IBM® Netezza® Analytics
Release 3.3.5.0

Perl Analytic Executables API Reference

Part Number 00X6638-00 Rev. 1

Note: Before using this information and the product that it supports, read the information in "Notices and Trademarks" on page 75.

Contents

Preface

1 Class Documentation

Notices and Trademarks

Index

Preface

This guide provides an API reference for Perl AE programmers.

Audience for This Guide

The *Perl Analytic Executables API Reference* is written for programmers who intend to create Analytic Executables for IBM Netezza Analytics using the Perl language. This guide does not provide a tutorial on AE concepts. More information about AEs can be found in the *User-Defined Analytic Process Developer's Guide*.

Purpose of This Guide

This guide describes the Perl AE API, which is a language adapter provided as part of IBM Netezza Analytics. The Perl AE API provides programmatic access to the AE interface for Perl programmers.

Conventions

Note on Terminology: The terms User-Defined Analytic Process (UDAP) and Analytic Executable (AE) are synonymous.

The following conventions apply:

- *Italics* for emphasis on terms and user-defined values, such as user input.
- Upper case for SQL commands, for example, INSERT or DELETE.
- Bold for command line input, for example, **nzsystem stop**.
- Bold to denote parameter names, argument names, or other named references.
- Angle brackets (< >) to indicate a placeholder (variable) that should be replaced with actual text, for example, **nzmat <- nz.matrix("<matrix_name>")**.
- A single backslash ("\") at the end of a line of code to denote a line continuation. Omit the backslash when using the code at the command line, in a SQL command, or in a file.
- When referencing a sequence of menu and submenu selections, the ">" character denotes the different menu options, for example *Menu Name > Submenu Name > Selection*.

If You Need Help

If you are having trouble using the IBM Netezza appliance, IBM Netezza Analytics or any of its components:

1. Retry the action, carefully following the instructions in the documentation.
2. Go to the IBM Support Portal at http://www.ibm.com/support. Log in using your IBM ID and password. You can search the Support Portal for solutions. To submit a support request, click the 'Service Requests & PMRs' tab.
3. If you have an active service contract maintenance agreement with IBM, you can contact customer support teams via telephone. For individual countries, please visit the Technical Support section of the IBM Directory of worldwide contacts

http://www14.software.ibm.com/webapp/set2/sas/f/handbook/contacts.html#phone.

Comments on the Documentation

We welcome any questions, comments, or suggestions that you have for the IBM Netezza documentation. Please send us an e-mail message at netezza-doc@wwpdl.vnet.ibm.com and include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

We appreciate your comments.

CHAPTER 1
Class Documentation

Ae Class Reference

This class provides the primary interface for running a Perl Analytic Executable.

Constructor Member Functions

- new(scalar class)
  The object constructor.

Destructor Member Functions

- DESTROY()
  The object destructor.

Logging Member Functions

- openLog(scalar logfile)
  Opens a logfile.
- closeLog()
  Closes a logfile.
- writeToLog(scalar msg)
  Writes the specified message to a logfile.
- writeToLogVerbose(scalar msg)
  Writes the specified message to the logfile if debugging is enabled.
- SetNonBlock(scalar filehandle)
  Puts a filehandle in nonblocking mode.
- initDebug()
  Initializes the name for the logfile and calls the openLog function.

- getLogLevelDebug()
  Gets the log level debug.
- getLogLevelTrace()
  Gets the log level trace.
- log(scalar text, scalar logLevel)
  Uses AE logging to display logging text to the user.

Attribute Fetching Member Functions

- getDataTypeBool()
  Gets the Data Type Bool.
- getDataTypeInt8()
  Gets the Data Type Int 8.
- getDataTypeInt16()
  Gets the Data Type Int 16.
- getDataTypeInt32()
  Gets the Data Type Int 32.
- getDataTypeInt64()
  Gets the Data Type Int 64.
- getDataTypeDouble()
  Gets the Data Type Double.
- getDataTypeFloat()
  Gets the Data Type Float.
- getDataTypeNumeric32()
  Gets the Data Type Numeric 32.
- getDataTypeNumeric64()
  Gets the Data Type Numeric 64.
- getDataTypeNumeric128()
  Gets the Data Type Numeric 128.
- getDataTypeFixed()
  Gets the Data Type Fixed.
- getDataTypeVariable()
  Gets the Data Type Variable.
- getDataTypeNationalFixed()
  Gets the Data Type National Fixed.
- getDataTypeNationalVariable()
  Gets the Data Type National Variable.
- getDataTypeVarbinary()
  Gets the Data Type Varbinary.
- getDataTypeNationalGeometry()

Gets the Data Type Geometry.

- getAggregateTypeInitialize()
  Gets the Aggregation Type Initialize.
- getAggregateTypeAccumulate()
  Gets the Aggregation Type Accumulate.
- getAggregateTypeMerge()
  Gets the Aggregation Type Merge.
- getAggregateTypeFinalResult()
  Gets the Aggregation Type Final Result.
- getAggregateTypeEnd()
  Gets the Aggregation Type End.
- getAggregateTypeError()
  Gets the Aggregation Type Error.
- getDataStringHash()
  Gets String Data Types in a hash.
- getDataNumericHash()
  Gets Numeric Data Types in a hash.
- getDataIntegerHash()
  Gets Integer Data Types in a hash.

Metadata Member Functions

- getDataTypeName(scalar inputType)
  Gets the name of a specified data type.
- getInputTypes()
  Gets a list of input data types.
- getNumberOfInputColumns()
  Gets the number of input columns.
- getNumberOfOutputColumns()
  Gets the number of output columns.
- getInputPrecision(scalar columnIndex)
  Gets the precision of an input numeric.
- getInputScale(scalar columnIndex)
  Gets the scale of an input numeric.
- getInputSize(scalar columnIndex)
  Gets the size of input column at specified index.
- getInputType(scalar columnIndex)
  Gets the data type of input column at specified index.
- getOutputPrecision(scalar columnIndex)
  Gets the precision of an output numeric.
- getOutputScale(scalar columnIndex)

Gets the scale of an output numeric.

