Data Security Policy Engine Interface API Guide

Applies to:	Data Security, v7.6.x - 7.8.x TRITON AP-DATA, v8.0.x - 8.3.x

Forcepoint™ TRITON® AP-DATA monitors and protects data by using a series of *agents* that are deployed according to your organization's needs. One such agent, the Integration Agent, allows third-party products to send data to Data Security for analysis. It is embedded in third-party installers and communicates with Data Security via a C-based API.

The API can be used to configure analysis operations on a transaction-by-transaction basis on the following variables:

- Channel/Protocol - Upon installation the third-party product can declare its ability to intercept various protocols, and assign each transaction to a protocol.
- Blocking/Monitoring mode - each transaction can work in a different mode.
- Timeout - can be different per transaction.

This API helps you to integrate with Data Security and adds data loss protection capabilities to your product.

The Data Security Policy Engine is responsible for analyzing the data flowing through your enterprise, comparing it to the rules in your policies, and governing remediation action, if necessary.

This guide provides details about the functions supported by the Data Security Policy Engine Interface API and includes the following topics:

For details regarding how to configure the Policy Engine, refer to *TRITON - Data Security Help*.

Analyzing Functions

The following functions enable you to submit transactions for data loss analysis. Some are optional, as indicated in the table below:

Function	Description
PEIErrors PEI_Transaction_Analyze(PEIHandle transaction, AnalysisCallback callback, void *userContext);	Perform asynchronous analysis on a policy engine transaction. Transaction must be a valid transaction (see below for details). A callback will be called by another thread when the analysis completes. userContext will be passed to the callback. If the function returns PEI_ERROR_NO_ERROR, it guarantees that the callback will be called.
PEIErrors PEI_Transaction_AnalyzeSynch(PEIHandle transaction, struct AnalysisResult **result);	Perform synchronous analysis on a transaction. Transaction must be a valid transaction (see below for details). If the function returns PEI_ERROR_NO_ERROR, the result will point to a valid AnalysisResult structure, which should be destroyed.
struct AnalysisResult *PEI_Transaction_AnalyzeSynch_Wrap(PEI Handle transaction);	A wrapper that avoids the AnalysisResult **result construct. It is more convenient for python calls.
PEIErrors PEI_Transaction_CancelTransaction(PEIHan dle transaction);	Cancel an in-progress analysis. This operation will free a blocking analysis request if any.
PEIErrors PEI_Transaction_SetGlobalActionTaken(PEI_UINT64 transactionID, const PEIActionType actionTaken, PEI_UINT32 descriptionID, const char *description);	Report the actual action taken (Optional).

Transaction Functions

The following lists functions to build and set a transaction object. A transaction object holds the entire information that is available for analysis – such as the source, destination, and content. Mandatory and optional functions are listed below.

Function	Description
PEIHandle PEI_Handle_GenerateTransaction();	This is a constructor.
PEIErrors PEI_Transaction_GetTransactionID(PEIHan dle transaction, PEI_UINT64 *id);	Get a Unique ID of the transaction. Can be used for logging purposes. This ID travels with the transaction all the way through to the DSS-Manager incident management system.

