc
12	Callback function <b>pmix_evhdlr_reg_cbfunc_t</b> (function reference)
13	IN cbdata
14	Data to be passed to the cbfunc callback function (memory reference)
	→ Required Attributes
15	The following attributes are required to be supported by all PMIx libraries:
16	<pre>PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)</pre>
17	String name identifying this handler.
18	PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool)
19	Invoke this event handler before any other handlers.
20	<pre>PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool)</pre>
21	Invoke this event handler after all other handlers have been called.
22	<pre>PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool)</pre>
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	<pre>PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)</pre>
27	Put this event handler immediately before the one specified in the (char*) value.
00	DMIV EVENUE UDID AEMED Unniversitate (change)
28 29	<pre>PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*) Put this event handler immediately after the one specified in the (char*) value.</pre>
19	r ut this event nandier infinediately after the one specified in the (Char*) value.
30	<pre>PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)</pre>
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.evrange" (pmix data array t*)

1	Array of <b>pmix_proc_t</b> defining range of event notification.
2 3	<pre>PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.</pre>
4 5 6	<pre>PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)    Object to be returned whenever the registered callback function cbfunc is invoked. The    object will only be returned to the process that registered it.</pre>
7 8	Host environments that implement support for PMIx event notification are required to support the following attributes:
9 10	<pre>PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t) The single process that was affected.</pre>
11 12	PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)  Array of pmix_proc_t defining affected processes.
	▼Optional Attributes
13 14 15	Host environments that support PMIx event notification <i>may</i> offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes are optional for host environments that support this operation:
16 17	<pre>PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool) The RM intends to terminate this session.</pre>
18 19	<pre>PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool) The RM intends to terminate this job.</pre>
20 21	PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool) The RM intends to terminate all processes on this node.
22 23	<pre>PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool) The RM intends to terminate just this process.</pre>
24 25	PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int) The time in seconds before the RM will execute error response.
26 27	PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)  Do not generate an event when this job normally terminates.

 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.

# 25 8.1.2 PMIx Deregister event handler

### Summary

Deregister an event handler.

```
Format
1
   PMIx v2.0
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
7
                    Event handler ID returned by registration (size_t)
              IN
8
9
                    Callback function to be executed upon completion of operation pmix_op_cbfunc_t
10
                    (function reference)
11
              IN
                   cbdata
                    Data to be passed to the cbfunc callback function (memory reference)
12
              Description
13
              Deregister an event handler. If non-NULL, the provided cbfunc will be called to confirm removal
14
15
              of the designated handler, including a status code as per the following:
               PMIX_SUCCESS The event handler was successfully deregistered.
16
               PMIX_ERR_BAD_PARAM The provided evhdlr_ref was unrecognized.
17
               PMIX ERR NOT SUPPORTED The PMIx implementation does not support event notification.
18
19
              The callback function must not be executed prior to returning from the API.
   8.1.3
             PMIx Notify event
              Summary
21
              Report an event for notification via any registered event handler.
22
              Format
23
   PMIx v2.0
24
              pmix_status_t
              PMIx_Notify_event(pmix_status_t status,
25
                                     const pmix_proc_t *source,
26
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 <b>pmix_proc_t</b> 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 <i>info</i> array (size_t)
12	IN cbfunc
13	Callback function to be executed upon completion of operation <pre>pmix_op_cbfunc_t</pre>
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 <i>not</i> reflect the success or failure of delivering the
20	event to any recipients. The callback function <i>must not</i> 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 <i>not</i> 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	<pre>PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)</pre>
34	Array of <b>pmix_proc_t</b> defining range of event notification.
35 36	Host environments that implement support for PMIx event notification are required to provide the following attributes for all events generated by the environment:
37	PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)

The single process that was affected.

### **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**

1

3

4 5

6 7

# **Data Packing and Unpacking**

PMIx intentionally does not include support for internode communications in the standard, instead relying on its host SMS environment to transfer any needed data and/or requests between nodes. These operations frequently involve PMIx-defined public data structures that include binary data. Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply. However, greater effort is required in heterogeneous environments to ensure binary data is correctly transferred. PMIx buffer manipulation functions are provided for this purpose via standardized 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

Summary 11 12 Allocate memory for a pmix data buffer t object and initialize it **Format** 13 PMIx v2.0 PMIX\_DATA\_BUFFER\_CREATE (buffer); 14 OUT buffer 15 Variable to be assigned the pointer to the allocated **pmix\_data\_buffer\_t** (handle) 16 **Description** 17 18 This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it

### 9.1.2 PMIX DATA BUFFER RELEASE Summary 2 Free a pmix\_data\_buffer\_t object and the data it contains 3 **Format** 4 PMIx v2.0 5 PMIX DATA BUFFER RELEASE (buffer); IN buffer 6 Pointer to the pmix\_data\_buffer\_t to be released (handle) 7 **Description** 8 Free's the data contained in the buffer, and then free's the buffer itself 9 9.1.3 PMIX DATA BUFFER CONSTRUCT Summary 11 12 Initialize a statically declared pmix\_data\_buffer\_t object **Format** 13 PMIx v2.0PMIX DATA\_BUFFER\_CONSTRUCT(buffer); 14 IN buffer 15 Pointer to the allocated pmix\_data\_buffer\_t that is to be initialized (handle) 16 **Description** 17 Initialize a pre-allocated buffer object 18 9.1.4 PMIX DATA BUFFER DESTRUCT Summary 20 21 Release the data contained in a pmix data buffer t object

PMIx Standard – Version 2.1 (Draft) – December 2018

```
Format
1
   PMIx v2.0
2
              PMIX DATA BUFFER DESTRUCT (buffer);
              IN
                   buffer
3
                    Pointer to the pmix data buffer t whose data is to be released (handle)
 4
              Description
5
 6
              Free's the data contained in a pmix_data_buffer_t object
   9.1.5
             PMIX DATA BUFFER LOAD
              Summary
8
9
              Load a blob into a pmix data buffer t object
              Format
10
   PMIx v2.0
              PMIX_DATA_BUFFER_LOAD(buffer, data, size);
11
              IN
                   buffer
12
                    Pointer to a pre-allocated pmix data buffer t (handle)
13
14
              IN
                   data
                    Pointer to a blob (char*)
15
              IN
                   size
16
                    Number of bytes in the blob size_t
17
              Description
18
              Load the given data into the provided pmix_data_buffer_t object, usually done in
19
20
              preparation for unpacking the provided data. Note that the data is not copied into the buffer - thus,
              the blob must not be released until after operations on the buffer have completed.
21
   9.1.6
             PMIX DATA BUFFER UNLOAD
              Summary
23
24
              Unload the data from a pmix data buffer t object
```

### **Format** 1 PMIx v2.0 2 PMIX DATA BUFFER UNLOAD (buffer, data, size); IN 3 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 **Description** 9 10 Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to 11 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 -12 13 thus, the user is responsible for releasing the blob when done with it. 9.2 **General Routines** 15 The following routines are provided to support internode transfers in heterogeneous environments. 9.2.1 PMIx\_Data\_pack Summary 17 18 Pack one or more values of a specified type into a buffer, usually for transmission to another process Format 19

PMIx v2.0

pmix\_status\_t

20

21

22 23

24

PMIx\_Data\_pack(const pmix\_proc\_t \*target,

pmix\_data\_buffer\_t \*buffer,

pmix\_data\_type\_t type);

void \*src, int32 t num vals,

6 7 8

5

9 10 11

16 17

18 19 20

21 22

23 24

25

26 27 28

29 30 31

32 33 34

IN target

Pointer to a pmix\_proc\_t containing the nspace/rank of the process that will be unpacking the final buffer. A NULL value may be used to indicate that the target is based on the same PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

IN buffer

Pointer to a pmix data buffer t where the packed data is to be stored (handle)

IN

Pointer to a location where the data resides. Strings are to be passed as (char \*\*) — i.e., the caller must pass the address of the pointer to the string as the (void\*). This allows the caller to pass multiple strings in a single call. (memory reference)

IN num vals

> Number of elements pointed to by the src pointer. A string value is counted as a single value regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g., string arrays) should be contiguous, although the data pointed to need not be contiguous across array entries.(int32 t)

IN type

The type of the data to be packed (pmix data type t)

**PMIX SUCCESS** The data has been packed as requested PMIX ERR NOT SUPPORTED The PMIx implementation does not support this function. PMIX ERR BAD PARAM The provided buffer or src is NULL PMIX ERR\_UNKNOWN\_DATA\_TYPE The specified data type is not known to this implementation

**PMIX ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation PMIX ERROR General error

### **Description**

The pack function packs one or more values of a specified type into the specified buffer. The buffer must have already been initialized via the PMIX\_DATA\_BUFFER\_CREATE or PMIX DATA BUFFER\_CONSTRUCT macros — otherwise, PMIx\_Data\_pack will return an error. Providing an unsupported type flag will likewise be reported as an error.

Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may lose precision when unpacked by a non-homogeneous recipient. The PMIx Data pack function will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code (generated upon unpacking) — the error cannot be detected during packing.

The namespace of the intended recipient of the packed buffer (i.e., the process that will be unpacking it) is used solely to resolve any data type differences between PMIx versions. The recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the recipient is using. Note that all processes in a given namespace are required to use the same PMIx version — thus, the caller must only know at least one process from the target's namespace.

## 3 9.2.2 PMIx\_Data\_unpack

### Summary

Unpack values from a pmix data buffer t

#### **Format**

PMIx v2.0

C

#### IN source

Pointer to a **pmix\_proc\_t** structure containing the nspace/rank of the process that packed the provided buffer. A NULL value may be used to indicate that the source is based on the same PMIx version as the caller. Note that only the source's nspace is relevant. (handle)

#### IN buffer

A pointer to the buffer from which the value will be extracted. (handle)

#### **INOUT** dest

A pointer to the memory location into which the data is to be stored. Note that these values will be stored contiguously in memory. For strings, this pointer must be to (char\*\*) to provide a means of supporting multiple string operations. The unpack function will allocate memory for each string in the array - the caller must only provide adequate memory for the array of pointers. (void\*)

#### INOUT max num values

The number of values to be unpacked — upon completion, the parameter will be set to the actual number of values unpacked. In most cases, this should match the maximum number provided in the parameters — but in no case will it exceed the value of this parameter. Note that unpacking fewer values than are actually available will leave the buffer in an unpackable state — the function will return an error code to warn of this condition.(int32 t)

#### IN type

The type of the data to be unpacked — must be one of the PMIx defined data types (
pmix\_data\_type\_t)

**PMIX SUCCESS** The data has been unpacked as requested

PMIX\_ERR\_NOT\_SUPPORTED The PMIx implementation does not support this function.

PMIX\_ERR\_BAD\_PARAM The provided buffer or dest is NULL

PMIX\_ERR\_UNKNOWN\_DATA\_TYPE The specified data type is not known to this implementation

PMIX\_ERR\_OUT\_OF\_RESOURCE Not enough memory to support the operation

PMIX\_ERROR General error

### Description

The unpack function unpacks the next value (or values) of a specified type from the given buffer. The buffer must have already been initialized via an PMIX\_DATA\_BUFFER\_CREATE or PMIX\_DATA\_BUFFER\_CONSTRUCT call (and assumedly filled with some data) — otherwise, the unpack\_value function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the unpack\_ptr.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMIx versions. The packer must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the packer is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the packer's namespace.

### 1 9.2.3 PMIx\_Data\_copy

### 2 Summary

3

4

21

22

23 24

25

26

27

29 30 Copy a data value from one location to another.

#### Format

```
PMIx v2.0
5
               pmix status t
               PMIx_Data_copy(void **dest, void *src,
6
 7
                                  pmix data type t type);
8
               IN
                    dest
9
                     The address of a pointer into which the address of the resulting data is to be stored.
                     (void**)
10
               IN
11
                    src
12
                     A pointer to the memory location from which the data is to be copied (handle)
               IN
13
                    type
                     The type of the data to be copied — must be one of the PMIx defined data types. (
14
                     pmix_data_type_t)
15
                PMIX SUCCESS The data has been copied as requested
16
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
17
                PMIX_ERR_BAD_PARAM The provided src or dest is NULL
18
                PMIX ERR UNKNOWN DATA TYPE The specified data type is not known to this
19
20
                     implementation
```

PMIX ERR OUT OF RESOURCE Not enough memory to support the operation

### Description

PMIX ERROR General error

Since registered data types can be complex structures, the system needs some way to know how to copy the data from one location to another (e.g., for storage in the registry). This function, which can call other copy functions to build up complex data types, defines the method for making a copy of the specified data type.

### 9.2.4 PMIx Data print

### Summary

Pretty-print a data value.

```
Format
1
   PMIx v2.0
2
               pmix_status_t
 3
               PMIx_Data_print(char **output, char *prefix,
 4
                                   void *src, pmix_data_type_t type);
               IN
5
                    output
6
                     The address of a pointer into which the address of the resulting output is to be stored.
7
                    (char**)
               IN
8
                    prefix
9
                     String to be prepended to the resulting output (char*)
               IN
10
11
                    A pointer to the memory location of the data value to be printed (handle)
               IN
12
13
                    The type of the data value to be printed — must be one of the PMIx defined data types. (
                    pmix_data_type_t)
14
15
                PMIX SUCCESS The data has been printed as requested
16
                PMIX_ERR_BAD_PARAM The provided data type is not recognized.
17
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
               Description
18
19
               Since registered data types can be complex structures, the system needs some way to know how to
20
               print them (i.e., convert them to a string representation). Primarily for debug purposes.
    9.2.5
             PMIx_Data_copy_payload
               Summary
22
23
               Copy a payload from one buffer to another
               Format
24
   PMIx v2.0
25
               pmix status t
26
               PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
27
                                            pmix data buffer t *src);
```

C

1	IN dest
2	Pointer to the destination <pre>pmix_data_buffer_t</pre> (handle)
3	IN src
4	Pointer to the source <pre>pmix_data_buffer_t</pre> (handle)
5	PMIX_SUCCESS The data has been copied as requested
6	PMIX_ERR_BAD_PARAM The src and dest pmix_data_buffer_t types do not match
7	PMIX ERR NOT SUPPORTED The PMIx implementation does not support this function.

### **Description**

8

9

10

11

This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

#### **CHAPTER 10**

1

2

3

4

5

6

7

8

9

10

16

22

23

24

# **Server-Specific Interfaces**

The RM daemon that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes generating regular expressions, registering information to be passed to client processes, and requesting information on behalf of a remote process. Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

# 10.1 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

## 14 10.1.1 PMIx\_generate\_regex

### 15 **Summary**

Generate a regular expression representation of the input string.

### 17 Format

20 **IN** input 21 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.

#### **Description** 1 2 Given a comma-separated list of *input* values, generate a regular expression that can be passed 3 down to the PMIx client for parsing. The caller is responsible for free'ing the resulting string. 4 If values have leading zero's, then that is preserved, as are prefix and suffix strings. For example, an 5 input string of "odin009.org,odin010.org,odin011.org,odin012.org,odin[102-107].org" 6 will return a regular expression of "pmix:odin[009-012,102-107].org" 7 Advice to users The returned regular expression will have a "pmix:" at the beginning of the string. This informs 8 the PMIx parser that the string was produced using the PRI's regular expression generator, and thus 9 that same plugin should be used for parsing the string 10 10.1.2 PMIx\_generate\_ppn Summary 12 Generate a regular expression representation of the input string. 13 **Format** 14 *PMIx v1.0* pmix\_status\_t PMIx\_generate\_ppn(const char \*input, char \*\*ppn) 15 IN input 16 17 String to process (string) 18 OUT regex 19 Regular expression representation of *input* (string)

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

### 1 Description

2

3

4

5

6 7

10

17

27

28

29

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

### 3 10.1.3 PMIx\_server\_register\_nspace

### 9 Summary

Setup the data about a particular namespace.

#### 11 Format

IN

PMIx v1.0