- getOutputSize(scalar columnIndex)
  Gets the size of output column at specified index.
- getOutputType(scalar columnIndex)
  Gets the data type of output column at specified index.
- getUdfReturnType(scalar columnIndex)
  Gets the return data type for a UDF.
- isCalledWithOrderByClause()
  Determines if the function was called with an ORDER BY clause.
- isCalledWithOverClause()
  Determines if the function was called with an OVER clause.
- isCalledWithPartitionByClause()
  Determines if the function was called with a PARTITION BY clause.
- isDataInnerCorrelated()
  Determines if the function data is inner-correlated.
- isDataLeftCorrelated()
  Determines if the function data is left correlated.
- isDataUncorrelated()
  Determines if the function data is uncorrelated.

Run-Time Member Functions

- getCurrentUsername()
  Gets the database username of the current AE being run.
- getDatasliceId()
  Determines the ID of the dataslice being serviced by the AE.
- getHardwareId()
  Determines the ID of the hardware on which the AE is running.
- getNumberOfDataSlices()
  Gets the number of dataslices that are running on the Netezza appliance.
- getNumberOfSpus()
  Gets the number of SPUs running on the Netezza appliance.
- getSuggestedMemoryLimit()
  Gets the suggested memory limit for the AE that is running.
- getSessionId()
  Gets the session ID associated with the AE that is running.
- getTransactionId()
  Gets the transaction ID associated with the AE that is running.
- isAUserQuery()
  Determines if the AE is handling a user query.

- isLoggingEnabled()
  Determines if logging is enabled.
- isRunningInPostgres()
  Determines if the AE is running in Postgres.
- isRunningInDbos()
  Determines if the AE is running in DBOS.
- isRunningOnSpu()
  Determines if the AE is running on the SPU.
- isRunningOnHost()
  Determines if the AE is running on the host.

Remote AE Member Functions

- setAeInfo(scalar aeInfo)
  Sets the AeInfo for the AE object that is running.
- getRemoteProtocolCodeControlData()
  Gets the Remote Protocol message type Control Data.
- getRemoteProtocolCodePing()
  Gets the Remote Protocol message type Ping.
- getRemoteProtocolCodeRequest()
  Gets the Remote Protocol message type Code Request.
- getRemoteProtocolCodeStatus()
  Gets the Remote Protocol message type Code Status.
- getRemoteProtocolCodeStop()
  Gets the Remote Protocol message type Code Stop.
- getDefaultRequestHandlingStyle()
  Gets the default remote request handling style.
- setDefaultRequestHandlingStyle(scalar style)
  Sets the default remote request handling style.
- getRequestHandlingStyleFork()
  Gets the remote request handling style fork handle.
- getRequestHandlingStyleSingleThreaded()
  Gets the remote request handling style single-threaded handle.
- getConnectionPointName()
  Returns the connection point name for the AE.
- getConnectionPointDatasliceId()
  Returns the connection point dataslice ID for the AE.
- getConnectionPointSessionId()
  Returns the connection point session ID for the AE.
- getConnectionPointTransactionId()
  Returns the connection point transaction ID for the AE.

- setConnectionPointName(scalar connectionPointName)
  Sets the connection point name for the AE.
- setConnectionPointDatasliceId(scalar datasliceId)
  Sets the connection point dataslice ID for the AE.
- setConnectionPointSessionId(scalar sessionId)
  Sets the connection point session ID for the AE.
- setConnectionPointTransactionId(scalar transactionId)
  Sets the connection point transaction ID for the AE.

General Member Functions

- isLocal()
  Determines if the AE that is currently running is a local AE.
- isRemote()
  Determines if the AE that is currently running is a remote AE.
- isAggregateAe()
  Determines if the AE that is currently running is an aggregate AE.
- isFunctionAe()
  Determines if the AE that is currently running is a function AE.
- isShaperAe()
  Determines if the AE that is currently running is a shaper/sizer AE.
- isUda()
  Determines if the AE that is currently running is an aggregate AE.
- isUdf()
  Determines if the AE that is currently running is a UDF-based AE.
- isUdfSizer()
  Determines if the AE that is currently running is a UDF sizer AE.
- isUdtf()
  Determines if the AE that is currently running is a UDTF-based AE.
- isUdtfShaper()
  Determines if the AE that is currently running is a UDTF shaper AE.
- didClassRun()
  Determines if the class was run.
- run()
  Runs an AE.
- pingNps()
  Notifies the Netezza software that work is still being performed and the AE should not be terminated.

Overridable Member Functions

- runLocal()
  This function can be overridden to control running of Local AEs.
- runRemote()
  This function can be overridden to control running of Remote AEs.
- _doRemoteRunWork(scalar aeInfo)
  This function can be overridden to control how remote AEs are run during single-threaded operation.
- _onIdle()
  This function can be overridden to handle idle time for remote AEs.
- runInstance(scalar dontexit)
  This function can be overridden to control generic running of an AE instance.
- _setup()
  This function can be overridden for one-time setup of the AE instance.
- _run()
  This function can be overridden to handle generic running of AEs.
- runUdtf()
  This function can be overridden to handle running a UDTF.
- runUdf()
  This function may be overridden to handle running a UDF.
- runShaper()
  This function must be overridden to handle running a Shaper.
- runSizer()
  This function must be overridden to run Sizer function.
- _cleanUp()
  This function may be overridden for one-time clean-up of the AE.
- iter()
  This function may be overridden to iterate over input rows, call getFunctionResult and send to output.
- _initializeResult()
  This function must be overridden to initialize state for an aggregate function.
- _accumulate()
  This function must be overridden to run aggregation functionality over values in the same dataslice.
- _merge()
  This function must be overridden to merge states gathered from various dataslices.
- _finalResult()
  This function must be overridden to set and return the final result for an aggregate function.
- runUda()
  This function runs the user-defined aggregate function.
- _handleRemoteProtocol(scalar code, scalar data)
  This function may be overridden to handle generic remote protocol functionality.
- _handleRemoteProtocolControlData(scalar data)

This function may be overridden to handle generic remote protocol control data.