Function	Description
PEIErrors PEI_Transaction_AddContent(PEIHandle transaction, PEIHandle content);	Add a content (data) - each transaction can have multiple contents. (Mandatory).
PEIErrors PEI_Transaction_SetContext(PEIHandle transaction, PEIHandle context);	Set the context (sender & recipients) of a transaction. (Mandatory).
PEIErrors PEI_Transaction_SetTitle(PEIHandle transaction, const char *title);	A descriptive name for the transaction in UTF-8. (Mandatory).
PEIErrors PEI_Transaction_SetChannelID(PEIHandle transaction, PEI_UINT32 channelID); PEIErrors PEI_Transaction_SetServiceID(PEIHandle transaction, PEI_UINT32 serviceID);	Set the Channel-ID or Service-ID denoting the specific protocol and/or port on which the transaction was intercepted. These IDs are determined during the installation/registration with the "DSS-Manager". (Mandatory).
PEIErrors PEI_Transaction_SetChannelIsSecured(PEIHandle transaction);	This is a property of the channel. For example, the channel HTTPS should be reported as HTTP + IsSecured property set as 'true'. The default value of this property is 'false' and calling this function will set it as true.
PEIErrors PEI_Transaction_SetBypassBlock(PEIHandle transaction);	By default, the PEI assumes that blocking is possible, and that a PEI_BLOCK action will be honored. Call this function if blocking capability will not be used. **Note:** Calling this function is equivalent to calling the function PEI_Transaction_SetBypassBlock(transaction, PEI_FALSE, PEI_FALSE, PEI_FALSE);
PEIErrors PEI_Transaction_SetBlockMode(PEIHandle transaction, PEIBool outbound, PEIBool inbound, PEIBool internal);	Support blocking according the orientation of the transaction's destinations. Call this function to disable blocking for some (or all) of the possible destination (outbound/inbound/internal).
PEIErrors PEI_Transaction_SetDetectionTime(PEIHandle transaction, PEI_UINT64 time, int timeZone);	The time of the transaction. The default value is the time analysis started. (Optional).
PEIErrors PEI_Transaction_SetPriority(PEIHandle transaction, PEI_UINT32 priority);	Base priority for the transaction. Defaults to a channel-based priority defined in the configuration. (Optional).
PEIErrors PEI_Transaction_SetTimeout(PEIHandle transaction, PEI_UINT32 ms);	Number of milliseconds that can be used for analysis. The Policy Engine will prioritize and plan its execution in order to give the best response within the given time. Defaults to unlimited. (Optional).
PEIErrors PEI_Transaction_SetLevelOfRequiredDetails (PEIHandle transaction, enum LevelOfRequiredDetails levelOfDetails);	What details should be returned about found breaches. Defaults to PEI_NONE. (Optional).

Function	Description
PEIErrors PEI_Transaction_SetSubID(PEIHandle transaction, PEI_UINT64 subID);	The transaction ID of a previous transaction that this transaction relates to. This has special meaning in various protocols. Defaults to no SubID. (Optional).
PEIErrors PEI_Transaction_AddProperty(PEIHandle transaction, const char *propertyName, const char *propertyValue);	Arbitrary name/value properties of the transaction. (Optional).
PEIErrors PEI_Transaction_MarkTransaction(PEIHandle transaction);	Mark the transaction (for debugging purposes of the PolicyEngine. (Optional).

Content Functions

Listed below are functions that help you build and set a content object. A content object holds the buffer or file which the caller wants to scan for sensitive information.

Function	Description
PEIHandle PEI_Handle_GenerateContentContainer();	A constructor.
PEIErrors PEI_Content_CreateBuffer(PEIHandle content, PEI_UINT32 bufferSize, const char *description);	Set the data of the content, either through a file or through a RAM buffer. (Mandatory)
PEIErrors PEI_Content_SetBuffer(PEIHandle content, const char *buffer, PEI_UINT32 bufferSize, PEI_UINT32 offset);	In RAM mode, create a buffer once, and use SetBuffer as many times as you like to fill the buffer either fully or in chunked operation.
PEIErrors PEI_Content_DownsizeBuffer(PEIHandle content, PEI_UINT32 newBufferSize);	Downsize the buffer to fit the actual data. Using a too large buffer may harm the analysis process. (Optional).
PEIErrors PEI_Content_SetFile(PEIHandle content, const char *fileName);	In file mode, point to a file containing the buffer.
PEIErrors PEI_Content_SetFileAndDescription(PEIHandle content, const char *fileName, const char* description);	In file mode, point to a file containing the buffer.
PEIErrors PEI_Content_SetPartType(PEIHandle content, enum ContentPartType type);	Specify what data the content represents. (Mandatory)
PEIErrors PEI_Content_SetEncoding(PEIHandle content, const char *encoding);	Specify the encoding of the buffer. (Optional).
PEIErrors PEI_Content_MarkAsSliced(PEIHandle content, PEI_UINT64 originalSize);	Specify that the file was sliced, along with its original size. (Optional).