```
pmix_status_t

PMIx_server_register_nspace(const pmix_nspace_t nspace,

int nlocalprocs,

pmix_info_t info[], size_t ninfo,

pmix_op_cbfunc_t cbfunc, void *cbdata)
```

```
namespace (string)
18
                IN
                      nlocalprocs
19
                      number of local processes (integer)
20
                      info
21
                IN
22
                      Array of info structures (array of handles)
23
                IN
                      ninfo
                      Number of elements in the info array (integer)
24
                IN
25
                      cbfunc
26
```

nspace

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 2 3	• <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.
4 5	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
6 7	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
	▼
8	The following attributes are <i>required</i> to be supported by all PMIx libraries:
9 10 11	<pre>PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool) Registration is for this namespace only, do not copy job data - this attribute is not accessed using the PMIx_Get</pre>
12	Host environments are required to provide the following attributes:
13	• for the session containing the given namespace:
14 15 16 17	- 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_NUM_SLOTS entry in the array - it is included in the Standard for historical reasons.
19	• for the given namespace:
20 21	- PMIX_JOBID "pmix.jobid" (char*)  Job identifier assigned by the scheduler.
22 23	- PMIX_JOB_SIZE "pmix.job.size" (uint32_t)  Total number of processes in this job across all contained applications
24 25	- PMIX_MAX_PROCS "pmix.max.size" (uint32_t)  Maximum number of processes for this job.
26 27	<ul> <li>PMIX_NODE_MAP "pmix.nmap" (char*)</li> <li>Regular expression of nodes - see 10.1.3.1 for an explanation of its generation.</li> </ul>
28 29 30	<ul> <li>PMIX_PROC_MAP "pmix.pmap" (char*)</li> <li>Regular expression describing processes on each node - see 10.1.3.1 for an explanation of its generation.</li> </ul>
31	• for its own node:
32 33	- PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)  Number of processes in this job on this node.
34	- PMIX_LOCAL_PEERS "pmix.lpeers" (char*)

1 2	Comma-delimited list of ranks on this node within the specified namespace - referenced using <b>PMIX_RANK_WILDCARD</b> .
3 4 5	<ul> <li>PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)</li> <li>Colon-delimited cpusets of local peers within the specified namespace - referenced using PMIX_RANK_WILDCARD.</li> </ul>
6	• for each process in the given namespace:
7 8	- PMIX_RANK "pmix.rank" (pmix_rank_t) Process rank within the job.
9 10	- PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)  Local rank on this node within this job.
1  2	<ul><li>- PMIX_NODE_RANK "pmix.nrank" (uint16_t)</li><li>Process rank on this node spanning all jobs.</li></ul>
13 14 15 16	<ul> <li>PMIX_NODEID "pmix.nodeid" (uint32_t)</li> <li>Node identifier where the specified process is located, expressed as the node's index (beginning at zero) in the array resulting from expansion of the PMIX_NODE_MAP regular expression for the job</li> </ul>
17 18	If more than one application is included in the namespace, then the host environment is also <i>required</i> to provide the following attributes:
19	• for each application:
20 21	- PMIX_APPNUM "pmix.appnum" (uint32_t) Application number within the job.
22 23 24	<ul> <li>PMIX_APPLDR "pmix.aldr" (pmix_rank_t)</li> <li>Lowest rank in this application within this job - referenced using</li> <li>PMIX_RANK_WILDCARD.</li> </ul>
25 26	- PMIX_APP_SIZE "pmix.app.size" (uint32_t)  Number of processes in this application.
27	• for each process:
28 29	<pre>- PMIX_APP_RANK "pmix.apprank" (pmix_rank_t) Process rank within this application.</pre>
30 31	- PMIX_APPNUM "pmix.appnum" (uint32_t) Application number within the job.
	<u> </u>

	▼ Optional Attributes
1	The following attributes <i>may</i> be provided by host environments:
2	<ul> <li>for the session containing the given namespace:</li> </ul>
3 4	<pre>- PMIX_SESSION_ID "pmix.session.id" (uint32_t) Session identifier - referenced using PMIX_RANK_WILDCARD.</pre>
5	• for the given namespace:
6 7	<pre>- PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*) Name of the namespace to use for this PMIx server.</pre>
8 9	- PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t) Rank of this PMIx server
10 11	<ul> <li>- PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)</li> <li>Starting global rank of this job - referenced using PMIX_RANK_WILDCARD .</li> </ul>
12 13 14	<ul> <li>- PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)</li> <li>Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using PMIX_RANK_WILDCARD.</li> </ul>
15 16	<pre>- PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)     Number of applications in this job.</pre>
17 18 19 20	<ul> <li>PMIX_MAPBY "pmix.mapby" (char*)</li> <li>Process mapping policy - when accessed using PMIx_Get , use the</li> <li>PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace</li> </ul>
21 22 23 24	<ul> <li>PMIX_RANKBY "pmix.rankby" (char*)</li> <li>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</li> </ul>
25 26 27 28	<ul> <li>PMIX_BINDTO "pmix.bindto" (char*)</li> <li>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</li> </ul>
29	• for its own node:
30 31	<ul> <li>- PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)</li> <li>Total available physical memory on this node.</li> </ul>
32 33	<pre>- PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)</pre>
34	- PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)

1	XML representation of local topology using hwloc's v2.x format.
2 3 4	<pre>- PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)     Lowest rank on this node within this job - referenced using PMIX_RANK_WILDCARD.</pre>
5 6	<ul> <li>- PMIX_NODE_SIZE "pmix.node.size" (uint32_t)</li> <li>Number of processes across all jobs on this node.</li> </ul>
7 8 9	<ul> <li>- PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)</li> <li>Array of pmix_proc_t of all processes on the specified node - referenced using PMIX_RANK_WILDCARD.</li> </ul>
10	<ul> <li>for each process in the given namespace:</li> </ul>
11 12	- PMIX_PROCID "pmix.procid" (pmix_proc_t) Process identifier
13 14	<ul> <li>- PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)</li> <li>Process rank spanning across all jobs in this session.</li> </ul>
15 16	<ul> <li>- PMIX_HOSTNAME "pmix.hname" (char*)</li> <li>Name of the host where the specified process is running.</li> </ul>
17 18 19	Attributes not directly provided by the host environment <i>may</i> be derived by the PMIx server library from other required information and included in the data made available to the server library's 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 23	Host environments are <i>required</i> to execute this operation prior to starting any local application process within the given namespace.
24 25 26 27 28 29	The PMIx server must register <i>all</i> namespaces that will participate in collective operations with local processes. This means that the server must register a namespace even if it will not host any local processes from within that namespace <i>if</i> any local process of another namespace might at some point perform an operation involving one or more processes from the new namespace. This is necessary so that the collective operation can identify the participants and know when it is locally complete.
30 31 32	The caller must also provide the number of local processes that will be launched within this namespace. This is required for the PMIx server library to correctly handle collectives as a collective operation call can occur before all the local processes have been started.

#### Advice to users

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to <code>PMIx\_Spawn</code> result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to <code>migrate</code> to another node via a call to <code>PMIx\_Job\_control\_nb</code>, thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore <code>critical</code> that applications not migrate processes without first ensuring that <code>PMIx-based</code> collective operations are not in progress, and that no such operations be initiated until process migration has completed.

### 10.1.3.1 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it is describes how information provided in the *info* parameter of the **PMIx\_server\_register\_nspace** shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

### - Advice to PMIx server hosts -

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single <code>pmix\_info\_t</code> structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object <code>pmix\_info\_t</code> structures transferred to the corresponding array element utilizing the <code>PMIX\_INFO\_XFER</code> macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a **session** or **job**. PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the **PMIx\_generate\_regex** function to create a regular expression parseable by the PMIx server library, as shown below:

```
char **nodes = NULL;
char *nodelist;
char *regex;
size_t n;
pmix_status_t rc;
pmix_info_t info;
/* loop over an array of nodes, adding each
 * name to the array */
for (n=0; n < num_nodes; n++)</pre>
    /* filter the nodes to ignore those not included
     * in the target range (session, job, etc.). In
     * this example, all nodes are accepted */
    PMIX_ARGV_APPEND(&nodes, node[n]->name);
/* join into a comma-delimited string */
nodelist = PMIX ARGV JOIN(nodes, ',');
/* release the array */
PMIX ARGV FREE (nodes);
/* generate regex */
rc = PMIx_generate_regex(nodelist, &regex);
/* release list */
free (nodelist);
/* pass the regex as the value to the PMIX_NODE_MAP key */
PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
/* release the regex */
free (regex);
```

2

3

4

5

6

7 8

9

10

11

12 13

18 19 20

21

22 23

242526

27

28 29

30 31

32 33

34 35

36

37

38

Changing the filter criteria allows the construction of node maps for any level of information.

A similar method is used to construct the map of processes on each node from the namespace being registered. This may be done for each information level of interest (e.g., to identify the process map for the entire **job** or for each **application** in the job) by changing the search criteria. An 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
            /* loop over an array of nodes */
11
            for (n=0; n < num nodes; n++)
12
                /* for each node, construct an array of ranks on that node */
13
14
                ndppn = NULL;
15
                for (m=0; m < node[n]->num procs; m++)
16
                    /* ignore processes that are not part of the target job */
                    if (!PMIX CHECK NSPACE(targetjob, node[n]->proc[m].nspace))
17
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 */
                localranks = PMIX_ARGV_JOIN(ndppn, ',');
24
                /* release the local array */
25
26
                PMIX ARGV FREE (ndppn);
                /* add this node's contribution to the overall array */
27
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
            /* release the array */
36
37
            PMIX_ARGV_FREE (ppnarray);
38
39
            /* generate ppn regex */
40
            rc = PMIx_generate_ppn(ppn, &regex);
41
42
            /* release list */
```

```
2
3
4
```

```
free(ppn);

/* pass the regex as the value to the PMIX_PROC_MAP key */
PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);
```

Note that the **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

The *info* array at this point might look like (where the labels identify the corresponding attribute - e.g., "Session ID" corresponds to the **PMIX\_SESSION\_ID** attribute):

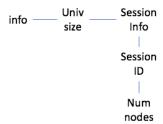


Figure 10.1.: Session-level information elements

Job-level information includes all job-specific values such as PMIX\_JOB\_SIZE,
 PMIX\_JOB\_NUM\_APPS, and PMIX\_JOBID. Since each invocation of
 PMIX\_server\_register\_nspace describes a single job, job-specific values can be specified independently - i.e., in their own pmix\_info\_t elements of the info array.
 Alternatively, they can be provided as a pmix\_data\_array\_t array of pmix\_info\_t identified by the PMIX JOB INFO ARRAY attribute - this is required in cases where

4

5

6

7

8

9

10

11

12 13

14

15 16

17 18

19

20

21

22

**PMIX NSPACE** attribute in the array.

Upon conclusion of this step, the *info* array might look like:

Session Job Info info Session Job ID ID Node Num map nodes Proc map Job size Max

non-specific attributes (e.g., PMIX\_NODE\_MAP) are passed to describe aspects of the job. Note

that since the invocation only involves a single namespace, there is no need to include the

Figure 10.2.: Job-level information elements

procs

Note that in this example, **PMIX NUM NODES** is not required as that information is contained in the PMIX NODE MAP attribute. Similarly, PMIX JOB SIZE is not technically required as that information is contained in the PMIX PROC MAP when combined with the corresponding node map - however, there is no issue with including the job size as a separate entry.

 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 identifed 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:

• 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

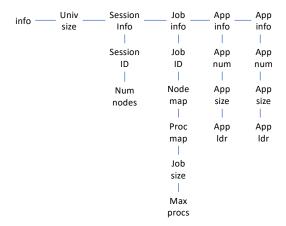


Figure 10.3.: Application-level information elements

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:

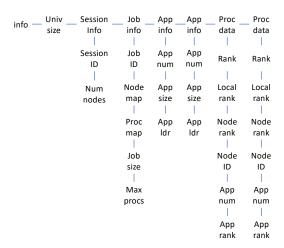


Figure 10.4.: Process-level information elements

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 own pmix\_info\_t elements of the *info* array. Alternatively, they can be provided as a pmix\_data\_array\_t array of pmix\_info\_t using the PMIX\_NODE\_INFO\_ARRAY attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The **PMIX\_LOCAL\_PEERS** value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

```
C
char **ndppn = NULL;
char rank[30];
char *localranks;
size t m;
pmix info t info;
for (m=0; m < mynode->num procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob, mynode->proc[m].nspace))
        continue;
    snprintf(rank, 30, "%d", mynode->proc[m].rank);
    PMIX_ARGV_APPEND(&ndppn, rank);
/* convert the array into a comma-delimited string of ranks */
localranks = PMIX_ARGV_JOIN(ndppn, ',');
/* release the local array */
PMIX_ARGV_FREE (ndppn);
/* pass the string as the value to the PMIX LOCAL PEERS key */
PMIX INFO LOAD (&info, PMIX LOCAL PEERS, localranks, PMIX STRING);
/* release the list */
free(localranks);
```

The PMIX\_LOCAL\_CPUSETS value is constructed in a similar manner. In the provided example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a comma-delimited string of processor IDs) of the processors assigned to each process has previously been generated and stored on the process description. Thus, the value can be constructed as shown below:

C

1

2

3

5

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

```
C
1
              char **ndcpus = NULL;
2
              char *localcpus;
3
              size_t m;
              pmix_info_t info;
4
5
6
              for (m=0; m < mynode->num_procs; m++)
7
                  /* ignore processes that are not part of the target job */
8
                  if (!PMIX_CHECK_NSPACE(targetjob, mynode->proc[m].nspace))
9
                       continue;
10
                  PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
11
12
13
              /* convert the array into a colon-delimited string */
              localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
14
              /* release the local array */
15
16
              PMIX ARGV FREE (ndcpus);
17
18
              /* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
              PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
19
              /* release the list */
20
21
              free(localcpus);
22
                                               C
23
              Note that for efficiency, these two values can be computed at the same time.
24
            The final info array might therefore look like the diagram in 10.5:
   10.1.4
             PMIx server deregister nspace
26
            Summary
27
            Deregister a namespace.
            Format
28
  PMIx v1.0
29
            void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                      pmix op cbfunc t cbfunc, void *cbdata)
30
```

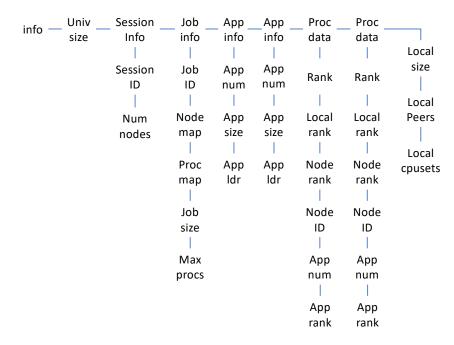


Figure 10.5.: Final information array

IN nspace Namespace (string) IN cbfunc Callback function **pmix\_op\_cbfunc\_t** (function reference) IN Data to be passed to the callback function (memory reference)

### **Description**

Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library *must not* invoke the callback function prior to returning from the API.