- _handleRemoteProtocolStopCommand()
  This function may be overridden to handle the remote protocol stop command in a non-standard way.
- _handleRemoteProtocolRequest(scalar request)
  This function may be overridden to handle generic remote protocol requests.

Error Handling Member Functions

- _handleException(scalar exception)
  Handles exceptions by writing to the logfile and sending the user the error, if needed.

Output Member Functions

- output(list args)
  Sets the output of the AE.
- setOutputWithList(list theList)
  Sets the output value using a list.
- setOutputValue(scalar columnIndex, scalar value)
  Checks the output value type using a library function and calls the appropriate set output function.
- setOutputNull(scalar columnIndex)
  Sets the value at column specified by column index to null.
- setOutputString(scalar columnIndex, scalar value, scalar length)
  Sets the value at column specified by column index to a string value.
- setOutputInt32(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to an Int 32 value.
- setOutputInt64(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to an Int 64 value.
- setOutputInt8(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to an Int 8 value.
- setOutputInt16(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to an Int 16 value.
- setOutputFloat(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to a float value.
- setOutputDouble(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to a double value.
- setOutputBool(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to a boolean value.
- setOutputUdsData(scalar columnIndex, scalar value)
  Sets the value at the column specified by the column index to a generic value.

Input Member Functions

- getInputRow()
  Gets a single input row as an array.
- getInputValue(scalar columnIndex)
  Looks up the relevant element in the row.
- getInputStringLength(scalar columnIndex)
  Gets the string length value at column specified by column index.
- getInputString(scalar columnIndex)
  Gets the string value at column specified by column index.
- getInputInt32(scalar columnIndex)
  Gets the Int32 value at column specified by column index.
- getInputInt64(scalar columnIndex)
  Gets the Int64 value at column specified by column index.
- getInputInt8(scalar columnIndex)
  Gets the Int8 value at column specified by column index.
- getInputInt16(scalar columnIndex)
  Gets the Int 16 value at column specified by column index.
- getInputFloat(scalar columnIndex)
  Gets the float value at column specified by column index.
- getInputDouble(scalar columnIndex)
  Gets the double value at column specified by column index.
- getInputUdsData(scalar columnIndex)
  Gets the generic value at column specified by column index.
- getNext()
  Loads the next input row into memory.

Aggregate Member Functions

- getState(scalar columnIndex)
  Gets the state for a given column index or all indices.
- getInputState(scalar columnIndex)
  Gets the input state for a given column index or all indices.
- _setState(list indexOrState, list inState)
  Sets a single aggregate value or a list of aggregate values.
- getStateValue(scalar index, scalar getMergedState)
  Gets value of a state variable.
- getStateString(scalar columnIndex, scalar getInputState)
  Gets the string state variable value at column specified by column index.
- getStateInt32(scalar columnIndex, scalar getInputState)
  Gets the Int32 state variable value at column specified by column index.
- getStateInt64(scalar columnIndex, scalar getInputState)

Gets the Int64 state variable value at column specified by column index.

- getStateInt8(scalar columnIndex, scalar getInputState)
  Gets the Int8 state variable value at column specified by column index.
- getStateInt16(scalar columnIndex, scalar getInputState)
  Gets the Int16 state variable value at column specified by column index.
- getStateFloat(scalar columnIndex, scalar getInputState)
  Gets the float state variable value at column specified by column index.
- getStateDouble(scalar columnIndex, scalar getInputState)
  Gets the double state variable value at column specified by column index.
- getStateUdsData(scalar columnIndex, scalar getInputState)
  Gets the generic state variable value at column specified by column index.
- _setAggregateResult(list result, scalar forFinalResult)
  Sets the aggregate result value.
- _setAggregateValue(scalar index, scalar value, scalar forFinalResult)
  Sets the return value for an aggregate.
- setAggregateNull(scalar columnIndex)
  Sets the aggregate or state value at column specified by column index to null.
- setAggregateString(scalar columnIndex, scalar value, scalar length)
  Sets the aggregate or state value at column specified by column index to a string value.
- setAggregateInt32(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to an Int 32 value.
- setAggregateInt64(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to an Int 64 value.
- setAggregateInt8(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to an Int 8 value.
- setAggregateInt16(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to an Int 16 value.
- setAggregateFloat(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to a float value.
- setAggregateDouble(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to a double value.
- setAggregateUdsData(scalar columnIndex, scalar value)
  Sets the aggregate or state value at the column specified by the column index to a generic value.
- getNumberOfStateColumns()

Gets the number of state columns.

- getStatePrecision(scalar columnIndex)
  Gets the precision of a field of state at the specified column index.
- getStateScale(scalar columnIndex)
  Gets the scale of a field of state at the specified column index.
- getStateSize(scalar columnIndex)
  Gets the size of a field of state at the specified column index.
- getStateType(scalar columnIndex)
  Gets the data type of state field at specified index.

Environment Member Functions

- getEnvironment()
  Returns a hash containing the entire environment.
- getEnvironmentVariable(scalar variableName, list defaultValue)
  Gets the value of an AE environment variable.

Shared Library Member Functions

- getSharedLibraryPath(scalar libraryName, scalar caseSensitive)
  Gets the path to a shared library.
- yieldSharedLibraries(scalar forProcess)
  Returns a hash that yields a shared library name and the corresponding full path for each shared library entry.