Context Functions

Context functions are functions to build and set the source and destination of a transaction. They are listed below:

Function	Description
PEIHandle PEI_Handle_GenerateContext();	A constructor.
PEIErrors PEI_Context_AddSourceProperty(PEIHandle context, enum ContextAddressType type, const char *value);	Set various attributes of the transaction source. Multiple calls can be used to specify several attributes of the source. (Mandatory).
PEIHandle PEI_Context_AddDestination(PEIHandle context, enum ContextAddressType type, const char *value); PEIHandle PEI_Context_AddOrientedDestination(PEIHandle context, enum ContextAddressType type, const char *value, PEIOrientation orient);	Set various attributes of the transaction destinations. Multiple calls can be used to specify several attributes of the destinations. (Mandatory for data in motion).
PEIErrors PEI_Context_AddDestinationProperty(PEIHandle context, PEIHandle dest, enum ContextAddressType type, const char *value);	Add a destination attribute to the destination pointed to by 'dest'. (Optional).
PEIErrors PEI_Context_AddLocationProperty(PEIHandle context, enum ContextAddressType type, const char *value);	Set various attributes of the transaction location. Multiple calls can be used to specify several attributes of the location. (Mandatory for data at rest).

Memory Management

Memory Management provides functions to set and control the memory usage of the policy engine interface.

Function	Description
void PEI_Handle_DestroyHandle(PEIHandle handle);	Destroy a transaction, content or context. Note that reference-counting is used, so handles to contents and contexts can (and should) be destroyed immediately after they are added to transactions.
PEIErrors PEI_Transaction_FreeAnalysisResults(struct AnalysisResult *result);	Must be called on AnalysisResult objects returned from PEI_Transaction_AnalyzeSynch(). (Mandatory).
PEIErrors PEI_Memory_GetMemoryConsumption(PEI_UINT64 *memory);	Get the current memory consumption (in Bytes).

Function	Description
PEIErrors PEI_Memory_SetMaxMemoryConsumption(PEI_UINT64 memory);	Set the maximum memory consumption (in Bytes). The default value is 4294967295 Bytes (there is no need to call this function). Memory boundary is 'softly' enforced. This means that it is not enforced when allocating buffers, but is enforced when submitting a transaction for analysis. If a transaction is submitted when the memory consumption is above the 'MaxMemoryConsumption,' the analysis request will end with an error code of 'full queue.' It allows the user of the adapter to resubmit the transaction later when the memory consumption is lower.
PEIErrors PEI_Memory_GetMaxMemoryConsumption(PEI_UINT64 *memory);	Get the maximum memory consumption (in Bytes).

General Functions

Given below is a list of other general functions used for configuring, reconfiguring, printing, starting up, shutting down, destroying, or retrieving statistics for interface operations.

Function	Description
PEIErrors PEI_LogConfig(const char *filePath);	Configure the Log System.
PEIErrors PEI_SetLogPriority(enum PEI_LOGS log, enum PEI_LOG_LEVEL level);	Dynamically set the priority level of the logs.
PEIErrors PEI_Startup(struct AdapterConfig *config); PEIErrors PEI_StartupByFile(const char *fileName); PEIErrors PEI_Shutdown();	Startup and shutdown — startup must be called before any other call; shutdown must be called for graceful shutdown. config is the configuration of the Policy Engine Interface. If you wish to disable OpenSSL MT initialization, and use your own initialization, you should define PEI_NO_SSL_MT above the #include 'PolicyEngineInterfaceC.h' line in your code.
PEIErrors PEI_Destroy();	Immediately destroys all data structures. Should be called after PEI_Shutdown().
PEIErrors PEI_Reconfigure(struct AdapterConfig *config); PEIErrors PEI_ReconfigureByFile(const char *fileName);	Reconfigure the Policy Engine Interface while in operation.
PEIErrors PEI_GetStatistics(AdapterStatistics *stat);	Retrieve statistics about the interface operation.

Function	Description
const char* PEI_TranslateErrorCode(PEIErrors error);	Get UTF-8 description of errors.
PEIErrors PEI_Print_Title(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_Channel(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_Timeout(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_TransactionID(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_ServiceID(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_SubID(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_BypassBlock(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_Priority(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_Buffer(PEIHandle transaction, char *output, PEI_UINT32 outputSize, PEI_UINT32 bufferOffset, PEI_UINT32 chunkSize, PEI_INT32 *bufferPath, PEI_UINT32 *BufferSize, PEI_UINT32 *numOfSiblings, PEI_UINT32 *numOfSons); PEIErrors PEI_Print_Source(PEIHandle transaction, char *output, PEI_UINT32 outputSize); PEIErrors PEI_Print_Destinations(PEIHandle transaction, char *output, PEI_UINT32 outputSize);	Print functions.