#### 10.1.5 PMIx\_server\_register\_client 12

#### Summary 13

Register a client process with the PMIx server library.

1

2

3

4

5

6

7 8

9

10

11

### 1 Format

<i>PMIx v1.0</i>		
2 3 4 5 6	-	<pre>x_status_t x_server_register_client(const pmix_proc_t *proc,</pre>
		С —
7	IN	proc
8		pmix_proc_t structure (handle)
9	IN	uid
10		user id (integer)
11	IN	gid
12		group id (integer)
13	IN	server_object
14		(memory reference)
15	IN	cbfunc
16		Callback function pmix_op_cbfunc_t (function reference)
17	IN	cbdata
18		Data to be passed to the callback function (memory reference)

Returns one of the following:

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

### **Description**

Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server function is called that relates to a specific process. For example, the host server may have an object that tracks the specific client. Passing the object to the library allows the library to provide that object to the host server during subsequent calls related to that client, such as a pmix\_server\_client\_connected\_fn\_t function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

#### Advice to PMIx server hosts -

Host environments are *required* to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate clients as they connect by requiring the two values to match. Accordingly, the detected user and group ID's of the connecting process are not included in the **pmix server client connected fn t** server module function.

### Advice to PMIx library implementers

For security purposes, the PMIx server library should check the user and group ID's of a connecting process against those provided for the declared client process identifier via the PMIx\_server\_register\_client prior to completing the connection.

## 10.1.6 PMIx\_server\_deregister\_client

#### Summary

Deregister a client and purge all data relating to it.

#### Format

PMIx v1.0 void

14 PMIx server deregister client (const. pmix proc. t. \*

IN proc
 pmix\_proc\_t structure (handle)
IN cbfunc
 Callback function pmix\_op\_cbfunc\_t (function reference)
IN cbdata
 Data to be passed to the callback function (memory reference)

### **Description**

The PMIx\_server\_deregister\_nspace API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library *must not* invoke the callback function prior to returning from the API.

### 10.1.7 PMIx server setup fork

### 2 Summary

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

### Format

4

8

9

10

11 12

13 14

15

16

17

18

19

20

23

```
PMIx v1.0

pmix_status_t

PMIx_server_setup_fork(const pmix_proc_t *proc,

char ***env)
```

```
IN proc
pmix_proc_t structure (handle)
IN env
Environment array (array of strings)
```

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

### **Description**

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

#### ——— Advice to PMIx server hosts ————

Host environments are *required* to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via PMIx\_Spawn) plus other values (e.g., variables required to properly initialize the client's fabric library).

### 10.1.8 PMIx\_server\_dmodex\_request

### 22 Summary

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

#### **Format**

PMIx v1.0

C

С

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

Callback function pmix\_dmodex\_response\_fn\_t (function reference)

IN cbdata

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

Returns one of the following:

- PMIX\_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the input the *cbfunc* will *not* be called

### Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the PMIx\_Put and PMIx\_Commit functions followed by a PMIx\_Fence barrier that globally exchanges the posted information. However, the barrier operation represents a signficant 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

1

2

3 4

6

7

8

9

18

19

20

21 22

23 24

25

26

27

28

29

Provide a function by which the resource manager can request application-specific setup data prior to launch of an application.

#### **Format**

```
PMIx v2.0
            pmix_status_t
10
            PMIx_server_setup_application(const pmix_nspace_t nspace,
11
                                       pmix_info_t info[], size_t ninfo,
12
13
                                       pmix setup application cbfunc t cbfunc,
                                       void *cbdata)
14
15
            IN
                 nspace
16
                 namespace (string)
17
```

IN info

Array of info structures (array of handles)

IN ninfo

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

IN

Callback function pmix setup application cbfunc t (function reference)

IN cbdata

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

Returns one of the following:

- PMIX\_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- a PMIx error constant indicating either an error in the input the *cbfunc* will *not* be called

### 1 Description

2

3

5

6

7

8

9

10

11

12

13

16

17

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.

### Advice to PMIx server hosts

Host environments are *required* to execute this operation prior to launching an application.

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.

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.

## Advice to PMIx library implementers

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

## 14 10.1.10 PMIx\_server\_setup\_local\_support

### 15 **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** 1 PMIx v2.0 2 pmix status t 3 PMIx\_server\_setup\_local\_support(const pmix\_nspace\_t nspace, pmix\_info\_t info[], size\_t ninfo, 4 5 pmix\_op\_cbfunc\_t cbfunc, 6 void \*cbdata); 7 IN nspace Namespace (string) 8 9 IN info 10 Array of info structures (array of handles) 11 IN ninfo Number of elements in the *info* array (integer) 12 IN cbfunc 13 Callback function **pmix** op **cbfunc** t (function reference) 14 IN cbdata 15 16 Data to be passed to the callback function (memory reference) 17 Returns one of the following: 18 • PMIX SUCCESS, indicating that the request is being processed by the host environment - result 19 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API. 20 21 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 22 returned success - the cbfunc will not be called • a PMIx error constant indicating either an error in the input or that the request was immediately 23 processed and failed - the cbfunc will not be called 24 **Description** 25 26 Provide a function by which the local PMIx server can perform any application-specific operations 27 prior to spawning local clients of a given application. For example, a network library might need to 28 setup the local driver for "instant on" addressing. The data provided in the info array is the data 29 provided to there host RM from the a call to PMIx server setup application. — Advice to PMIx server hosts ——— 30 Host environments are required to execute this operation prior to starting any local application processes from the specified namespace. 31

# 1 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 <code>pmix\_server\_module\_t</code> structure passed to <code>PMIx\_server\_init</code>. 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

# 7 10.2.1 pmix\_server\_module\_t Module

# 18 Summary

2

3

4

5

6 7

8

9 10

11

12

13

14

15

16

19

List of function pointers that a PMIx server passes to **PMIx\_server\_init** during startup.

### 20 Format

```
\mathcal{C}
```

```
1
            typedef struct pmix server module 2 0 0 t
2
                /* v1x interfaces */
3
                pmix_server_client_connected_fn_t
                                                      client_connected;
4
                pmix_server_client_finalized_fn_t
                                                      client finalized;
5
                pmix_server_abort_fn_t
                                                      abort;
6
                pmix_server_fencenb_fn_t
                                                      fence nb;
7
                pmix_server_dmodex_req_fn_t
                                                      direct modex;
8
                pmix_server_publish_fn_t
                                                      publish;
9
                pmix_server_lookup_fn_t
                                                      lookup;
10
                pmix_server_unpublish_fn_t
                                                      unpublish;
                pmix_server_spawn_fn_t
11
                                                      spawn;
12
                pmix_server_connect_fn_t
                                                      connect;
13
                pmix_server_disconnect_fn_t
                                                      disconnect;
14
                pmix_server_register_events_fn_t
                                                      register_events;
15
                pmix server deregister events fn t
                                                      deregister_events;
16
                pmix server listener fn t
                                                      listener;
17
                /* v2x interfaces */
18
                pmix server notify event fn t
                                                      notify_event;
19
                pmix_server_query_fn_t
                                                      query;
20
                pmix_server_tool_connection_fn_t
                                                      tool_connected;
21
                pmix_server_log_fn_t
                                                      log;
22
                pmix_server_alloc_fn_t
                                                      allocate;
23
                pmix_server_job_control_fn_t
                                                      job_control;
24
                pmix_server_monitor_fn_t
                                                      monitor;
25
             pmix_server_module_t;
```

# 6 10.2.2 pmix\_server\_client\_connected\_fn\_t

# 27 Summary

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

#### 29 **Format**

```
PMIx v1.0

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

1	IN	proc
2		<pre>pmix_proc_t structure (handle)</pre>
3	IN	server_object
4		object reference (memory reference)
5	IN	cbfunc
6		Callback function <b>pmix_op_cbfunc_t</b> (function reference)
7	IN	cbdata
8		Data to be passed to the callback function (memory reference)
9	Retu	rns 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.

# 1 10.2.3 pmix\_server\_client\_finalized\_fn\_t

### 2 Summary

 Notify the host environment that a client called **PMIx\_Finalize**.

### Format

```
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 1 2 pmix server client finalized fn t to NULL. 3 Note that the host server is only being informed that the client has called **PMIx Finalize**. The 4 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. 5 ——— 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 6 also a convenient and well defined time to release resources used to support that client. 7 10.2.4 pmix\_server\_abort\_fn\_t Summary 9 Notify the host environment that a local client called **PMIx\_Abort**. 10 **Format** 11 PMIx v1.0 typedef pmix\_status\_t (\*pmix\_server\_abort\_fn\_t)( 12 13 const pmix\_proc\_t \*proc, 14 void \*server\_object, int status, 15 const char msg[], 16 17 pmix\_proc\_t procs[], size\_t nprocs, 18 19 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 20 21 IN proc 22 pmix\_proc\_t structure identifying the process requesting the abort (handle) 23 IN server object object reference (memory reference) 24

Array of pmix\_proc\_t structures identifying the processes to be terminated (array of

25

26

27

28

29

30 31 IN

IN

IN

status

handles)

exit status (integer)

exit status message (string)

1 2 3 4 5	<ul> <li>IN nprocs         <ul> <li>Number of elements in the procs array (integer)</li> </ul> </li> <li>IN cbfunc         <ul> <li>Callback function pmix_op_cbfunc_t (function reference)</li> </ul> </li> <li>IN cbdata         <ul> <li>Data to be passed to the callback function (memory reference)</li> </ul> </li> </ul>
7	Returns one of the following:
8 9 10	• <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host <i>must not</i> invoke the callback function prior to returning from the API.
11 12	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
13 14	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called
15	Description
16 17 18 19	A local client called <b>PMIx_Abort</b> . Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server library to release the client. The array of <i>procs</i> indicates which processes are to be terminated. A <b>NULL</b> indicates that all processes in the client's namespace are to be terminated.
20 10.2.5	pmix_server_fencenb_fn_t
21	Summary
22	At least one client called either PMIx_Fence or PMIx_Fence_nb.
23	Format
PMIx v1.0  24  25  26  27  28  29  30  31	<pre>typedef pmix_status_t (*pmix_server_fencenb_fn_t)(</pre>

	C
IN	procs
	Array of <pre>pmix_proc_t</pre> structures identifying operation participants(array of handles)
IN	nprocs
	Number of elements in the <i>procs</i> array (integer)
IN	info
	Array of info structures (array of handles)
IN	ninfo
	Number of elements in the <i>info</i> array (integer)
IN	data
	(string)
IN	ndata
	(integer)
IN	cbfunc
	Callback function <pre>pmix_modex_cbfunc_t</pre> (function reference)
IN	cbdata
	Data to be passed to the callback function (memory reference)
Retu	arns one of the following:
W	MIX_SUCCESS, indicating that the request is being processed by the host environment - result ill be returned in the provided <i>cbfunc</i> . Note that the host <i>must not</i> invoke the callback function rior to returning from the API.
	MIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and turned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called
	PMIx error constant indicating either an error in the input or that the request was immediately cocessed and failed - the <i>cbfunc</i> will <i>not</i> be called
<b>—</b> -	Required Attributes
PMI	x 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
1	The following attributes are optional for host environments:
2 3 4 5	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.
6 7 8 9 10	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.
11 12	<pre>PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)     If true, indicates that the requested choice of algorithm is mandatory.</pre>
	Advice to PMIx server hosts
13 14 15	Host environment are <i>required</i> to return <b>PMIX_ERR_NOT_SUPPORTED</b> if passed an attributed marked as <b>PMIX_INFO_REQD</b> that they do not support, even if support for that attribute is optional.
16	Description
17 18 19 20 21 22	All local clients in the provided array of <i>procs</i> called either <b>PMIx_Fence</b> or <b>PMIx_Fence_nb</b> . 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 <i>procs</i> array are required to participate in the <b>PMIx_Fence/PMIx_Fence_nb</b> operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's <b>pmix_server_fencenb_fn_t</b> function.
	Advice to PMIx library implementers —
23 24	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
25 26	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

1

2

3 4

5

6 7

9

10

11

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.

• PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result

#### **Format**

```
12
   PMIx v1.0
13
              typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
                                                    const pmix_proc_t *proc,
14
                                                    const pmix_info_t info[],
15
                                                    size_t ninfo,
16
17
                                                    pmix_modex_cbfunc_t cbfunc,
18
                                                    void *cbdata)
                                                        C
19
              IN
                   proc
                    pmix proc t structure identifying the process whose data is being requested (handle)
20
              IN
21
22
                    Array of info structures (array of handles)
23
              IN
                   ninfo
24
                    Number of elements in the info array (integer)
              IN
25
                    Callback function pmix_modex_cbfunc_t (function reference)
26
27
              IN
                    cbdata
28
                    Data to be passed to the callback function (memory reference)
29
              Returns one of the following:
```

will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function prior to returning from the API.

30

31

1 2	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called				
3 4	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called				
	▼ Required Attributes				
5	PMIx libraries are required to pass any provided attributes to the host environment for processing.				
	▼Optional Attributes				
6	The following attributes are optional for host environments that support this operation:				
7 8 9	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.				
11	Description				
12 13 14	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.				
15 16 17	The array of <i>info</i> 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 <i>unless</i> the <i>mandatory</i> flag has been set - in such cases, the host RM is required to return an				
18	error if the directive cannot be met.				
9 <b>10.2.7</b>	error if the directive cannot be met.				

Publish data per the PMIx API specification.

#### PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_publish\_fn\_t)( 3 const pmix proc t \*proc, 4 const pmix\_info\_t info[], 5 size\_t ninfo, 6 pmix\_op\_cbfunc\_t cbfunc, 7 void \*cbdata) C IN 8 proc 9 pmix\_proc\_t structure of the process publishing the data (handle) 10 IN info 11 Array of info structures (array of handles) IN ninfo 12 Number of elements in the *info* array (integer) 13 IN 14 cbfunc Callback function **pmix** op **cbfunc** t (function reference) 15 IN 16 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 20 prior to returning from the API. 21 22 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 23 returned success - the cbfunc will not be called 24 • a PMIx error constant indicating either an error in the input or that the request was immediately 25 processed and failed - the cbfunc will not be called Required Attributes PMIx libraries are required to pass any provided attributes to the host environment for processing. 26 27 In addition, the following attributes are required to be included in the passed *info* array: 28 PMIX\_USERID "pmix.euid" (uint32\_t) 29 Effective user id. PMIX\_GRPID "pmix.egid" (uint32\_t) 30 31 Effective group id. Host environments that implement this entry point are required to support the following attributes: 32 PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t) 33 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 34

**Format** 

1 2	PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.
	→ Optional Attributes
3	The following attributes are optional for host environments that support this operation:
4 5 6 7	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.
8	Description
9  0  1	Publish data per the <b>PMIx_Publish</b> 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 <b>PMIX_SESSION</b> , and the default persistence <b>PMIX_PERSIST_SESSION</b> or their equivalent. These values can be specified by including the respective attributed in the <i>info</i> array.
13	The persistence indicates how long the server should retain the data.
	Advice to PMIx server hosts
14 15 16 17 18	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.
20 21 22	The <b>PMIX_USERID</b> and <b>PMIX_GRPID</b> of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

# 23 10.2.8 pmix\_server\_lookup\_fn\_t

# Summary

24

25

Lookup published data.

#### PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_lookup\_fn\_t)( 3 const pmix proc t \*proc, 4 char \*\*keys, 5 const pmix\_info\_t info[], 6 size\_t ninfo, 7 pmix\_lookup\_cbfunc\_t cbfunc, void \*cbdata) 8 C IN 9 proc 10 pmix\_proc\_t structure of the process seeking the data (handle) 11 IN kevs (array of strings) 12 IN info 13 Array of info structures (array of handles) 14 IN 15 16 Number of elements in the *info* array (integer) IN 17 Callback function **pmix lookup cbfunc t** (function reference) 18 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 host *must not* invoke the callback function 23 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 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 \_\_\_\_\_ PMIx libraries are required to pass any provided attributes to the host environment for processing. 29 In addition, the following attributes are required to be included in the passed *info* array: 30 31 PMIX\_USERID "pmix.euid" (uint32\_t) Effective user id. 32 PMIX GRPID "pmix.egid" (uint32\_t) 33 Effective group id. 34

**Format** 

Host environments that implement this entry point are required to support the following attributes: 1 PMIX RANGE "pmix.range" (pmix data range t) 2 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 3 4 PMIX\_WAIT "pmix.wait" (int) 5 Caller requests that the PMIx server wait until at least the specified number of values are found (0 indicates all and is the default). 6 ------ Optional Attributes ------7 The following attributes are optional for host environments that support this operation: 8 PMIX\_TIMEOUT "pmix.timeout" (int) 9 Time in seconds before the specified operation should time out ( $\theta$  indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 10 the target process from ever exposing its data. 11 **Description** 12 13 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested. 14 15 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 16 17 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 18 19 be specified on the wait to preclude an indefinite wait for data that may never be published. — Advice to PMIx server hosts —— 20 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 21 guarantee support for any specific range - i.e., the environment does not need to return an error if 22 23 the data store doesn't support a specified range so long as it is covered by some internally defined 24 range.

# 5 10.2.9 pmix\_server\_unpublish\_fn\_t

## Summary

26 27

Delete data from the data store.

#### **Format** 1 PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_unpublish\_fn\_t)( 3 const pmix proc t \*proc, 4 char \*\*keys, 5 const pmix\_info\_t info[], 6 size\_t ninfo, 7 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 8 IN 9 proc 10 pmix\_proc\_t structure identifying the process making the request (handle) 11 IN kevs (array of strings) 12 IN info 13 Array of info structures (array of handles) 14 IN 15 16 Number of elements in the *info* array (integer) IN 17 Callback function **pmix** op **cbfunc t** (function reference) 18 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 host *must not* invoke the callback function 23 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 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 \_\_\_\_\_ PMIx libraries are required to pass any provided attributes to the host environment for processing. 29 In addition, the following attributes are required to be included in the passed *info* array: 30 31 PMIX\_USERID "pmix.euid" (uint32\_t) Effective user id. 32 33 PMIX\_GRPID "pmix.egid" (uint32\_t) Effective group id. 34

1 Host environments that implement this entry point are required to support the following attributes: 2 PMIX RANGE "pmix.range" (pmix data range t) Value for calls to publish/lookup/unpublish or for monitoring event notifications. 3 Optional Attributes The following attributes are optional for host environments that support this operation: 4 5 PMIX\_TIMEOUT "pmix.timeout" (int) 6 Time in seconds before the specified operation should time out ( $\theta$  indicating infinite) in 7 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 8 the target process from ever exposing its data. Description 9 Delete data from the data store. The host server will be passed a **NULL**-terminated array of string 10 keys, plus potential directives such as the data range within which the keys should be deleted. The 11 default data range is left to the host environment, but expected to be PMIX\_SESSION. The 12 callback is to be executed upon completion of the delete procedure. 13 Advice to PMIx server hosts -The PMIX\_USERID and PMIX\_GRPID of the requesting process will be provided to support 14 authorization-based access to published information. The host environment is not required to 15 16 guarantee support for any specific range - i.e., the environment does not need to return an error if 17 the data store doesn't support a specified range so long as it is covered by some internally defined 18 range.

# 19 10.2.10 pmix\_server\_spawn\_fn\_t

# 20 Summary

21

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

#### **Format** 1 PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_spawn\_fn\_t)( 3 const pmix proc t \*proc, 4 const pmix\_info\_t job\_info[], 5 size\_t ninfo, 6 const pmix\_app\_t apps[], 7 size\_t napps, 8 pmix\_spawn\_cbfunc\_t cbfunc, void \*cbdata) 9 IN 10 proc 11 pmix\_proc\_t structure of the process making the request (handle) IN job info 12 Array of info structures (array of handles) 13 IN ninfo 14 15 Number of elements in the *jobinfo* array (integer) 16 IN Array of **pmix\_app\_t** structures (array of handles) 17 IN 18 19 Number of elements in the *apps* array (integer) 20 IN cbfunc 21 Callback function pmix spawn cbfunc t (function reference) 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 26 27 prior to returning from the API. 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 PMIx libraries are required to pass any provided attributes to the host environment for processing. 32

In addition, the following attributes are required to be included in the passed *info* array:

PMIX USERID "pmix.euid" (uint32\_t)

PMIX\_USERID "pmix.euid" (uint32\_t)

Effective user id.

33

34

1 2	<pre>PMIX_GRPID "pmix.egid" (uint32_t) Effective group id.</pre>				
3 4 5 6 7	Host environments that provide this module entry point are required to pass the PMIX_SPAWNED and PMIX_PARENT_ID attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the <i>job_info</i> or the <i>info</i> array of an element of the <i>apps</i> array:				
8 9	<pre>PMIX_WDIR "pmix.wdir" (char*) Working directory for spawned processes.</pre>				
0 1 2 3	PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)  Set the application's current working directory to the session working directory assigned by the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the session working directory assigned to the provided namespace				
4 5	PMIX_PREFIX "pmix.prefix" (char*)  Prefix to use for starting spawned processes.				
6 7	<pre>PMIX_HOST "pmix.host" (char*)</pre>				
8 9	PMIX_HOSTFILE "pmix.hostfile" (char*)  Hostfile to use for spawned processes.				
	▼Optional Attributes				
20	The following attributes are optional for host environments that support this operation:				
21 22	<pre>PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation.</pre>				
23 24	PMIX_ADD_HOST "pmix.addhost" (char*)  Comma-delimited list of hosts to add to the allocation.				
25 26	PMIX_PRELOAD_BIN "pmix.preloadbin" (bool) Preload binaries onto nodes.				
?7 !8	<pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*) Comma-delimited list of files to pre-position on nodes.</pre>				
9 80	PMIX_PERSONALITY "pmix.pers" (char*)  Name of personality to use.				
31 32 33 44	<pre>PMIX_MAPPER "pmix.mapper" (char*)     Mapping mechanism to use for placing spawned processes - when accessed using     PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping     mechanism used for the provided namespace.</pre>				

1 2	PMIX_DISPLAY_MAP "pmix.dispmap" (bool) Display process mapping upon spawn.
3 4	<pre>PMIX_PPR "pmix.ppr" (char*) Number of processes to spawn on each identified resource.</pre>
5 6 7 8	<pre>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</pre>
9 10 11 12	<pre>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</pre>
13 14 15 16	<pre>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</pre>
17 18	PMIX_NON_PMI "pmix.nonpmi" (bool)  Spawned processes will not call PMIx_Init.
19 20	PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive stdin.
21 22	PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)  Forward this process's stdin to the designated process.
23 24	PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)  Forward stdout from spawned processes to this process.
25 26	PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)  Forward stderr from spawned processes to this process.
27 28	PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.
29 30	PMIX_TAG_OUTPUT "pmix.tagout" (bool)  Tag application output with the identity of the source process.
31 32	PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.
33 34	PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  Merge stdout and stderr streams from application processes.
35 36	<pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Output application output to the specified file.</pre>

1 2	PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.
3	PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
4	Number of cpus to assign to each rank - when accessed using PMIx_Get, use the
5	PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
6	provided namespace
7	PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
8	Do not place processes on the head node.
9	PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
10	Do not oversubscribe the cpus.
11	PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
12	Report bindings of the individual processes.
13	PMIX_CPU_LIST "pmix.cpulist" (char*)
14	List of cpus to use for this job - when accessed using PMIx_Get, use the
15	PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided
16	namespace
17	PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
18	Application supports recoverable operations.
19	PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
20	Application is continuous, all failed processes should be immediately restarted.
21	PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
22	Maximum number of times to restart a job - when accessed using PMIx_Get, use the
23	PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
24	namespace
25	PMIX_TIMEOUT "pmix.timeout" (int)
26	Time in seconds before the specified operation should time out $(0 \text{ indicating infinite})$ in
27	error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
28	the target process from ever exposing its data.