Detailed Description

This class provides the primary interface for running a Perl Analytic Executable.

The functions in this class provide an interface to the underlying API, enabling the user to write an AE in Perl.

Public Member Function Documentation

- **new(scalar class)**
  The object constructor.
  - Parameters
    - **$class**
      The package name.
  - Returns
    (object) The new class object.

  This function calls the _AeInternal constructor and creates a new AE object.

- **DESTROY()**
  The object destructor.

- Returns
  None.

This function is a finalization method that cleans up the class state when there are no more references to an object or the program exits.

- **openLog(scalar logfile)**
  Opens a logfile.
  - Parameters
    - **$logfile**
      The logfile name.
  - Returns
    None.

  This function opens a logfile and sets it to non-blocking mode. It also sets the environment variable nzaeDebugLog to the file handle.

- **closeLog()**
  Closes a logfile.
  - Returns
    None.

- **writeToLog(scalar msg)**
  Writes the specified message to a logfile.
  - Parameters
    - **$msg**
      (String) Optional. The message to be written.
  - Returns
    None.

  This function writes a message to the logfile specified by the nzaeDebugLog environment variable.

- **writeToLogVerbose(scalar msg)**
  Writes the specified message to the logfile if debugging is enabled.
  - Parameters
    - **$msg**
      The message to be written.
  - Returns
    None.

  This function determines if the debug level has been set greater than 0. If so, it writes the mes-

sage to the logfile.

- **SetNonBlock(scalar filehandle)**
  Puts a filehandle in nonblocking mode.
  - Parameters
    - **$filehandle**
      The filehandle for which nonblocking mode is set.
  - Returns
    None.

  This function puts a filehandle into nonblocking mode.

- **initDebug()**
  Initializes the name for the logfile and calls the openLog function.
  - Returns
    None.

  This function determines if debugging is enabled for the system. If so, the logfile name is created using environment variables and openLog is called.

- **getLogLevelDebug()**
  Gets the log level debug.
  - Returns
    (integer) The value of _AeInternal.LOG_LEVEL__DEBUG.

- **getLogLevelTrace()**
  Gets the log level trace.
  - Returns
    (integer) The value of _AeInternal.LOG_LEVEL__TRACE.

- **log(scalar text, scalar logLevel)**
  Uses AE logging to display logging text to the user.
  - Parameters
    - **$text**
      (string) The log text.
    - **$logLevel**
      (string) The log level.
  - Returns
    None.

- **getDataTypeBool()**

Gets the Data Type Bool.

- Returns
  (integer) The value of _AeInternal.DATA_TYPE__BOOL.

- **getDataTypeInt8()**
  Gets the Data Type Int 8.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__INT8.

- **getDataTypeInt16()**
  Gets the Data Type Int 16.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__INT16.

- **getDataTypeInt32()**
  Gets the Data Type Int 32.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__INT32.

- **getDataTypeInt64()**
  Gets the Data Type Int 64.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__INT64.

- **getDataTypeDouble()**
  Gets the Data Type Double.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__DOUBLE.

- **getDataTypeFloat()**
  Gets the Data Type Float.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__FLOAT.

- **getDataTypeNumeric32()**
  Gets the Data Type Numeric 32.
  - Returns

(integer) The value of _AeInternal.DATA_TYPE__NUMERIC32.

- **getDataTypeNumeric64()**
  Gets the Data Type Numeric 64.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__NUMERIC64.

- **getDataTypeNumeric128()**
  Gets the Data Type Numeric 128.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__NUMERIC128.

- **getDataTypeFixed()**
  Gets the Data Type Fixed.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__FIXED.

- **getDataTypeVariable()**
  Gets the Data Type Variable.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__VARIABLE.

- **getDataTypeNationalFixed()**
  Gets the Data Type National Fixed.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__NATIONAL_FIXED.

- **getDataTypeNationalVariable()**
  Gets the Data Type National Variable.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__NATIONAL_VARIABLE.

- **getDataTypeVarbinary()**
  Gets the Data Type Varbinary.
  - Returns
    (integer) The value of _AeInternal.DATA_TYPE__VARBINARY

- **getDataTypeNationalGeometry()**
  Gets the Data Type Geometry.

- Returns
  (integer) The value of _AeInternal.DATA_TYPE__GEOMETRY.

- **getAggregateTypeInitialize()**
  Gets the Aggregation Type Initialize.
  - Returns
    (integer) The value of _AeInternal.AGGREGATION_TYPE__INITIALIZE.

- **getAggregateTypeAccumulate()**
  Gets the Aggregation Type Accumulate.
  - Returns
    (integer) The value indicating _AeInternal.AGGREGATION_TYPE__ACCUMULATE.

- **getAggregateTypeMerge()**
  Gets the Aggregation Type Merge.
  - Returns
    (integer) The value indicating _AeInternal.AGGREGATION_TYPE__MERGE.

- **getAggregateTypeFinalResult()**
  Gets the Aggregation Type Final Result.
  - Returns
    (integer) The value indicating _AeInternal.AGGREGATION_TYPE__FINAL_RESULT.

- **getAggregateTypeEnd()**
  Gets the Aggregation Type End.
  - Returns
    (integer) The value of _AeInternal.AGGREGATION_TYPE__END.

- **getAggregateTypeError()**
  Gets the Aggregation Type Error.
  - Returns
    (integer) The value of _AeInternal.AGGREGATION_TYPE__ERROR.

- **getDataStringHash()**
  Gets String Data Types in a hash.
  - Returns
    hash<integer,string> The integer denotes the value of string type; the string denotes the name of string type.

- **getDataNumericHash()**
  Gets Numeric Data Types in a hash.
  - Returns
    hash<integer,string> The integer denotes the value of numeric type; the string denotes the name of numeric type.