Structs and Enums

General primitives for improved portability:

```
typedef short               PEI_INT16;
typedef unsigned short      PEI_UINT16;
typedef long                PEI_INT32;
typedef unsigned long       PEI_UINT32;
typedef long long           PEI_INT64;
typedef unsigned long long PEI_UINT64;

typedef enum _PEIBool
{
       PEI_FALSE = 0,
       PEI_TRUE
} PEIBool;Basic opaque handle to policy engine entities:
```

This is a handle definition which may refer to transaction, context, content or destination:

```
typedef PEI_UINT64 PEIHandle;
```

Return code of most functions

```
typedef enum _PEIErrors
{
      PEI_ERROR_NO_ERROR = 0,
      PEI_ERROR_BAD_HANDLE,
      PEI_ERROR_NULL_POINTER,
      PEI_ERROR_CONFIGURATION_ERROR,
      PEI_ERROR_FILE_FAULT,
      PEI_ERROR_NULL_BUFFER,
      PEI_ERROR_BUFFER_OUT_OF_RANGE,
      PEI_ERROR_BAD_PARAMETERS,
      PEI_ERROR_TRANSACTION_ID_DOES_NOT_EXIST,
      PEI_ERROR_FULL_QUEUE,
      PEI_ERROR_TIME_OUT,
      PEI_ERROR_BAD_CONFIGURATION,
      PEI_ERROR_LIBRARY_NOT_INITIALIZED,
      PEI_ERROR_NO_LICENSE,
      PEI_ERROR_UNKNOWN_SERVICE,
      PEI_ERROR_MEMORY_ALLOCATION,
      PEI_ERROR_NO_SERVER,
      PEI_ERROR_GENERAL
} PEIErrors;
```

Context

A Context is information about the transaction - mainly sender and recipient(s). Every transaction can have multiple ContextAddresses of the following types:

```
enum ContextAddressType
{
	PEI_EMailAddress,
	PEI_IP,
	PEI_Host,
	PEI_UserDisplayName,
	PEI_UserLoginName,
	PEI_UserSID,
	PEI_Port,
	PEI_Category,
	PEI_DeviceName,
	PEI_DeviceType,
	PEI_AppName,
	PEI_URL,
	PEI_DNSDomainName,
	PEI_NTDomainName,
	PEI_PrinterName,
	PEI_Other_Address,
	PEI_FormattedUserDetails,
	PEI_LDAPDistinguishName,
	PEI_LDAPServerIdentity
};
```

Content

A content part is a buffer of information. Each transaction should contain at least one content part. Each part should have one of the following types:

```
enum ContentPartType
{
	PEI_Other,
	PEI_Subject,
	PEI_Body,
	PEI_Attachment,
	PEI_File,
	PEI_BCCField,
	PEI_CCField,
	PEI_TOField,
	PEI_FROMField,
	PEI_SENTField,
	PEI_PriorityField,
	PEI_ContentDescriptor,
	PEI_HTTPHeaders,
	PEI_MetaData,
	PEI_HTTPPost,
	PEI_HTTPGet,
	PEI_Forensics,
	PEI_FormData
};
```

Level of required information for incidents

```
enum LevelOfRequiredDetails
{
      PEI_NONE,
      PEI_MINIMAL,
      PEI_FULL
};
```

Analysis result - required action

```
typedef enum _PEIActionType
{

PEI_UNKNOWN = 0,              /* usually an error occurred
                              */
PEI_ALLOW = 1,                /* let the transaction go */
PEI_PE_REQUEST = 2,           /* The transaction was
                              identified as a PolicyEngine
                              request message, hence the
                              agent should block it without
                              generating an incident. */
PEI_CONFIRM_ALLOW = 4,        /* Ask the user to confirm
                              the operation, allow if no
                              response */
PEI_CONFIRM_BLOCK = 8,        /* Ask the user to confirm
                              the operation, block if no
                              response */
EI_ENCRYPT = 16,              /* Encrypt the transaction
                              (or forward to an encryption
                              gateway) */
PEI_DROP = 32,                /* Remove the related part
                              from the transaction */
PEI_BLOCK = 64                /* block the transaction */

} PEIActionType;


typedef struct _DropList
{

PEI_UINT32 offset;            /* The offset of the part to
                              drop in bytes */
char *partName;               /* optional part name */

} DropList;
```