## Description

1

3

4

5

6

7

8

9

11

12

13

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
14
               typedef pmix_status_t (*pmix_server_connect_fn_t)(
                                                    const pmix_proc_t procs[],
15
16
                                                    size_t nprocs,
                                                    const pmix_info_t info[],
17
                                                    size_t ninfo,
18
19
                                                    pmix_op_cbfunc_t cbfunc,
20
                                                    void *cbdata)
               IN
21
                    procs
22
                    Array of pmix_proc_t structures identifying participants (array of handles)
               IN
23
24
                    Number of elements in the procs array (integer)
25
               IN
                    info
                    Array of info structures (array of handles)
26
27
               IN
                    ninfo
                    Number of elements in the info array (integer)
28
29
               IN
                    cbfunc
                    Callback function pmix_op_cbfunc_t (function reference)
30
               IN
31
                    cbdata
                    Data to be passed to the callback function (memory reference)
32
33
               Returns one of the following:
```

1 2 3	• <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host <i>must not</i> invoke the callback function prior to returning from the API.						
4 5	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called						
6 7	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called						
	▼						
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:						
0 1 2 3	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.						
4 5 6 7 8	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.						
9	<pre>PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool) If true, indicates that the requested choice of algorithm is mandatory.</pre>						

# 1 Description

2

3

4

5

6 7

8

9

10

13

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix\_server\_connect\_fn\_t** function, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

# Advice to PMIx library implementers —

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

### Advice to PMIx server hosts —

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

# 11 10.2.12 pmix\_server\_disconnect\_fn\_t

## 12 Summary

Disconnect a previously connected set of processes.

#### **Format** 1 PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_disconnect\_fn\_t)( 3 const pmix proc t procs[], 4 size\_t nprocs, 5 const pmix\_info\_t info[], 6 size\_t ninfo, 7 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 8 IN 9 procs 10 Array of pmix\_proc\_t structures identifying participants (array of handles) 11 IN nprocs Number of elements in the *procs* array (integer) 12 IN info 13 Array of info structures (array of handles) 14 IN ninfo 15 16 Number of elements in the *info* array (integer) IN 17 Callback function **pmix** op **cbfunc** t (function reference) 18 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 host *must not* invoke the callback function 23 prior to returning from the API. 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.

**A**-----

	▼ Optional Attributes						
1	The following attributes are optional for host environments that support this operation:						
2 3 4 5	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.						
6	Description						
7 8 9 10	Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server's has called the <pre>pmix_server_disconnect_fn_t</pre> function, and the host environment has completed any required supporting operations.						
	Advice to PMIx library implementers —						
11 12	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						
13 14 15	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.						
16 17	A <b>PMIX_ERR_INVALID_OPERATION</b> error must be returned if the specified set of <i>procs</i> was not previously <i>connected</i> via a call to the <b>pmix_server_connect_fn_t</b> function.						

# 8 10.2.13 pmix\_server\_register\_events\_fn\_t

# Summary

Register to receive notifications for the specified events.

#### **Format** 1 PMIx v1.0 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, void \*cbdata) 8 IN 9 codes 10 Array of pmix\_status\_t values (array of handles) 11 IN ncodes Number of elements in the *codes* array (integer) 12 IN info 13 Array of info structures (array of handles) 14 IN ninfo 15 16 Number of elements in the *info* array (integer) IN 17 Callback function **pmix** op **cbfunc t** (function reference) 18 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 host *must not* invoke the callback function 23 prior to returning from the API. 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 PMIx libraries are required to pass any provided attributes to the host environment for processing. 29 In addition, the following attributes are required to be included in the passed *info* array: 30 31 PMIX\_USERID "pmix.euid" (uint32\_t) Effective user id. 32 PMIX GRPID "pmix.egid" (uint32\_t) 33 Effective group id. 34

1	$\sim$	•^	r	-	••	$\sim$	n
.,	es	<b>S</b> L -		.,		.,	
_	$\overline{}$	,,		r	•	•	

1

3

4

5

6

7

8

10

11

12 13

14

17

Register to receive notifications for the specified status codes. The *info* array included in this API is reserved for possible future directives to further steer notification.

### Advice to PMIx library implementers ——

The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

- the client has requested notification of an environmental code (i.e., a PMIx code in the range between PMIX\_ERR\_SYS\_BASE and PMIX\_ERR\_SYS\_OTHER, inclusive) or a code that lies outside the defined PMIx range of constants; and
- the PMIx server library has not previously requested notification of that code i.e., the host environment is to be contacted only once a given unique code value

# Advice to PMIx server hosts

The host environment is *required* to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between PMIX\_ERR\_SYS\_BASE and PMIX\_ERR\_SYS\_OTHER (inclusive).

# 10.2.14 pmix server deregister events fn t

# 16 Summary

Deregister to receive notifications for the specified events.

#### PMIx v1.0 typedef pmix\_status\_t (\*pmix\_server\_deregister\_events\_fn\_t)( 2 3 pmix status t \*codes, 4 size\_t ncodes, 5 pmix\_op\_cbfunc\_t cbfunc, 6 void \*cbdata) 7 IN codes 8 Array of pmix\_status\_t values (array of handles) 9 IN ncodes 10 Number of elements in the *codes* array (integer) 11 IN cbfunc Callback function pmix\_op\_cbfunc\_t (function reference) 12 IN cbdata 13 Data to be passed to the callback function (memory reference) 14 15 Returns one of the following: • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result 16 will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function 17 18 prior to returning from the API. • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 19 returned success - the cbfunc will not be called 20 • a PMIx error constant indicating either an error in the input or that the request was immediately 21 22 processed and failed - the *cbfunc* will *not* be called **Description** 23 24 Deregister to receive notifications for the specified events to which the PMIx server has previously registered. 25 Advice to PMIx library implementers — The PMIx server library must track all client registrations. This module function shall only be 26 called when: 27 • the library is deregistering environmental codes (i.e., a PMIx codes in the range between 28 29 PMIX ERR SYS BASE and PMIX ERR SYS OTHER, inclusive) or codes that lies outside the defined PMIx range of constants; and 30 • no client (including the server library itself) remains registered for notifications on any included 31 32 code - i.e., a code should be included in this call only when no registered notifications against it 33 remain.

**Format** 

# 10.2.15 pmix\_server\_notify\_event\_fn\_t

Summary

2

#### 3 Notify the specified processes of an event. **Format** 4 PMIx v2.0typedef pmix\_status\_t (\*pmix\_server\_notify\_event\_fn\_t)(pmix\_status\_t code, 5 6 const pmix\_proc\_t \*source, 7 pmix\_data\_range\_t range, 8 pmix info t info[], 9 size\_t ninfo, 10 pmix\_op\_cbfunc\_t cbfunc, 11 void \*cbdata); 12 IN code The pmix status t event code being referenced structure (handle) 13 IN source 14 pmix\_proc\_t of process that generated the event (handle) 15 16 IN range 17 pmix data range t range over which the event is to be distributed (handle) IN info 18 Optional array of pmix\_info\_t structures containing additional information on the event 19 20 (array of handles) 21 IN ninfo 22 Number of elements in the *info* array (integer) IN cbfunc 23 24 Callback function **pmix\_op\_cbfunc\_t** (function reference) IN cbdata 25 Data to be passed to the callback function (memory reference) 26 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 prior to returning from the API. 30 31 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 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 1 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. Host environments that provide this module entry point are required to support the following 4 5 attributes: PMIX RANGE "pmix.range" (pmix\_data\_range\_t) 6 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 7 **▲**------**Description** 8 9 Notify the specified processes (described through a combination of range and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. 10 The process generating the event is provided in the *source* parameter, and any further descriptive 11 12 information is included in the info array. Advice to PMIx server hosts — 13 The callback function is to be executed once the host environment no longer requires that the PMIx 14 server library maintain the provided data structures. It does not necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery 15

# 6 **10.2.16 pmix\_server\_listener\_fn\_t**

# 17 Summary

18

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

#### **Format** 1 PMIx v1.0 2 typedef pmix\_status\_t (\*pmix\_server\_listener\_fn\_t)( 3 int listening sd, 4 pmix\_connection\_cbfunc\_t cbfunc, 5 void \*cbdata) 6 IN incoming sd 7 (integer) IN cbfunc 8 9 Callback function pmix\_connection\_cbfunc\_t (function reference) cbdata 10 IN 11 (memory reference) 12 Returns **PMIX\_SUCCESS** indicating that the request is accepted, or a negative value 13 corresponding to a PMIx error constant indicating that the request has been rejected. **Description** 14 15 Register a socket the host environment can monitor for connection requests, harvest them, and then call the PMIx server library's internal callback function for further processing. A listener thread is 16 17 essential to efficiently harvesting connection requests from large numbers of local clients such as occur when running on large SMPs. The host server listener is required to call accept on the 18 incoming connection request, and then pass the resulting socket to the provided cbfunc. A NULL 19 20 for this function will cause the internal PMIx server to spawn its own listener thread. 10.2.17 pmix server query fn t Summary 22 23 Query information from the resource manager. **Format** 24 PMIx v2.025 typedef pmix\_status\_t (\*pmix\_server\_query\_fn\_t)( 26 pmix\_proc\_t \*proct, 27 pmix query t \*queries, size t nqueries,

pmix info cbfunc t cbfunc,

void \*cbdata)

28

1	IN proct
2	<pre>pmix_proc_t structure of the requesting process (handle)</pre>
3	IN queries
4	Array of pmix_query_t structures (array of handles)
5	IN nqueries  Number of elements in the queries error (integer)
6 7	Number of elements in the <i>queries</i> array (integer)  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 <i>cbfunc</i> . Note that the host <i>must not</i> 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 <i>success</i> - the <i>cbfunc</i> will <i>not</i> 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 <i>cbfunc</i> will <i>not</i> be called
	▼ Required Attributes
19 20	PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:
21	PMIX USERID "pmix.euid" (uint32 t)
22	Effective user id.
23	
23 24	PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.
- •	A.
	▼ Optional Attributes
_	·
25	The following attributes are optional for host environments that support this operation:
26	<pre>PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)</pre>
27	Request a comma-delimited list of active namespaces.
28	<pre>PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)</pre>
29	Status of a specified, currently executing job.
30	PMIX OUERY OUEUE LIST "pmix.grv.glst" (char*)
	Request a comma-delimited list of scheduler queues.
32	•
29 30 31	PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*)

1	Status of a specified scheduler queue.
2 3 4	PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns ( pmix_data_array_t) an array of pmix_proc_info_t.
5 6 7 8	PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*) Input namespace of the job whose information is being requested returns ( pmix_data_array_t) an array of pmix_proc_info_t for processes in job on same node.
9 10	PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes.
11 12	PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes.
13 14	PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)  Return information on memory usage for the processes indicated in the qualifiers.
15 16	PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)  Constrain the query to local information only.
17 18	PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool) Report average values.
19 20	PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values.
21 22	<pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested.</pre>
23 24 25	PMIX_TIME_REMAINING "pmix.time.remaining" (char*)  Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
26	Description
27 28 29	Query information from the host environment. The query will include the namespace/rank of the process that is requesting the info, an array of <pre>pmix_query_t</pre> describing the request, and a callback function/data for the return.  Advice to PMIx library implementers
30 31	The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

#### 10.2.18 pmix\_server\_tool\_connection\_fn\_t Summary 2 Register that a tool has connected to the server. 3 **Format** 4 PMIx v2.05 typedef void (\*pmix\_server\_tool\_connection\_fn\_t)( 6 pmix\_info\_t info[], size\_t ninfo, 7 pmix\_tool\_connection\_cbfunc\_t cbfunc, 8 void \*cbdata) 9 IN info Array of **pmix\_info\_t** structures (array of handles) 10 11 IN Number of elements in the *info* array (integer) 12 IN cbfunc 13 Callback function pmix\_tool\_connection\_cbfunc\_t (function reference) 14 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) Effective user id. 19 20 PMIX\_GRPID "pmix.egid" (uint32\_t) 21 Effective group id. The following attributes are optional for host environments that support this operation: 22 23 PMIX\_FWD\_STDOUT "pmix.fwd.stdout" (bool)

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.

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

## Description

1

3

4

5

6

7 8

11

12

18

19

20 21

22

23

24

25 26

27

28 29

30 31 Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The <code>pmix\_info\_t</code> 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

## 10 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)

C
```

```
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 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.

# Description

1 2

3

5 6

7

9

10 11

12 13

14

15 16

17 18

19 20

21 22

23

24

25

26 27

28

29

Log data on behalf of a client. This function is *not* intended for output of computational results, but rather for reporting status and error messages. The host *must not* execute the callback function prior to returning from the API.

## 10.2.20 pmix\_server\_alloc\_fn\_t

#### Summary 2 3 Request allocation operations on behalf of a client. **Format** 4 PMIx v2.0typedef pmix\_status\_t (\*pmix\_server\_alloc\_fn\_t)( 5 6 const pmix proc t \*client, 7 pmix alloc directive t directive, 8 const pmix info t data[], size t ndata, 9 pmix info cbfunc t cbfunc, void \*cbdata) IN client 10 pmix\_proc\_t structure of process making request (handle) 11 IN 12 directive Specific action being requested ( pmix\_alloc\_directive\_t ) 13 IN 14 Array of info structures (array of handles) 15 16 IN ndata 17 Number of elements in the *data* array (integer) IN 18 19 Callback function pmix\_info\_cbfunc\_t (function reference) IN cbdata 20 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 will be returned in the provided *cbfunc*. Note that the host *must not* invoke the callback function 24 25 prior to returning from the API. 26 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called 27 28 • a PMIx error constant indicating either an error in the input or that the request was immediately 29 processed and failed - the cbfunc will not be called

#### Required Attributes -----PMIx libraries are required to pass any provided attributes to the host environment for processing. 1 2 In addition, the following attributes are required to be included in the passed *info* array: 3 PMIX USERID "pmix.euid" (uint32 t) Effective user id. PMIX GRPID "pmix.egid" (uint32\_t) 5 6 Effective group id. 7 Host environments that provide this module entry point are required to support the following attributes: 8 9 PMIX\_ALLOC\_ID "pmix.alloc.id" (char\*) Provide a string identifier for this allocation request which can later be used to query status 10 of the request. 11 PMIX ALLOC NUM NODES "pmix.alloc.nnodes" (uint64 t) 12 The number of nodes. 13 14 PMIX ALLOC NUM CPUS "pmix.alloc.ncpus" (uint64 t) Number of cpus. 15 PMIX ALLOC TIME "pmix.alloc.time" (uint32 t) 16 17 Time in seconds. \_\_\_\_\_\_ Optional Attributes -----The following attributes are optional for host environments that support this operation: 18 PMIX ALLOC NODE LIST "pmix.alloc.nlist" (char\*) 19 Regular expression of the specific nodes. 20 21 PMIX ALLOC NUM CPU LIST "pmix.alloc.ncpulist" (char\*) 22 Regular expression of the number of cpus for each node. PMIX ALLOC CPU LIST "pmix.alloc.cpulist" (char\*) 23 24 Regular expression of the specific cpus indicating the cpus involved. PMIX ALLOC MEM SIZE "pmix.alloc.msize" (float) 25 Number of Megabytes. 26 PMIX\_ALLOC\_NETWORK "pmix.alloc.net" (array) 27 28 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 29 applied across all nodes in the requestor's allocation. 30 PMIX ALLOC NETWORK ID "pmix.alloc.netid" (char\*) 31 32 Name of the network.

```
PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
 1
 2
                      Mbits/sec.
 3
                PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
                      Ouality of service level.
 4
                Description
 5
                Request new allocation or modifications to an existing allocation on behalf of a client. Several
 6
 7
                broad categories are envisioned, including the ability to:
                • Request allocation of additional resources, including memory, bandwidth, and compute for an
 8
 9
                  existing allocation. Any additional allocated resources will be considered as part of the current
10
                  allocation, and thus will be released at the same time.
                • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
11
12
                  affiliated with) the allocation of the requestor - thus the termination of one allocation will not
                  impact the other.
13
14
                • Extend the reservation on currently allocated resources, subject to scheduling availability and
15
                  priorities.
                • Return no-longer-required resources to the scheduler. This includes the loan of resources back to
16
17
                  the scheduler with a promise to return them upon subsequent request.
18
                The callback function provides a status to indicate whether or not the request was granted, and to
19
                provide some information as to the reason for any denial in the pmix_info_cbfunc_t array of
20
                pmix_info_t structures.
    10.2.21
                  pmix server_job_control_fn_t
21
                Summary
22
23
                Execute a job control action on behalf of a client.
                Format
24
   PMIx v2.0
                typedef pmix_status_t (*pmix_server_job_control_fn_t)(
25
26
                                                       const pmix proc t *requestor,
27
                                                       const pmix_proc_t targets[], size_t ntargets,
```

const pmix info t directives[], size t ndirs,

pmix info cbfunc t cbfunc, void \*cbdata)

28

29

1	IN requestor		
2	<pre>pmix_proc_t structure of requesting process (handle)</pre>		
3	IN targets		
4	Array of proc structures (array of handles)		
5	IN ntargets		
6	Number of elements in the targets array (integer)		
7	IN directives		
8	Array of info structures (array of handles)		
9	IN ndirs		
0	Number of elements in the <i>info</i> array (integer)		
1	IN cbfunc		
2	Callback function pmix_op_cbfunc_t (function reference)		
3	IN cbdata		
4	Data to be passed to the callback function (memory reference)		
5	Returns one of the following:		
6	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - resul		
7	will be returned in the provided <i>cbfunc</i> . Note that the host <i>must not</i> invoke the callback function		
8	prior to returning from the API.		
9	• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and		
20	returned success - the cbfunc will not be called		
21	• a PMIx error constant indicating either an error in the input or that the request was immediately		
22	processed and failed - the <i>cbfunc</i> will <i>not</i> be called		
	Required Attributes		
23 24	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 <i>info</i> array:		
25	PMIX_USERID "pmix.euid" (uint32_t)		
26	Effective user id.		
27	<pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>		
28	Effective group id.		
<u>.</u> 9	Host environments that provide this module entry point are required to support the following		
10	attributes:		
31 	PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)		
32	Provide a string identifier for this request.		
3	PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)		
34	Pause the specified processes.		
35	PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)		

```
Resume ("un-pause") the specified processes.
 1
              PMIX JOB CTRL KILL "pmix.jctrl.kill" (bool)
 2
                   Forcibly terminate the specified processes and cleanup.
 3
              PMIX JOB CTRL SIGNAL "pmix.jctrl.sig" (int)
 4
5
                   Send given signal to specified processes.
6
              PMIX JOB CTRL TERMINATE "pmix.jctrl.term" (bool)
7
                   Politely terminate the specified processes.
                _____
                                            Optional Attributes
              The following attributes are optional for host environments that support this operation:
8
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
9
                   Cancel the specified request (NULL implies cancel all requests from this requestor).
10
11
              PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
                   Restart the specified processes using the given checkpoint ID.
12
              PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
13
                   Checkpoint the specified processes and assign the given ID to it.
14
              PMIX JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
15
                   Use event notification to trigger a process checkpoint.
16
17
              PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
18
                   Use the given signal to trigger a process checkpoint.
19
              PMIX JOB CTRL CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
                   Time in seconds to wait for a checkpoint to complete.
20
21
              PMIX_JOB_CTRL_CHECKPOINT_METHOD
22
              "pmix.jctrl.ckmethod" (pmix_data_array_t)
                   Array of pmix_info_t declaring each method and value supported by this application.
23
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
24
                   Regular expression identifying nodes that are to be provisioned.
25
              PMIX JOB CTRL PROVISION IMAGE "pmix.jctrl.pvnimg" (char*)
26
                   Name of the image that is to be provisioned.
27
              PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
28
29
                   Indicate that the job can be pre-empted.
```

### Description

1

3

4

5

6 7

8

9

11

12

13

Execute a job control action on behalf of a client. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX\_RANK\_WILDARD** can also be used to indicate that all processes in the given namespace are to be included.

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

## 10.2.22 pmix\_server\_monitor\_fn\_t

#### Summary

Request that a client be monitored for activity.

### Format