- **getDataIntegerHash()**
  Gets Integer Data Types in a hash.
  - Returns
    hash<integer,string> The integer denotes the value of integer type; the string denotes the name of integer type.

- **getDataTypeName(scalar inputType)**
  Gets the name of a specified data type.
  - Parameters
    - **$inputType**
      (DATA_TYPE) The data type.
  - Returns
    (string) The name of the specified data type.

- **getInputTypes()**
  Gets a list of input data types.
  - Returns
    The list of input data types.

- **getNumberOfInputColumns()**
  Gets the number of input columns.
  - Returns
    (integer) The number of input columns.

  This function is valid for function and shaper AEs and when aggregate AEs are in the 'accumulate' processing state.

- **getNumberOfOutputColumns()**
  Gets the number of output columns.
  - Returns
    (integer) The number of output columns.

  This function is valid for function AEs and when aggregate AEs are in the 'finalize' processing state.

- **getInputPrecision(scalar columnIndex)**
  Gets the precision of an input numeric.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The precision of the input string at the specified column index.

  This function is valid for function and shaper AEs and when aggregate AEs are in the 'accumulate' processing state.

- **getInputScale(scalar columnIndex)**
  Gets the scale of an input numeric.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The scale of the input string at the specified column index.

  This function is valid for function and shaper AEs and when aggregate AEs are in the 'accumulate' processing state.

- **getInputSize(scalar columnIndex)**
  Gets the size of input column at specified index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The size of the input column at the specified column index.

  This function is valid for function and shaper AEs and when aggregate AEs are in the 'accumulate' processing state.

- **getInputType(scalar columnIndex)**
  Gets the data type of input column at specified index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (DATA_TYPE) The data type of the input column at the specified column index.

  This function is valid for function and shaper AEs and when aggregate AEs are in the 'accumulate' processing state.

- **getOutputPrecision(scalar columnIndex)**
  Gets the precision of an output numeric.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The precision of the output string at the specified column index.

  This function is valid for function AEs and when aggregate AEs are in the 'finalize' processing state.

- **getOutputScale(scalar columnIndex)**
  Gets the scale of an output numeric.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The scale of the output string at the specified column index.

  This function is valid for function AEs and when aggregate AEs are in the 'finalize' processing state.

- **getOutputSize(scalar columnIndex)**
  Gets the size of output column at specified index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The size of the output column at the specified column index.

  This function is valid for function AEs and when aggregate AEs are in the 'finalize' processing state.

- **getOutputType(scalar columnIndex)**
  Gets the data type of output column at specified index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the column.
  - Returns
    (DATA_TYPE) The data type of the output column at the specified column index.

  This function is valid for function AEs and when aggregate AEs are in the 'finalize' processing state.

- **getUdfReturnType(scalar columnIndex)**
  Gets the return data type for a UDF.

- Parameters
  - **$columnIndex**
    (integer) The index of the column.
- Returns
  (DATA_TYPE) The data type of the output field.

This function is valid only for UDF sizer AEs.

- **isCalledWithOrderByClause()**
  Determines if the function was called with an ORDER BY clause.
  - Returns
    (boolean) A value of 1 if the function was called with an ORDER BY clause; otherwise returns 0.

  This function is valid only for function AEs.

- **isCalledWithOverClause()**
  Determines if the function was called with an OVER clause.
  - Returns
    (boolean) A value of 1 if the function was called with an OVER clause; otherwise returns 0.

  This function is valid only for function AEs.

- **isCalledWithPartitionByClause()**
  Determines if the function was called with a PARTITION BY clause.
  - Returns
    (boolean) A value of 1 if the function was called with a PARTITION BY clause; otherwise returns 0.

  This function is valid only for function AEs.

- **isDataInnerCorrelated()**
  Determines if the function data is inner-correlated.
  - Returns
    (boolean) A value of 1 if the function data is inner-correlated; otherwise returns 0.

  This function is valid only for function AEs.

- **isDataLeftCorrelated()**
  Determines if the function data is left correlated.
  - Returns
    (boolean) A value of 1 if the function data is left correlated; otherwise returns 0.

This function is valid only for function AEs.

- **isDataUncorrelated()**
  Determines if the function data is uncorrelated.
  - Returns
    (boolean) A value of 1 if the function data is uncorrelated; otherwise returns 0.

  This function is valid only for function AEs.

- **getCurrentUsername()**
  Gets the database username of the current AE being run.
  - Returns
    (string) The current username.

- **getDatasliceId()**
  Determines the ID of the dataslice being serviced by the AE.
  - Returns
    (integer) The dataslice ID being serviced by the AE.

- **getHardwareId()**
  Determines the ID of the hardware on which the AE is running.
  - Returns
    (integer) The hardware ID for the hardware where the AE is running.

- **getNumberOfDataSlices()**
  Gets the number of dataslices that are running on the Netezza appliance.
  - Returns
    (integer) The number of dataslices running on the Netezza appliance.

- **getNumberOfSpus()**
  Gets the number of SPUs running on the Netezza appliance.
  - Returns
    (integer) The number of SPUs running on the Netezza appliance.

- **getSuggestedMemoryLimit()**
  Gets the suggested memory limit for the AE that is running.
  - Returns
    (integer) The suggested memory limit for the AE that is running.

- **getSessionId()**

Gets the session ID associated with the AE that is running.

- Returns
  (integer) The session ID associated with the AE that is running.

- **getTransactionId()**
  Gets the transaction ID associated with the AE that is running.
  - Returns
    (integer) The transaction ID associated with the AE that is running.

- **isAUserQuery()**
  Determines if the AE is handling a user query.
  - Returns
    (boolean) A value of 1 if the AE is handling a user query; otherwise returns 0.

- **isLoggingEnabled()**
  Determines if logging is enabled.
  - Returns
    (boolean) A value of 1 if logging is enabled; otherwise returns 0.