List of desired result actions

```
typedef struct _ActionList
{

PEI_UINT32entryNumber;        /* The index number of the
                              relevant
                              PEI_Context_AddDestination()
                              call. For example the first
                              call to the function is entry
                              number 0 (zero), and the
                              fifth call is entry number 4.
                              */
PEIActionType action;         /* The relevant (most severe)
                              action */
PEI_UINT32                    /* other (less severe)
additionalActions;            actions */
DropList   *dropList;         /* Optional - Will appear
                              only if different from the
                              global list */

PEI_UINT32
dropListSize;


} ActionList;
```

Actual result of the analysis

```
struct AnalysisResult
{

PEI_UINT64                    /* Unique across the system.
transactionID;                In case the transaction
                              generated an incident, this
                              ID can be used to locate the
                              incident in the Management UI
                              */
PEIErrors   errorCode;        /* Did analysis succeed, and
                              if not, why */
PEIActionType                 /* The desired action on the
globalAction;                 transaction - This is the
                              most severe action on the
                              transaction. */
PEI_UINT32                    /* other actions to take on
additionalActions;            the transaction - these
                              actions (if exist) will be
                              less severe than the
                              globalAction. */
```

```
ActionList  *actionList;      /* If the transaction had
                              multiple destinations
                              (recipients), the result may
                              have a different action per
                              destination. This list
                              describes the destinations
                              which received an action
                              which is different from the
                              global action. */
PEI_UINT32                    /* The size of 'actionList'
actionListSize;               */
char *analysisDetails;        /* In case of a policy
                              breach, this is a UTF-8 XML
                              with details about the breach
                              */
void *userContext;            /* User defined value, for
                              async analysis */
char  **TransactionLog;       /* If the transaction was
                              marked. This field will hold
                              the processing log of the
                              PolicyEngine */
PEI_UINT32                    /* The size of the
TransactionLogSize;           transaction log */
DropList *dropList;           /* for SMTP transaction this
                              field may contain parts (of
                              the MIME) which need to be
                              dropped */
PEI_UINT32 dropListSize;      /* The size of 'dropList' */
PEIBool wasAudited;           /* Was an incident logged or
                              not." */


};
```

Asynchronous Analysis

This is a callback signature to be used in Asynchronous mode:

```
typedef void (*AnalysisCallback)(struct AnalysisResult *);
```

Statistics

```
typedef struct _AdapterStatistics
{
   PEI_UINT64 upTime;
      // Microseconds since startup().
   PEI_UINT32 submittedTrans;
      // The number of
      // transactions which
      // were submitted for analysis.
```

```
PEI_UINT64 submittedBytes;
   // Same as above but in bytes.
PEI_UINT32 completedTrans;
   // The number of transactions
   // successfully analyzed by a
   // PolicyEngine.
PEI_UINT32 pendingTrans;
   // submittedTrans -
   // failedTransactions -
   // completedTrans -
   // completedTransWithTimeout -
   // completedTransWithError
PEI_UINT32 failedTransactions;
   // The number of transactions
   // which were submitted for
   // analysis, but rejected
   // as invalid transactions.
PEI_UINT32 queueLoad;
   // The number of transactions in
   // the adapter queue.
PEI_UINT32 queueMaxLoad;
   // The maximum recorded
   // "queueload" since
   // the last retrieval of counters
   // (GetStatistics()).
PEI_UINT32 queueMaxSize;
   // The max size of the queue.
PEI_UINT32 reportedErrors;
   // each time the adapter reports
   // an error - this counter
   //increases by one.
PEI_UINT32 outstandingTransactionHandles;
   // The number of transaction
   // handles yet to be destroyed.
PEI_UINT32 outstandingContextHandles;
   // The number of context handles
   // yet to be destroyed.
PEI_UINT32 outstandingContentHandles;
   // The number of content handles
   // yet to be destroyed.
PEI_UINT32 completedTransWithTimeout;
   // The number of analysis
   // requests which ended in
   // timeout.
PEI_UINT32 completedTransWithError;
```

```
        // The number of analysis requests
        // which ended in a reported error
        // by the PolicyEngine.
    PEI_UINT64 memoryConsumption;
        // The total size of currently
        // allocated buffers.
} AdapterStatistics;
```