```
PMIx v2.0
14
              /* Request that a client be monitored for activity */
              typedef pmix_status_t (*pmix_server_monitor_fn_t)(
15
                                                  const pmix_proc_t *requestor,
16
                                                  const pmix_info_t *monitor, pmix_status_t error
17
                                                  const pmix_info_t directives[], size_t ndirs,
18
                                                  pmix_info_cbfunc_t cbfunc, void *cbdata);
19
20
              IN
                   requestor
21
                   pmix_proc_t structure of requesting process (handle)
22
              IN
                   monitor
23
                   pmix info t identifying the type of monitor being requested (handle)
24
              IN
25
                   Status code to use in generating event if alarm triggers (integer)
              IN
26
                   directives
                   Array of info structures (array of handles)
27
              IN
28
29
                   Number of elements in the info array (integer)
30
              IN
                   Callback function pmix_op_cbfunc_t (function reference)
31
32
              IN
                   cbdata
33
                    Data to be passed to the callback function (memory reference)
```

1	Returns one of the following:		
2 3 4	• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host <i>must not</i> invoke the callback function prior to returning from the API.		
5 6	<ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called</li> </ul>		
7 8	• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called		
9 10	This entry point is only called for monitoring requests that are not directly supported by the PMIx server library itself.		
	▼		
11 12 13 14	If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed <i>info</i> array:		
15 16	PMIX_USERID "pmix.euid" (uint32_t) Effective user id.		
17 18	<pre>PMIX_GRPID "pmix.egid" (uint32_t)     Effective group id.</pre>		
19	Host environments are not required to support any specific monitoring attributes.		
	▼Optional Attributes		
20	The following attributes may be implemented by a host environment.		
21 22	PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.		
23 24	<pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)     Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>		
25 26	<pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>		
27 28	PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)  Register to have the PMIx server monitor the requestor for heartbeats.		
29 30	<pre>PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.</pre>		
31	<pre>PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)</pre>		

1	Number of heartbeats that can be missed before generating the event.
2 3	<pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>
4 5	<pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)  Monitor size of given file is growing to determine if the application is running.</pre>
6 7	<pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre>
8 9	<pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)  Monitor time since last modified of given file to determine if the application is running.</pre>
10 11	<pre>PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t) Time in seconds between checking the file.</pre>
12 13	PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)  Number of file checks that can be missed before generating the event.
14	Description
15	Request that a client be monitored for activity.
	Advice to PMIx server hosts
16 17 18	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 <b>PMIX_ERR_NOT_SUPPORTED</b> to the provided <i>cbfunc</i> .

### **APPENDIX A**

# Acknowledgements

1 2 3 4	This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.
5 <b>A.1</b>	Version 2.0
6	The following list includes some of the active participants in the PMIx v2 standardization process.
7 8	<ul> <li>Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm and Terry Wilmarth</li> </ul>
9	• Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
0	Aurelien Bouteiller and George Bosilca
1	<ul> <li>Artem Polyakov, Igor Ivanov and Boris Karasev</li> </ul>
2	Gilles Gouaillardet
3	Michael A Raymond and Jim Stoffel
4	• Dirk Schubert
5	Moe Jette
6	Takahiro Kawashima and Shinji Sumimoto
7	Howard Pritchard
8	David Beer
9	Brice Goglin
0	• Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
1	Adam Moody and Martin Schulz
2	Ryan Grant and Stephen Olivier

• Michael Karo

23

The following institutions supported this effort through time and travel support for the people listed 1 2 above. • Intel Corporation • IBM. Inc. 4 5 • University of Tennessee, Knoxville • The Exascale Computing Project, an initiative of the US Department of Energy 6 7 National Science Foundation • Mellanox, Inc. 8 9 Research Organization for Information Science and Technology • HPE Co. 10 • Allinea (ARM) 11 12 • SchedMD, Inc. • Fujitsu Limited 13 14 • Los Alamos National Laboratory 15 • Adaptive Solutions, Inc. INRIA 16 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 • Gilles Gouaillardet 27

28

• Gary Brown

- Moe Jette 1
- The following institutions supported this effort through time and travel support for the people listed 2 above.
- Intel Corporation 4
- IBM, Inc. 5

8

- University of Tennessee, Knoxville 6
- Mellanox, Inc. 7
  - Research Organization for Information Science and Technology
- Adaptive Solutions, Inc. 9
- SchedMD, Inc. 10

# **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, 57, 58, 99, 107, 154, 193, 196	PMIX_ALLOC_NETWORK_ID, 156, 245 Defintion, 69
Defintion, 12	PMIX_ALLOC_NETWORK_QOS, 157, 246
host environment	Defintion, 69
Defintion, 13	PMIX_ALLOC_NEW, 37
	PMIX_ALLOC_NODE_LIST, 156, 245
job, 10, 57, 58, 99, 107, 109, 154, 189, 192,	Defintion, 69
193, 195, 196	PMIX_ALLOC_NUM_CPU_LIST, 156, 245
Defintion, 12	Defintion, 69
	PMIX_ALLOC_NUM_CPUS, 156, 245
namespace	Defintion, 69
Defintion, 12	PMIX_ALLOC_NUM_NODES, 156, 245
DMI Alexa 0 25 120 212 212	Defintion, 69
PMIx_Abort, 8, 25, 130, 212, 213	PMIX_ALLOC_REAQUIRE, 37
Defintion, 129	PMIX_ALLOC_RELEASE, 37
PMIX_ADD_HOST, 132, 136, 225	PMIX_ALLOC_TIME, 156, 245
Defintion, 64	Defintion, 70
PMIX_ADD_HOSTFILE, 132, 136, 225 Defintion, <b>64</b>	PMIX_ALLOCATED_NODELIST, 190
PMIX_ALLOC_BANDWIDTH, 156, 246	Defintion, 57
Defintion, 69	PMIx_Allocation_request_nb, 9, 69, 148,
PMIX_ALLOC_CPU_LIST, 156, 245	157
Defintion, 69	Defintion, 155
PMIX_ALLOC_DIRECTIVE, 51	PMIX_ANL_MAP
PMIx_Alloc_directive_string, 9	Defintion, 62
Defintion, 85	PMIX_APP, 51
pmix_alloc_directive_t, 37, 51, 85, 244	PMIX_APP_CONSTRUCT
Defintion, <b>37</b>	Defintion, 41
PMIX_ALLOC_EXTEND, 37	PMIX_APP_CREATE
PMIX_ALLOC_EXTERNAL, 37	Defintion, 42
PMIX_ALLOC_ID, 156, 245	PMIX_APP_DESTRUCT
Defintion, 69	Defintion, 42
PMIX_ALLOC_MEM_SIZE, 156, 245	PMIX_APP_FREE
Defintion, 69	Defintion, 42
PMIX_ALLOC_NETWORK, 156, 245	
	PMIX_APP_INFO, 102, 105, 110, 151

PMIX_APP_INFO_ARRAY, 196	PMIX_COLLECTIVE_ALGO, 10, 113,
Defintion, 58	116, 140, 143, 215, 229
PMIX_APP_MAP_REGEX	Defintion, 61
Defintion, 62	PMIX_COLLECTIVE_ALGO_REQD, 113
PMIX_APP_MAP_TYPE	116, 140, 143, 215, 229
Defintion, 62	Defintion, 61
PMIX_APP_RANK, 189	PMIX_COMMAND, 51
Defintion, 56	PMIx_Commit, 8, 82, 100, 112, 139, 204,
PMIX_APP_SIZE, 109, 189, 196	217
Defintion, 59	Defintion, 112
pmix_app_t, 41, 42, 131, 135, 224	PMIX_COMPRESSED_STRING, 51
Defintion, 41	PMIx_Connect, 8, 10, 19, 134, 139, 141,
PMIX_APPLDR, 189, 196	142, 144, 145
Defintion, 56	Defintion, 139
PMIX_APPNUM, 58, 102, 105, 109, 151,	PMIX_CONNECT_MAX_RETRIES, 92
189, 196	Defintion, 53
Defintion, 56	PMIx_Connect_nb, 8, 142
PMIX_ARCH	Defintion, 142
Defintion, 55	PMIX_CONNECT_RETRY_DELAY, 92
PMIX_ATTR_UNDEF	Defintion, 53
Defintion, 52	PMIX_CONNECT_SYSTEM_FIRST, 92,
PMIX_AVAIL_PHYS_MEMORY, 190	94
Defintion, 59	Defintion, 53
PMIX_BINDTO, 133, 137, 190, 226	PMIX_CONNECT_TO_SYSTEM, 92, 94
Defintion, 65	Defintion, 53
PMIX_BOOL, 50	pmix_connection_cbfunc_t, 238
PMIX_BUFFER, 51	Defintion, 82
PMIX_BYTE, 50	PMIX_COSPAWN_APP
PMIX_BYTE_OBJECT, 51	Defintion, 65
PMIX_BYTE_OBJECT_CREATE	PMIX_CPU_LIST, 134, 138, 227
Defintion, 47	Defintion, 66
PMIX_BYTE_OBJECT_DESTRUCT	PMIX_CPUS_PER_PROC, 133, 138, 227
Defintion, 47	Defintion, 66
PMIX_BYTE_OBJECT_FREE	PMIX_CPUSET
Defintion, 47	Defintion, 55
PMIX_BYTE_OBJECT_LOAD	PMIX_CREDENTIAL
Defintion, 48	Defintion, 55
pmix_byte_object_t, 46–48, 51	PMIX_DAEMON_MEMORY
Defintion, 46	Defintion, 59
PMIX_CLIENT_AVG_MEMORY	PMIX_DATA_ARRAY, 51
Defintion, 59	pmix_data_array_t, 29, 50, 51, 67, 152,
PMIX_COLLECT_DATA, 113, 115, 214	195–197, 240
Defintion, 60	Defintion, 50

PMIX_DATA_BUFFER_CONSTRUCT,	Defintion, 68
179, 181	PMIX_DEBUG_STOP_ON_EXEC
Defintion, 176	Defintion, 68
PMIX_DATA_BUFFER_CREATE, 179,	PMIX_DEBUG_WAIT_FOR_NOTIFY
181	Defintion, 68
Defintion, 49, 175	PMIX_DEBUG_WAITING_FOR_NOTIFY
PMIX_DATA_BUFFER_DESTRUCT	Defintion, 68
Defintion, 49, 176	PMIX_DEBUGGER_DAEMONS, 133, 137,
PMIX_DATA_BUFFER_LOAD	226
Defintion, 177	Defintion, 65
PMIX_DATA_BUFFER_RELEASE	PMIx_Deregister_event_handler, 9
Defintion, 49, 176	Defintion, 171
pmix_data_buffer_t, 48, 49, 175–180, 184	PMIx_Disconnect, 8, 10, 19, 142, 145–147
Defintion, 48	Defintion, 144
PMIX_DATA_BUFFER_UNLOAD	PMIx_Disconnect_nb, 8, 147
Defintion, 177	Defintion, 146
PMIx_Data_copy, 9	PMIX_DISPLAY_MAP, 132, 137, 226
Defintion, 182	Defintion, 65
PMIx_Data_copy_payload, 9	pmix_dmodex_response_fn_t, 204
Defintion, 183	Defintion, 81
PMIx_Data_pack, 9, 179	PMIX_DOUBLE, 51
Defintion, 178	PMIX_DSTPATH
PMIx_Data_print, 9	Defintion, 54
Defintion, 182	PMIX_EMBED_BARRIER, 90
PMIX_DATA_RANGE, 51	Defintion, 61
PMIx_Data_range_string, 9	PMIX_ERR_BAD_PARAM, 18
Defintion, 84	PMIX_ERR_COMM_FAILURE, 19
pmix_data_range_t, 28, 51, 84, 173, 236	PMIX_ERR_DATA_VALUE_NOT_FOUND,
Defintion, 28	18
PMIX_DATA_SCOPE, 101, 105	PMIX_ERR_DEBUGGER_RELEASE, 18
Defintion, 61	PMIX_ERR_EVENT_REGISTRATION, 19
PMIX_DATA_TYPE, 51	PMIX_ERR_HANDSHAKE_FAILED, 18
PMIX_DATA_TYPE_MAX, 51	PMIX_ERR_IN_ERRNO, 18
PMIx_Data_type_string, 9	PMIX_ERR_INIT, 18