- **isRunningInPostgres()**
  Determines if the AE is running in Postgres.
  - Returns
    (boolean) A value of 1 if the AE is running in Postgres; otherwise returns 0.

- **isRunningInDbos()**
  Determines if the AE is running in DBOS.
  - Returns
    (boolean) A value of 1 if the AE is running in DBOS; otherwise returns 0.

- **isRunningOnSpu()**
  Determines if the AE is running on the SPU.
  - Returns
    (boolean) A value of 1 if the AE is running on the SPU; otherwise returns 0.

- **isRunningOnHost()**
  Determines if the AE is running on the host.
  - Returns

(boolean) A value of 1 if the AE is running on the host; otherwise returns 0.

- **setAeInfo(scalar aeInfo)**
  Sets the AeInfo for the AE object that is running.
  - Parameters
    - **$aeInfo**
      The AeInfo object.
  - Returns
    None.

- **getRemoteProtocolCodeControlData()**
  Gets the Remote Protocol message type Control Data.
  - Returns
    (integer) The value of _AeInternal.REMOTE_PROTOCOL_CODE__CONTROL_DATA.

- **getRemoteProtocolCodePing()**
  Gets the Remote Protocol message type Ping.
  - Returns
    (integer) The value of _AeInternal.REMOTE_PROTOCOL_CODE__PING.

- **getRemoteProtocolCodeRequest()**
  Gets the Remote Protocol message type Code Request.
  - Returns
    (integer) The value of _AeInternal.REMOTE_PROTOCOL_CODE__REQUEST.

- **getRemoteProtocolCodeStatus()**
  Gets the Remote Protocol message type Code Status.
  - Returns
    (integer) The value of _AeInternal.REMOTE_PROTOCOL_CODE__STATUS.

- **getRemoteProtocolCodeStop()**
  Gets the Remote Protocol message type Code Stop.
  - Returns
    (integer) The value of _AeInternal.REMOTE_PROTOCOL_CODE__STOP.

- **getDefaultRequestHandlingStyle()**
  Gets the default remote request handling style.
  - Returns
    (integer) The value of the default request handling style.

- **setDefaultRequestHandlingStyle(scalar style)**
  Sets the default remote request handling style.
  - Parameters
    - **$style**
      (integer) The value denoting the request handling style for the remote AE.
  - Returns
    None.

  This function sets the default remote request handling style for requests to a Remote AE. Options are getRequestHandlingStyleFork and getRequestHandlingStyleSingleThreaded .

- **getRequestHandlingStyleFork()**
  Gets the remote request handling style fork handle.
  - Returns
    (integer) The value denoting the request fork handling of the remote AE request.

- **getRequestHandlingStyleSingleThreaded()**
  Gets the remote request handling style single-threaded handle.
  - Returns
    (integer) The value denoting the single threaded handling of the remote AE request.

- **getConnectionPointName()**
  Returns the connection point name for the AE.
  - Returns
    (string) The connection point name.

- **getConnectionPointDatasliceId()**
  Returns the connection point dataslice ID for the AE.
  - Returns
    (integer) The dataslice ID; returns -1 if the AE is not running in remote mode by dataslice.

  This function returns the dataslice ID where the AE is running.

- **getConnectionPointSessionId()**
  Returns the connection point session ID for the AE.
  - Returns
    (integer) The session ID; returns -1 if the AE is not running in remote mode by session.

  This function returns the session ID under which the AE is running.

- **getConnectionPointTransactionId()**
  Returns the connection point transaction ID for the AE.
  - Returns
    (integer) The transaction ID; returns -1 if the AE is not running in remote mode by transaction.

  This function returns the transaction ID for the connection point of the AE.

- **setConnectionPointName(scalar connectionPointName)**
  Sets the connection point name for the AE.
  - Parameters
    - **$connectionPointName**
      (string) The name of the connection point.
  - Returns
    None.

  This function sets the connection point name. If using the built-in remote AE launching mechanism, the connection point info is automatically set from the process environment, and this function does not need to be called. An alternative to this function is to allow the deriving class to override the class variable, CONNECTION_POINT_NAME.

- **setConnectionPointDatasliceId(scalar datasliceId)**
  Sets the connection point dataslice ID for the AE.
  - Parameters
    - **$datasliceId**
      (integer) The connection point dataslice ID.
  - Returns
    None.

  This function set the connection point dataslice ID. If using the built-in remote AE launching mechanism, the connection point info is automatically set from the process environment and this function does not need to be called.

- **setConnectionPointSessionId(scalar sessionId)**
  Sets the connection point session ID for the AE.
  - Parameters
    - **$sessionId**
      (integer) The connection point session ID.
  - Returns
    None.

  This function sets the connection point session ID. If using the built-in remote AE launching mechanism, the connection point info is automatically set from the process environment and this function does not need to be called.

- **setConnectionPointTransactionId(scalar transactionId)**
  Sets the connection point transaction ID for the AE.
  - Parameters
    - **$transactionId**
      (integer) The connection point transaction ID.
  - Returns
    None.

  This function sets the connection point transaction ID. If using the built-in remote AE launching mechanism, the connection point info is automatically set from the process environment and this function does not need to be called.

- **isLocal()**
  Determines if the AE that is currently running is a local AE.
  - Returns
    (boolean) A value of 1 if the AE is a Local AE; 0 otherwise.

- **isRemote()**
  Determines if the AE that is currently running is a remote AE.
  - Returns
    (boolean) A value of 1 if the AE is a Remote AE; 0 otherwise.

- **isAggregateAe()**
  Determines if the AE that is currently running is an aggregate AE.
  - Returns
    (boolean) A value of 1 if the AE is an Aggregate AE; 0 otherwise.

- **isFunctionAe()**
  Determines if the AE that is currently running is a function AE.
  - Returns
    (boolean) A value of 1 if the AE is a Function AE; 0 otherwise.