Policy Engine Interface Configuration

Usually, the Policy Engine Interface is configured via XML. However, the following structures allow configuring the system in code. This can come in handy for tight integrations.

```
struct ServiceDescriptor
{
        unsigned int id;
        unsigned int channelType;
        struct ServiceDescriptor *next;
};


struct ServerSupportedChannel
{
        int channel;
        struct ServerSupportedChannel *next;
};


struct ServerSupportedChannels
{
        int supportAll;
        struct ServerSupportedChannel *channels;
};


struct SocketDescriptor
{
        int isSecured;
        PEI_UINT16 port;
        const char* host[2];
        struct ServerSupportedChannels supportedChannels;
        struct SocketDescriptor *next;
};


struct PipeDescriptor
{
```

```
        int isSecured;
        const char* name;
        struct ServerSupportedChannels supportedChannels;
        struct PipeDescriptor *next;
};


struct AutomaticReconfigure
{
        const char* configurationPath;
                // The path from which to read
                // (write) the configuration.
        const char* certificatePath;
                // The path of the certificate for
                // contacting the mgmtd process.
        int refreshInterval;
                // How often to check for
                // configuration updates (in ms).
int agentType;
                // A fixed identification number given
                // by Websense.
const char* agentHost;
                // Identification string for
                // contacting the mgmtd process.
const char* mgmtdAddress;
                // The address of the mgmtd process
                // to use for updating the
                // configuration.
};


struct AdapterConfig
{
        PEI_UINT32 requestThreadQueueSize;
                // Queue size for Analyze requests.
        PEI_UINT32 replyThreadQueueSize;
                // Queue size for analysis results.
        PEI_UINT32 orangeToGreenThreshold;
                // Trafficlight algo. Parameter.
        PEI_UINT32 greenToOrangeThreshold;
                // Trafficlight algo. Parameter.
        PEI_UINT32 redToOrangeThreshold;
                // Trafficlight algo. Parameter.
        PEI_UINT32 orangeToRedThreshold;
                // Trafficlight algo. Parameter.
```

```
        PEI_UINT32 orangeSelectionThreshold;
               // Trafficlight algo. Parameter.
        PEI_UINT32 redSelectionThreshold;
               // Trafficlight algo. Parameter.
        PEI_UINT32 numOfRequestThreads;
               // # of threads which handle Analyze requests.
        PEI_UINT32 numOfReplyThreads;
               // # of threads which handle Analysis results.
        PEI_UINT32 refreshingMsInterval;
               // how often should the PEI refresh its status.
        PEI_UINT32 connectionTimeout;
               // how long to wait for a connection
               // to establish.
        PEI_UINT32 loggerAutomaticUpdate;
               // should the logger automatically update
               // its state.
        PEI_UINT32 serverSelectionMsTimeout;
               // how much time to spend for selecting
               // a server.
        struct SocketDescriptor *sockets;
               // socket connections.
        struct PipeDescriptor *pipes;
               // pipe connections.
        struct ServiceDescriptor *channels;
               // relevant services.
        struct AutomaticReconfigure *automaticReconfigure;
               // Integration Agent -
               // How should the PEI configure itself.
};
```

Using the Integration Agent

After installing an Integration Agent, the installation directory will contain a (preconfigured) configuration file. In order to start the PolicyEngine Interface correctly, one must call the PEI_StartupByFile(const char *fileName) function, with fileName set as the path to the configuration file. Another option is to set the above mentioned structs correctly and call the PEI_Startup(struct AdapterConfig *config) function.

Note:
An Integration Agent user should never call the API functions: PEI_Reconfigure() and PEI_ReconfigureByFile().

Available log topics

```
enum PEI_LOGS
{
        PEI_LOG_COMMUNICATION,
        PEI_LOG_PEINTERFACE,
```

```
        PEI_LOG_TRAFFIC,
        PEI_LOG_LOAD_BALANCING,
        PEI_LOG_TRANSACTION,
        PEI_LOG_ROOT,
        PEI_LOG_REFRESHER
};
```

Available log levels

```
enum PEI_LOG_LEVEL
{
        PEI_LOG_LEVEL_FATAL,
        PEI_LOG_LEVEL_ERROR,
        PEI_LOG_LEVEL_DEBUG,
        PEI_LOG_LEVEL_INFO,
        PEI_LOG_LEVEL_OFF,
        PEI_LOG_LEVEL_ALL
   };
```

Orientation – The direction of the transaction

```
typedef  enum _PEIOrientation
{
        PEI_OUTBOUND,
        PEI_INBOUND,
        PEI_INTERNAL
   } PEIOrientation;
```