Defintion, 85	PMIX_ERR_INVALID_ARG, 18
pmix_data_type_t, 31, 34, 40, 50, 85, 179,	PMIX_ERR_INVALID_ARGS, 19
180, 182, 183	PMIX_ERR_INVALID_CRED, 18
Defintion, 50	PMIX_ERR_INVALID_KEY, 18
PMIx_Data_unpack, 9	PMIX_ERR_INVALID_KEY_LENGTH, 18
Defintion, 180	PMIX_ERR_INVALID_KEYVALP, 19
PMIX_DEBUG_JOB	PMIX_ERR_INVALID_LENGTH, 19
Defintion, 68	PMIX_ERR_INVALID_NAMESPACE, 19
PMIX_DEBUG_STOP_IN_INIT	PMIX_ERR_INVALID_NUM_ARGS, 19

PMIX_ERR_INVALID_NUM_PARSED, 19	18
PMIX_ERR_INVALID_OPERATION, 20	PMIX_ERR_UNPACK_FAILURE, 18
PMIX_ERR_INVALID_SIZE, 19	PMIX_ERR_UNPACK_INADEQUATE_SPACE,
PMIX_ERR_INVALID_TERMINATION,	18
19	PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER
PMIX_ERR_INVALID_VAL, 18	19
PMIX_ERR_INVALID_VAL_LENGTH, 19	PMIX_ERR_UNREACH, 18
PMIX_ERR_JOB_TERMINATED, 19	PMIX_ERR_UPDATE_ENDPOINTS, 19
PMIX_ERR_LOST_CONNECTION_TO_CLIENT,	
19	PMIX_ERROR, 18
PMIX_ERR_LOST_CONNECTION_TO_SERVER,	
19	Defintion, 62
PMIX_ERR_LOST_PEER_CONNECTION,	PMIX_ERROR_GROUP_COMM
19	Defintion, 62
PMIX_ERR_NO_PERMISSIONS, 18	PMIX_ERROR_GROUP_GENERAL
PMIX_ERR_NODE_DOWN, 20	Defintion, 63
PMIX_ERR_NODE_OFFLINE, 20	PMIX_ERROR_GROUP_LOCAL
PMIX_ERR_NOMEM, 18	Defintion, 63
PMIX_ERR_NOT_FOUND, 19	PMIX_ERROR_GROUP_MIGRATE
PMIX_ERR_NOT_IMPLEMENTED, 19	Defintion, 62
PMIX_ERR_NOT_SUPPORTED, 19	PMIX_ERROR_GROUP_NODE
PMIX_ERR_OUT_OF_RESOURCE, 18	Defintion, 62
PMIX_ERR_PACK_FAILURE, 18	PMIX_ERROR_GROUP_RESOURCE
PMIX_ERR_PACK_MISMATCH, 18	Defintion, 62
PMIX_ERR_PROC_ABORTED, 18	PMIX_ERROR_GROUP_SPAWN
PMIX_ERR_PROC_ABORTING, 18	Defintion, 62
PMIX_ERR_PROC_CHECKPOINT, 18	PMIX_ERROR_HANDLER_ID
PMIX_ERR_PROC_ENTRY_NOT_FOUND,	Defintion, 63
18	PMIX_ERROR_NAME
PMIX_ERR_PROC_MIGRATE, 18	Defintion, 62
PMIX_ERR_PROC_REQUESTED_ABORT,	PMIx_Error_string, 9
18	Defintion, 84
PMIX_ERR_PROC_RESTART, 18	PMIX_EVENT_ACTION_COMPLETE, 20
PMIX_ERR_READY_FOR_HANDSHAKE,	PMIX_EVENT_ACTION_DEFERRED, 20
18	PMIX_EVENT_ACTION_TIMEOUT, 170
PMIX_ERR_RESOURCE_BUSY, 18	Defintion, 64
PMIX_ERR_SERVER_FAILED_REQUEST,	PMIX_EVENT_AFFECTED_PROC, 170,
18	173
PMIX_ERR_SERVER_NOT_AVAIL, 19	Defintion, 63
PMIX_ERR_SILENT, 18	PMIX_EVENT_AFFECTED_PROCS, 170,
PMIX_ERR_TIMEOUT, 18	174
PMIX_ERR_TYPE_MISMATCH, 18	Defintion, 63
PMIX ERR UNKNOWN DATA TYPE,	PMIX_EVENT_BASE, 89, 93, 97

Defintion, 52	Defintion, 64
PMIX_EVENT_CUSTOM_RANGE, 169,	PMIX_EVENT_TERMINATE_PROC, 170
173	Defintion, 64
Defintion, 63	PMIX_EVENT_TERMINATE_SESSION,
PMIX_EVENT_DO_NOT_CACHE	170
Defintion, 63	Defintion, 64
PMIX_EVENT_HDLR_AFTER, 169	PMIX_EVENT_WANT_TERMINATION
Defintion, 63	Defintion, 64
PMIX_EVENT_HDLR_APPEND, 169	pmix_evhdlr_reg_cbfunc_t, 76, 169
Defintion, 63	Defintion, 76
PMIX_EVENT_HDLR_BEFORE, 169	PMIX_EXISTS, 18
Defintion, 63	PMIX_EXTERNAL_ERR_BASE, 20
PMIX_EVENT_HDLR_FIRST, 169	PMIx_Fence, 3, 7, 8, 11, 97, 114, 116, 139,
Defintion, 63	142, 146, 204, 213, 215
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY,	Defintion, 112
169	PMIx_Fence_nb, 8, 74, 116, 213, 215
Defintion, 63	Defintion, 114
PMIX_EVENT_HDLR_LAST, 169	PMIx_Finalize, 8, 19, 25, 61, 89, 90, 139,
Defintion, 63	211, 212
PMIX_EVENT_HDLR_LAST_IN_CATEGORY,	Defintion, 90
169	PMIX_FLOAT, 51
Defintion, 63	PMIX_FWD_STDERR, 133, 137, 226, 241
PMIX_EVENT_HDLR_NAME, 169	Defintion, <b>65</b>
Defintion, 63	PMIX_FWD_STDIN, 133, 137, 226, 241
PMIX_EVENT_HDLR_PREPEND, 169	Defintion, <b>65</b>
Defintion, 63	PMIX_FWD_STDOUT, 133, 137, 226, 241
PMIX_EVENT_NO_ACTION_TAKEN, 20	Defintion, <b>65</b>
PMIX_EVENT_NO_TERMINATION	PMIX_GDS_ACTION_COMPLETE, 20
Defintion, 64	PMIX_GDS_MODULE, 89, 93, 97
PMIX_EVENT_NON_DEFAULT, 173	Defintion, 55
Defintion, 63	PMIx_generate_ppn, 8
pmix_event_notification_cbfunc_fn_t, 77, 79	Defintion, 186
Defintion, 77	PMIx_generate_regex, 8, 192
PMIX_EVENT_PARTIAL_ACTION_TAKEN,	Defintion, 185
20	PMIx_Get, 3, 8, 10, 29, 52–55, 60–62,
PMIX_EVENT_RETURN_OBJECT, 170	64–71, 89, 101, 103, 105–109,
Defintion, 63	111, 132–134, 136–138, 153, 154,
PMIX_EVENT_SILENT_TERMINATION,	188, 190, 225–227
170	Defintion, <b>100</b>
Defintion, 63	PMIx_Get_nb, 8, 75
PMIX_EVENT_TERMINATE_JOB, 170	Defintion, 103
Defintion, 64	PMIx_Get_version, 9, 14
PMIX_EVENT_TERMINATE_NODE, 170	Defintion, 87
1 1	2011111011, 07

PMIX_GLOBAL, 28	Defintion, 33
PMIX_GLOBAL_RANK, 191	PMIX_INFO_DIRECTIVES, 51
Defintion, 56	PMIx_Info_directives_string, 9
PMIX_GRPID, 118, 120, 122, 124, 126,	Defintion, 85
128, 152, 156, 158, 161, 164,	pmix_info_directives_t, 36, 85
218–223, 225, 233, 239, 241, 243,	Defintion, 36
245, 247, 250	PMIX_INFO_FREE
Defintion, 54	Defintion, 34
PMIx_Heartbeat, 9	PMIX_INFO_IS_REQUIRED, 36
Defintion, 163	Defintion, 37
PMIX_HOST, 132, 136, 225	PMIX_INFO_LOAD
Defintion, 64	Defintion, 34
PMIX_HOSTFILE, 132, 136, 225	PMIX_INFO_REQD, 36
Defintion, 64	PMIX_INFO_REQUIRED, 36
PMIX_HOSTNAME, 58, 102, 105, 111,	Defintion, 37
151, 191	pmix_info_t, 3, 9, 11, 28, 32–37, 58, 68–70,
Defintion, 57	76–79, 88, 90, 91, 95, 97, 119,
PMIX_HWLOC_SHMEM_ADDR	123, 153, 155, 156, 159, 160, 163,
Defintion, 60	165, 173, 192, 195–197, 236,
PMIX_HWLOC_SHMEM_FILE	241–243, 245, 246, 248, 249
Defintion, 60	Defintion, 32
PMIX_HWLOC_SHMEM_SIZE	PMIX_INFO_TRUE
Defintion, 60	Defintion, 35
PMIX_HWLOC_XML_V1, 190	PMIX_INFO_XFER, 192
Defintion, 60	Defintion, 35
PMIX_HWLOC_XML_V2, 190	PMIx_Init, 9, 65, 68, 87, 89, 90, 133, 137,
Defintion, 60	210, 226
PMIX_IMMEDIATE, 101, 104	Defintion, 87
Defintion, 60	PMIx_Initialized, 8
PMIX_INDEX_ARGV, 133, 138, 227	Defintion, 86
Defintion, 66	PMIX_INT, 50
PMIX_INFO, 51	PMIX_INT16, 50
PMIX_INFO_ARRAY, 51	PMIX_INT32, 50
pmix_info_array, 32, 33	PMIX_INT64, 50
Defintion, 32	PMIX_INT8, 50
pmix_info_cbfunc_t, 72, 75, 150, 155, 158,	PMIX_INTERNAL, 28
160, 161, 163, 239, 244, 246, 249	PMIX_JCTRL_CHECKPOINT, 19
Defintion, 75	PMIX_JCTRL_CHECKPOINT_COMPLETE
PMIX_INFO_CONSTRUCT	19
Defintion, 33	PMIX_JCTRL_PREEMPT_ALERT, 19
PMIX_INFO_CREATE	PMIX_JOB_CONTINUOUS, 134, 138, 227
Defintion, 34	Defintion, 66
PMIX_INFO_DESTRUCT	PMIx_Job_control_nb, 9, 70, 148, 157, 192

Defintion, 157	Defintion, 58
PMIX_JOB_CTRL_CANCEL, 159, 248	PMIX_JOB_NUM_APPS, 109, 190, 195
Defintion, 70	Defintion, 59
PMIX_JOB_CTRL_CHECKPOINT, 159,	PMIX_JOB_RECOVERABLE, 134, 138,
248	227
Defintion, 70	Defintion, 66
PMIX_JOB_CTRL_CHECKPOINT_EVENT,	PMIX_JOB_SIZE, 10, 103, 106, 109, 188,
159, 248	195, 196
Defintion, 70	Defintion, 59
PMIX_JOB_CTRL_CHECKPOINT_METHOD,	PMIX_JOB_TERM_STATUS
159, 248	Defintion, 61
Defintion, 70	PMIX_JOBID, 58, 102, 105, 109, 151, 188,
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL,	195
159, 248	Defintion, 56
Defintion, 70	pmix_key_t, 20, 21, 99, 101
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT,	Defintion, 20
159, 248	PMIX_KVAL, 51
Defintion, 70	PMIX_LOCAL, 27
PMIX_JOB_CTRL_ID, 158, 247	PMIX_LOCAL_CPUSETS, 189, 198
Defintion, 70	Defintion, 57
PMIX_JOB_CTRL_KILL, 159, 248	PMIX_LOCAL_PEERS, 188, 198
Defintion, 70	Defintion, 57
PMIX_JOB_CTRL_PAUSE, 159, 247	PMIX_LOCAL_PROCS, 191
Defintion, 70	Defintion, 57
PMIX_JOB_CTRL_PREEMPTIBLE, 159,	PMIX_LOCAL_RANK, 189
248	Defintion, 56
Defintion, 70	PMIX_LOCAL_SIZE, 188
PMIX_JOB_CTRL_PROVISION, 159, 248	Defintion, 59
Defintion, 70	PMIX_LOCAL_TOPO
PMIX_JOB_CTRL_PROVISION_IMAGE,	Defintion, 59
159, 248	PMIX_LOCALITY
Defintion, 70	Defintion, 57
PMIX_JOB_CTRL_RESTART, 159, 248	PMIX_LOCALITY_STRING
Defintion, 70	Defintion, 60
PMIX_JOB_CTRL_RESUME, 159, 247	PMIX_LOCALLDR, 191
Defintion, 70	Defintion, 56
PMIX_JOB_CTRL_SIGNAL, 159, 248	PMIX_LOG_EMAIL, 165, 243
Defintion, 70	Defintion, 68
PMIX_JOB_CTRL_TERMINATE, 159, 248	PMIX_LOG_EMAIL_ADDR, 165, 243
Defintion, 70	Defintion, 68
PMIX_JOB_INFO, 101, 105, 109, 151	PMIX_LOG_EMAIL_MSG, 165, 243
Defintion, 57	Defintion, 68
PMIX_JOB_INFO_ARRAY, 195	PMIX_LOG_EMAIL_SUBJECT, 165, 243

Defintion, 68	Defintion, 45
PMIX_LOG_MSG, 165, 243	pmix_modex_data_t, 44-46
Defintion, 68	Defintion, 44
PMIx_Log_nb, 9, 67, 165	PMIX_MODEX_DESTRUCT
Defintion, 163	Defintion, 45
PMIX_LOG_STDERR, 165, 243	PMIX_MODEX_FREE
Defintion, 67	Defintion, 46
PMIX_LOG_STDOUT, 165, 243	PMIX_MONITOR_APP_CONTROL, 162,
Defintion, 67	250
PMIX_LOG_SYSLOG, 165, 243	Defintion, 71
Defintion, 68	PMIX_MONITOR_CANCEL, 161, 250
PMIx_Lookup, 8, 38, 117, 123, 125	Defintion, 71
Defintion, 121	PMIX_MONITOR_FILE, 162, 251
pmix_lookup_cbfunc_t, 74, 220	Defintion, 71
Defintion, 74	PMIX_MONITOR_FILE_ACCESS, 162,
PMIx_Lookup_nb, 74, 75	251
Defintion, 123	Defintion, 71
PMIX_MAP_BLOB	PMIX_MONITOR_FILE_ALERT, 19
Defintion, 62	PMIX_MONITOR_FILE_CHECK_TIME,
PMIX_MAPBY, 132, 137, 190, 226	162, 251
Defintion, 65	Defintion, 71
PMIX_MAPPER, 64, 132, 136, 225	PMIX_MONITOR_FILE_DROPS, 162, 251
Defintion, 65	Defintion, 71
PMIX_MAX_KEYLEN, 16	PMIX_MONITOR_FILE_MODIFY, 162,
PMIX_MAX_NSLEN, 16	251
PMIX_MAX_PROCS, 188	Defintion, 71
Defintion, 59	PMIX_MONITOR_FILE_SIZE, 162, 251
PMIX_MAX_RESTARTS, 134, 138, 227	Defintion, 71
Defintion, 66	PMIX_MONITOR_HEARTBEAT, 162, 250
PMIX_MERGE_STDERR_STDOUT, 133,	Defintion, 71
137, 226	PMIX_MONITOR_HEARTBEAT_ALERT,
Defintion, 66	19
PMIX_MODEL_DECLARED, 20	PMIX_MONITOR_HEARTBEAT_DROPS,
PMIX_MODEL_LIBRARY_NAME	162, 250
Defintion, 54	Defintion, 71
PMIX_MODEL_LIBRARY_VERSION	PMIX_MONITOR_HEARTBEAT_TIME,
Defintion, 54	162, 250
PMIX_MODEX, 51	Defintion, 71
pmix_modex_cbfunc_t, 72, 214, 216	PMIX_MONITOR_ID, 161, 250
Defintion, 72	Defintion, 71
PMIX_MODEX_CONSTRUCT	PMIX_NET_TOPO
Defintion, 45	Defintion, 59
PMIX_MODEX_CREATE	PMIX_NO_OVERSUBSCRIBE, 133, 138,

227	142, 146, 164, 172, 173, 187,
Defintion, 66	200–202, 207, 210, 211, 213, 218
PMIX_NO_PROCS_ON_HEAD, 133, 138,	222, 228, 231, 233, 235, 236, 242
227	247, 249
Defintion, 66	Defintion, 74
PMIX_NODE_INFO, 102, 105, 111, 151	PMIX_OPTIONAL, 101, 104
Defintion, 58	Defintion, 61
PMIX_NODE_INFO_ARRAY, 197	PMIX_OUTPUT_TO_FILE, 133, 137, 226
Defintion, 58	Defintion, 66
PMIX_NODE_LIST	PMIX_PARENT_ID, 131, 135, 136, 225
Defintion, <b>57</b> , <b>59</b>	Defintion, 57
PMIX_NODE_MAP, 57, 188, 189, 195, 196	PMIX_PDATA, 51
Defintion, 61	PMIX_PDATA_CONSTRUCT
PMIX_NODE_RANK, 189	Defintion, 38
Defintion, 56	PMIX_PDATA_CREATE
PMIX_NODE_SIZE, 111, 191	Defintion, 39
Defintion, 59	PMIX_PDATA_DESTRUCT
PMIX_NODEID, 58, 102, 105, 111, 151,	Defintion, 38
189	PMIX_PDATA_FREE
Defintion, <b>57</b>	Defintion, 39