- **isShaperAe()**
  Determines if the AE that is currently running is a shaper/sizer AE.
  - Returns
    (boolean) A value of 1 if the AE is a Shaper/Sizer AE; 0 otherwise.

- **isUda()**
  Determines if the AE that is currently running is an aggregate AE.

- Returns
  (boolean) A value of 1 if the AE is an Aggregate AE; 0 otherwise.

- **isUdf()**
  Determines if the AE that is currently running is a UDF-based AE.
  - Returns
    (boolean) A value of 1 if the AE is a UDF based AE; 0 otherwise.

- **isUdfSizer()**
  Determines if the AE that is currently running is a UDF sizer AE.
  - Returns
    (boolean) A value of 1 if the AE is a UDF sizer AE; 0 otherwise.

- **isUdtf()**
  Determines if the AE that is currently running is a UDTF-based AE.
  - Returns
    (boolean) A value of 1 if the AE is a UDTF based AE; 0 otherwise.

- **isUdtfShaper()**
  Determines if the AE that is currently running is a UDTF shaper AE.
  - Returns
    (boolean) A value of 1 if the AE is a UDTF shaper AE; 0 otherwise.

- **didClassRun()**
  Determines if the class was run.
  - Returns
    (boolean) A value of 1 if the class was run; 0 otherwise.

- **run()**
  Runs an AE.
  - Returns
    None.

  This function determines whether the AE has already been run. If so, it prints an error and exits. If the AE has not run, the function determines whether the AE is remote or local and calls the appropriate run method. For local AEs, when a request is made, this function calls runLocal on the AE Class. For remote AEs, when a request is made, it calls runRemote on the AE class.

- **pingNps()**
  Notifies the Netezza software that work is still being performed and the AE should not be terminated.

- Returns
  None.

This function should be called when there are long periods of inactivity with the AE system to notify the NPS that the AE is still working.

- **runLocal()**
  This function can be overridden to control running of Local AEs.
  - Returns
    None.

  This function, by default, instantiates the AE and then calls runInstance . The capability to override this function is provided for advanced users, since typically an AE writer does not override the function.

- **runRemote()**
  This function can be overridden to control running of Remote AEs.
  - Returns
    None.

  This function, by default, creates an AE remote listener, accepts requests from the Netezza software, forks or runs single threaded as appropriate, creates an instance of the AE, and calls _runInstance(). It also handles exceptions appropriately. The capability to override this function is provided for advanced users since typically an AE writer does not override the function.

- **_doRemoteRunWork(scalar aeInfo)**
  This function can be overridden to control how remote AEs are run during single-threaded operation.
  - Parameters
    - **$aeInfo**
      The object.
  - Returns
    None.

  This function, by default, sets the AeInfo for single-threaded operation and calls runInstance . The function is called from runRemote . The capability to override this function is provided for advanced users, since typically an AE writer does not override the function.

- **_onIdle()**
  This function can be overridden to handle idle time for remote AEs.
  - Returns
    None.

- **runInstance(scalar dontexit)**
  This function can be overridden to control generic running of an AE instance.
  - Parameters
    - **$dontexit**
      Optional. A boolean value to prevent exit on completion, where 1 causes the instance to exit and 0 does not.
  - Returns
    None.

  This function, by default calls _setup , _run , _cleanUp , done() and close() for a specified AE instance, and handles exceptions appropriately. This capability is provided for advanced users, since typically an AE writer does not override the function.

- **_setup()**
  This function can be overridden for one-time setup of the AE instance.
  - Returns
    None.

  This function is called by runInstance immediately after creating the instance of the AE. It is possible to override the new function of the AE to perform initialization, however, it is preferable to perform custom initialization in _setup to guarantee appropriate setup of the underlying Perl AE system before performing custom initialization. This provides better exception handling during custom initialization.

- **_run()**
  This function can be overridden to handle generic running of AEs.
  - Returns
    None.

  This function is called for all AEs by default. Typically, an AE writer should import the Ae class and override runUdtf , runUdf, runUda , runShaper , or runSizer as appropriate. This capability is provided for advanced users, since typically an AE writer does not override the function.

- **runUdtf()**
  This function can be overridden to handle running a UDTF.
  - Returns
    None.

  This function, by default, fetches input rows and calls _getFunctionResult(), then outputs the result of the _getFunctionResult() function. The AE writer can override _getFunctionResult() to output more or less than one line of output.

- **runUdf()**
  This function may be overridden to handle running a UDF.
  - Returns
    None.

This function, by default, iterates through the rows and sends data to getFunctionResults(), then outputs the result of the _getFunctionResult() function. An AE writer can override _get-FunctionResult() for a simpler interface to UDF functionality.

- **runShaper()**
  This function must be overridden to handle running a Shaper.
  - Returns
    None.

  This function must be overridden to implement shaper functionality.

- **runSizer()**
  This function must be overridden to run Sizer function.
  - Returns
    None.

  This function must be overridden to implement sizer functionality.

- **_cleanUp()**
  This function may be overridden for one-time clean-up of the AE.
  - Returns
    None.

- **iter()**
  This function may be overridden to iterate over input rows, call getFunctionResult and send to output.
  - Returns
    None.

  This function, by default, iterates over the input rows, calls getFunctionResult and passes its results to the output function. An AE writer can override this function for a simpler interface to UDF functionality.

- **_initializeResult()**
  This function must be overridden to initialize state for an aggregate function.
  - Returns
    None.

- **_accumulate()**
  This function must be overridden to run aggregation functionality over values in the same dataslice.

- Returns
  None.

- **_merge()**
  This function must be overridden to merge states gathered from various dataslices.
  - Returns
    None.

- **_finalResult()**
  This function must be overridden to set and return the final result for an aggregate function.
  - Returns
    None.

- **runUda()**
  This function runs the user-defined aggregate function.
  - Returns
    None.