PMIX_NON_PMI, 133, 137, 226	PMIX_PDATA_LOAD
Defintion, 65	Defintion, 39
pmix_notification_fn_t, 78, 169	pmix_pdata_t, 38-40, 74, 75, 123
Defintion, 78	Defintion, 38
PMIX_NOTIFY_ALLOC_COMPLETE, 19	PMIX_PDATA_XFER
PMIX_NOTIFY_COMPLETION	Defintion, 40
Defintion, 61	PMIX_PERSIST, 51
PMIx_Notify_event, 9	PMIX_PERSIST_APP, 28
Defintion, 172	PMIX_PERSIST_FIRST_READ, 28
PMIX_NPROC_OFFSET, 190	PMIX_PERSIST_INDEF, 28
Defintion, 56	PMIX_PERSIST_PROC, 28
PMIX_NSDIR, 56	PMIX_PERSIST_SESSION, 28
Defintion, 56	PMIX_PERSISTENCE, 118, 121, 219
PMIX_NSPACE, 58, 102, 105, 109, 151, 196	Defintion, 61
Defintion, 56	PMIx_Persistence_string, 9
pmix_nspace_t, 21, 24, 73	Defintion, 84
Defintion, 21	pmix_persistence_t, 28, 51, 84
PMIX_NUM_NODES, 99, 103, 106, 107,	Defintion, 28
109, 110, 195, 196	PMIX_PERSONALITY, 132, 136, 225
Defintion, 59	Defintion, 64
PMIX_NUM_SLOTS, 59, 111, 188	PMIX_PID, 50
Defintion, 59	PMIX_POINTER, 51
pmix_op_cbfunc_t, 74, 77, 81, 120, 127,	PMIX_PPR, 132, 137, 226

Defintion, 65	25
PMIX_PREFIX, 132, 136, 225	PMIX_PROC_STATE_CANNOT_RESTART,
Defintion, 64	25
PMIX_PRELOAD_BIN, 132, 136, 225	PMIX_PROC_STATE_COMM_FAILED,
Defintion, 65	25
PMIX_PRELOAD_FILES, 132, 136, 225	PMIX_PROC_STATE_CONNECTED, 25
Defintion, 65	PMIX_PROC_STATE_ERROR, 25
PMIX_PROC, 51	PMIX_PROC_STATE_FAILED_TO_LAUNCH
PMIX_PROC_BLOB	25
Defintion, 62	PMIX_PROC_STATE_FAILED_TO_START,
PMIX_PROC_CONSTRUCT, 23	25
Defintion, 23, 46, 49	PMIX_PROC_STATE_KILLED_BY_CMD,
PMIX_PROC_CREATE	25
Defintion, 23	PMIX_PROC_STATE_LAUNCH_UNDERWAY
PMIX_PROC_DATA, 196	25
Defintion, 61	PMIX_PROC_STATE_MIGRATING, 25
PMIX_PROC_DESTRUCT	PMIX_PROC_STATE_PREPPED, 25
Defintion, 23	PMIX_PROC_STATE_RESTART, 25
PMIX_PROC_FREE, 149	PMIX_PROC_STATE_RUNNING, 25
Defintion, 24	PMIX_PROC_STATE_STATUS
PMIX_PROC_INFO, 51	Defintion, 61
PMIX_PROC_INFO_CONSTRUCT	PMIx_Proc_state_string, 9
Defintion, 26	Defintion, 84
PMIX_PROC_INFO_CREATE	pmix_proc_state_t, 24, 51, 84
Defintion, 27	Defintion, 24
PMIX_PROC_INFO_DESTRUCT	PMIX_PROC_STATE_TERM_NON_ZERO,
Defintion, 26	25
PMIX_PROC_INFO_FREE	PMIX_PROC_STATE_TERM_WO_SYNC,
Defintion, 27	25
pmix_proc_info_t, 25–27, 51, 67, 152, 240	PMIX_PROC_STATE_TERMINATE, 25
Defintion, 25	PMIX_PROC_STATE_TERMINATED, 25
PMIX_PROC_LOAD	PMIX_PROC_STATE_UNDEF, 25
Defintion, 24	PMIX_PROC_STATE_UNTERMINATED,
PMIX_PROC_MAP, 188, 195, 196	25
Defintion, 61	pmix_proc_t, 22-24, 39, 40, 51, 57, 63, 79,
PMIX_PROC_PID	83, 89, 91, 93, 103, 113–115, 129,
Defintion, 56	170, 173, 174, 179, 180, 191,
PMIX_PROC_RANK, 51	201–204, 210–212, 214, 216, 218,
PMIX_PROC_STATE, 51	220, 222, 224, 228, 231, 236, 239,
PMIX_PROC_STATE_ABORTED, 25	242, 244, 247, 249
PMIX_PROC_STATE_ABORTED_BY_SIG,	Defintion, 22
25	PMIX_PROC_TERMINATED, 19
PMIX_PROC_STATE_CALLED_ABORT,	PMIX_PROC_URI

Defintion, 57	PMIX_QUERY_MEMORY_USAGE, 152,
PMIX_PROCDIR	240
Defintion, 56	Defintion, 67
PMIx_Process_monitor_nb, 9, 71, 148, 163	PMIX_QUERY_NAMESPACES, 152, 239
Defintion, 160	Defintion, 67
PMIX_PROCID, 191	PMIX_QUERY_PARTIAL_SUCCESS, 19
Defintion, 56	PMIX_QUERY_PROC_TABLE, 152, 240
PMIX_PROGRAMMING_MODEL	Defintion, 67
Defintion, 54	PMIX_QUERY_QUEUE_LIST, 152, 239
PMIx_Publish, 8, 28, 61, 118, 119, 121, 219	Defintion, 67
Defintion, 117	PMIX_QUERY_QUEUE_STATUS, 152,
PMIx_Publish_nb, 8, 121	239
Defintion, 119	Defintion, 67
PMIx_Put, 8, 27–29, 82, 100, 103, 106, 112,	PMIX_QUERY_REFRESH_CACHE, 151,
114, 139, 154, 204, 217	153, 154
Defintion, 99	Defintion, 66
PMIX_QUERY, 51	PMIX_QUERY_REPORT_AVG, 152, 240
PMIX_QUERY_ALLOC_STATUS, 152,	Defintion, 67
240	PMIX_QUERY_REPORT_MINMAX, 152,
Defintion, 67	240
PMIX_QUERY_AUTHORIZATIONS	Defintion, 67
Defintion, 67	PMIX_QUERY_SPAWN_SUPPORT, 152,
PMIX_QUERY_CONSTRUCT	240
Defintion, 43	Defintion, 67
PMIX_QUERY_CREATE	pmix_query_t, 43, 44, 154, 239, 240
Defintion, 44	Defintion, 43
PMIX_QUERY_DEBUG_SUPPORT, 152,	PMIX_RANGE, 118, 121, 122, 124, 126,
240	128, 170, 218, 221, 223, 237
Defintion, 67	Defintion, 61
PMIX_QUERY_DESTRUCT	PMIX_RANGE_CUSTOM, 28
Defintion, 43	PMIX_RANGE_GLOBAL, 28
PMIX_QUERY_FREE	PMIX_RANGE_LOCAL, 28
Defintion, 44	PMIX_RANGE_NAMESPACE, 28
PMIx_Query_info_nb, 9, 10, 43, 66, 111,	PMIX_RANGE_PROC_LOCAL, 28
139, 148, 153, 154	PMIX_RANGE_RM, 28
Defintion, 150	PMIX_RANGE_SESSION, 28
PMIX_QUERY_JOB_STATUS, 152, 239	PMIX_RANGE_UNDEF, 28
Defintion, 67	PMIX_RANK, 189
PMIX_QUERY_LOCAL_ONLY, 240	Defintion, 56
Defintion, 67	PMIX_RANK_LOCAL_NODE, 22
PMIX_QUERY_LOCAL_PROC_TABLE,	pmix_rank_t, 22, 24, 51
152, 240	Defintion, 22
Defintion, 67	PMIX_RANK_UNDEF, 22

PMIX_RANK_WILDCARD, 22	Defintion, 228
PMIX_RANKBY, 132, 137, 190, 226	PMIx_server_deregister_client, 8
Defintion, 65	Defintion, 202
PMIx_Register_event_handler, 9, 77, 148	pmix_server_deregister_events_fn_t
Defintion, 168	Defintion, 234
PMIX_REGISTER_NODATA, 188	PMIx_server_deregister_nspace, 8, 202
Defintion, 53, 61	Defintion, 199
pmix_release_cbfunc_t, 72	pmix_server_disconnect_fn_t, 232
Defintion, 72	Defintion, 230
PMIX_REMOTE, 27	pmix_server_dmodex_req_fn_t, 72
PMIX_REPORT_BINDINGS, 134, 138, 227	Defintion, 216
Defintion, 66	PMIx_server_dmodex_request, 9, 81, 82,
PMIX_REQUESTOR_IS_CLIENT, 131,	204
135	Defintion, 203
Defintion, 54	PMIX_SERVER_ENABLE_MONITORING
PMIX_REQUESTOR_IS_TOOL, 131, 136	Defintion, 53
Defintion, 54	pmix_server_fencenb_fn_t, 72, 215
PMIx_Resolve_nodes, 8	Defintion, 213
Defintion, 149	PMIx_server_finalize, 8
PMIx_Resolve_peers, 8	Defintion, 97
Defintion, 149	PMIX_SERVER_HOSTNAME
PMIX_RM_NAME	Defintion, 53
Defintion, 68	PMIx_server_init, 8, 87, 208
PMIX_RM_VERSION	Defintion, 95
Defintion, 69	pmix_server_job_control_fn_t
PMIX_SCOPE, 51	Defintion, 246
PMIx_Scope_string, 9	pmix_server_listener_fn_t
Defintion, 84	Defintion, 237
pmix_scope_t, 27, 51, 84, 100	pmix_server_log_fn_t
Defintion, 27	Defintion, 242
PMIX_SCOPE_UNDEF, 27	pmix_server_lookup_fn_t
PMIX_SEND_HEARTBEAT	Defintion, 219
Defintion, 71	pmix_server_module_t, 95, 97, 208
pmix_server_abort_fn_t	Defintion, 208
Defintion, 212	pmix_server_monitor_fn_t
pmix_server_alloc_fn_t	Defintion, 249
Defintion, 244	pmix_server_notify_event_fn_t, 80
pmix_server_client_connected_fn_t, 74, 201,	Defintion, 236
202, 210	PMIX_SERVER_NSPACE, 95, 190
Defintion, <b>209</b>	Defintion, 53
pmix_server_client_finalized_fn_t, 211, 212	PMIX_SERVER_PIDINFO, 92, 93
Defintion, <b>211</b>	Defintion, 53
pmix_server_connect_fn_t, 230, 232	pmix_server_publish_fn_t

Defintion, 217	195
pmix_server_query_fn_t	Defintion, 58
Defintion, 238	PMIX_SET_ENVAR
PMIX_SERVER_RANK, 95, 190	Defintion, 69
Defintion, 53	PMIX_SET_SESSION_CWD, 132, 136, 225
PMIx_server_register_client, 8, 202, 210,	Defintion, 65
211	pmix_setup_application_cbfunc_t, 205
Defintion, 200	Defintion, 80
pmix_server_register_events_fn_t	PMIX_SINGLE_LISTENER, 88
Defintion, 232	Defintion, 54
PMIx_server_register_nspace, 8, 10, 14, 58,	PMIX_SIZE, 50
74, 192, 195	PMIX_SOCKET_MODE, 88, 92, 96
Defintion, 187	Defintion, 54
PMIX_SERVER_REMOTE_CONNECTIONS,	PMIx_Spawn, 8, 41, 55, 64, 68, 69, 130, 131,
97	135, 136, 138, 192, 203, 223, 228
Defintion, 52	Defintion, 130
PMIx_server_setup_application, 9, 80, 81,	pmix_spawn_cbfunc_t, 73, 135, 224
207	Defintion, 73
Defintion, 205	PMIx_Spawn_nb, 8, 41, 73
PMIx_server_setup_fork, 9	Defintion, 135
Defintion, 203	PMIX_SPAWNED, 131, 135, 136, 225
PMIx_server_setup_local_support, 9	Defintion, 55
Defintion, 206	PMIX_STATUS, 51
pmix_server_spawn_fn_t, 73	pmix_status_t, 17, 32, 51, 76, 77, 79, 81–84,
Defintion, 223	169, 173, 233, 235, 236
PMIX_SERVER_SYSTEM_SUPPORT, 96	Defintion, 17
Defintion, 52	PMIX_STDIN_TGT, 133, 137, 226
PMIX_SERVER_TMPDIR, 95	Defintion, 65
Defintion, 52	PMIx_Store_internal, 9
pmix_server_tool_connection_fn_t	Defintion, 106
Defintion, 241	PMIX_STRING, 50
PMIX_SERVER_TOOL_SUPPORT, 95	PMIX_SUCCESS, 18
Defintion, 52	PMIX_SYSTEM_TMPDIR, 95
pmix_server_unpublish_fn_t	Defintion, 52
Defintion, 221	PMIX_TAG_OUTPUT, 133, 137, 226
PMIX_SERVER_URI, 91, 93	Defintion, 66
Defintion, 53	PMIX_TCP_DISABLE_IPV4, 89, 93, 96
PMIX_SESSION_ID, 57, 58, 101, 105, 108,	Defintion, 55
151, 190, 195	PMIX_TCP_DISABLE_IPV6, 89, 93, 96
Defintion, 57	Defintion, 55
PMIX_SESSION_INFO, 101, 105, 107, 151	PMIX_TCP_IF_EXCLUDE, 88, 92, 96
Defintion, 57	Defintion, 55
PMIX_SESSION_INFO_ARRAY, 59, 188,	PMIX_TCP_IF_INCLUDE, 88, 92, 96

Defintion, 55	Defintion, 60
PMIX_TCP_IPV4_PORT, 89, 92, 96	PMIX_UINT, 50
Defintion, 55	PMIX_UINT16, 51
PMIX_TCP_IPV6_PORT, 89, 92, 96	PMIX_UINT32, 51
Defintion, 55	PMIX_UINT64, 51
PMIX_TCP_REPORT_URI, 88, 92, 96	PMIX_UINT8, 50
Defintion, 54	PMIX_UNDEF, 50
PMIX_TCP_URI, 92, 93	PMIX_UNIV_SIZE, 10, 103, 106, 107, 188,
Defintion, 54	195
PMIX_TDIR_RMCLEAN	Defintion, 58
Defintion, <b>56</b>	PMIx_Unpublish, 8, 127, 128
PMIX_THREADING_MODEL	Defintion, 125
Defintion, 54	PMIx_Unpublish_nb, 8
PMIX_TIME, 51	Defintion, 127
PMIX_TIME_REMAINING, 148, 152, 240	PMIX_UNSET_ENVAR
Defintion, 67	Defintion, 69
PMIX_TIMEOUT, 3, 11, 102, 103, 105, 106,	PMIX_USERID, 118, 120, 122, 124, 126,
113, 114, 116, 118, 121–126, 128,	128, 152, 156, 158, 161, 164,
140, 141, 143, 145, 147, 215, 217,	218–224, 233, 239, 241, 243, 245,
219, 221, 223, 227, 229, 232	247, 250
Defintion, 60	Defintion, 53
PMIX_TIMESTAMP_OUTPUT, 133, 137,	PMIX_USOCK_DISABLE, 88, 96
226	Defintion, 54
Defintion, 66	PMIX_VALUE, 51
PMIX_TIMEVAL, 51	pmix_value_cbfunc_t, 75
PMIX_TMPDIR, 56	Defintion, 75
Defintion, 56	PMIX_VALUE_CONSTRUCT
pmix_tool_connection_cbfunc_t, 241	Defintion, 30
Defintion, 83	PMIX_VALUE_CREATE
PMIX_TOOL_DO_NOT_CONNECT, 91,	Defintion, 30
93	PMIX_VALUE_DESTRUCT
Defintion, 53	Defintion, 30
PMIx_tool_finalize, 9	PMIX_VALUE_FREE
Defintion, 94	Defintion, 31
PMIx_tool_init, 9, 53, 87, 94	PMIX_VALUE_LOAD
Defintion, 91	Defintion, 31
PMIX_TOOL_NSPACE, 91	pmix_value_t, 29–32, 51, 75, 99, 100
Defintion, 53	Defintion, 29
PMIX_TOOL_RANK, 91	PMIX_VALUE_XFER
Defintion, 53	Defintion, 32
PMIX_TOPOLOGY	PMIX_VERSION_INFO
Defintion, 60	Defintion, <b>54</b>
PMIX_TOPOLOGY_SIGNATURE	PMIX_WAIT, 122–124, 221

Defintion, 60	session, 10, 57, 58, 99, 107, 154, 192
PMIX_WDIR, 131, 136, 225	Defintion, 12
Defintion, 64	slot
	Defintion, 12
rank, 109, 196 Defintion, 12	slots Defintion, 12
resource manager	workflow
Defintion, 13	Defintion, 12