  This function, by default, gets the aggregation type and depending on the type calls the appropriate function: initialize, accumulate, merge or finalize. This can be overridden by the user for more granular control of the aggregate function operation.

- **_handleRemoteProtocol(scalar code, scalar data)**
  This function may be overridden to handle generic remote protocol functionality.
  - Parameters
    - **$code**
      (integer) The remote protocol code identifying the remote protocol request.
    - **$data**
      (string) The remote protocol data.
  - Returns
    (tuple<integer, string>) A tuple of the output code, where 0 means as size of zero K, and the return data.

  This function is not typically overridden. It is more typical to override either _getRemoteProtocolStatus(), _handleRemoteProtocolControlData , _handleRemoteProtocolRequest , or_handleRemoteProtocolStopRequest(). However, this capability is provided for the advanced AE writer.

- **_handleRemoteProtocolControlData(scalar data)**
  This function may be overridden to handle generic remote protocol control data.
  - Parameters
    - **$data**
      (string) The data received from the remote protocol subsystem.

- Returns
  (string) The data to be sent to the remote protocol subsystem.

- **_handleRemoteProtocolStopCommand()**
  This function may be overridden to handle the remote protocol stop command in a non-standard way.
  - Returns
    None.

  This function, by default, calls the DESTROY function.

- **_handleRemoteProtocolRequest(scalar request)**
  This function may be overridden to handle generic remote protocol requests.
  - Parameters
    - **$request**
      (string) The request received from the remote protocol subsystem.
  - Returns
    (string) The data to be sent to the remote protocol subsystem.

- **_handleException(scalar exception)**
  Handles exceptions by writing to the logfile and sending the user the error, if needed.
  - Parameters
    - **$exception**
      Optional. The exception string.
  - Returns
    None.

- **output(list args)**
  Sets the output of the AE.
  - Parameters
    - **@args**
      The list of output arguments.
  - Returns
    None.

  This function, by default, accepts one or more arguments and sets the output value or values using these arguments.

- **setOutputWithList(list theList)**
  Sets the output value using a list.

- Parameters
  - **@theList**
    The list of output arguments.
- Returns
  None.

This function, given a list of values, sets the output value using those values. It is called in output .

- **setOutputValue(scalar columnIndex, scalar value)**
  Checks the output value type using a library function and calls the appropriate set output function.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      Optional. The value to be set.
  - Returns
    A Boolean value indicating whether value was set or not. A value of 1 indicates the value was set; otherwise 0.

  This function, given a value, sets the output value. It is called in output .

- **setOutputNull(scalar columnIndex)**
  Sets the value at column specified by column index to null.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    None.

- **setOutputString(scalar columnIndex, scalar value, scalar length)**
  Sets the value at column specified by column index to a string value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The string value to be set.
    - **$length**
      The length of the string value to be set.
  - Returns
    None.

- **setOutputInt32(scalar columnIndex, scalar value)**

Sets the value at the column specified by the column index to an Int 32 value.

- Parameters
  - **$columnIndex**
    The column index.
  - **$value**
    The Int32 value to be set.
- Returns
  None.

- **setOutputInt64(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to an Int 64 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int64 value to be set.
  - Returns
    None.

- **setOutputInt8(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to an Int 8 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int8 value to be set.
  - Returns
    None.

- **setOutputInt16(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to an Int 16 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int16 value to be set.
  - Returns
    None.

- **setOutputFloat(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to a float value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The float value to be set.
  - Returns
    None.

- **setOutputDouble(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to a double value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The double value to be set.
  - Returns
    None.

- **setOutputBool(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to a boolean value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The double value to be set.
  - Returns
    None.

- **setOutputUdsData(scalar columnIndex, scalar value)**
  Sets the value at the column specified by the column index to a generic value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The generic value to be set.
  - Returns
    None.

- **getInputRow()**
  Gets a single input row as an array.
  - Returns
    The array containing the row elements.

  This function iterates over an input row and adds the value of each element in that row to an array.

- **getInputValue(scalar columnIndex)**
  Looks up the relevant element in the row.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    A Boolean value indicating whether the value was successfully retrieved. A value of 1 indicates success; otherwise 0.

  This function, given a column index, looks up the relevant element in the row and returns each element in that row to an array.

- **getInputStringLength(scalar columnIndex)**
  Gets the string length value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    int value.

- **getInputString(scalar columnIndex)**
  Gets the string value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    String value.

- **getInputInt32(scalar columnIndex)**
  Gets the Int32 value at column specified by column index.
  - Parameters

- **$columnIndex**
  The column index.

- Returns
  Int32 value.

- **getInputInt64(scalar columnIndex)**
  Gets the Int64 value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    Int64 value.

- **getInputInt8(scalar columnIndex)**
  Gets the Int8 value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    Int8 value.

- **getInputInt16(scalar columnIndex)**
  Gets the Int 16 value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    Int16 value.

- **getInputFloat(scalar columnIndex)**
  Gets the float value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    Float value.

- **getInputDouble(scalar columnIndex)**
  Gets the double value at column specified by column index.
  - Parameters

- **$columnIndex**
  The column index.

- Returns
  Double value.

- **getInputUdsData(scalar columnIndex)**
  Gets the generic value at column specified by column index.

  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    Generic Udsdata value.

- **getNext()**
  Loads the next input row into memory.

  - Returns
    None.

- **getState(scalar columnIndex)**
  Gets the state for a given column index or all indices.

  - Parameters
    - **$columnIndex**
      Optional. The column index.
  - Returns
    An array containing state value elements or a single state value.

  This function determines if the column index has been passed in to the function. If so it retrieves the specific value, otherwise it iterates over all state columns and returns an array containing all state values.

- **getInputState(scalar columnIndex)**
  Gets the input state for a given column index or all indices.

  - Parameters
    - **$columnIndex**
      Optional. The column index.
  - Returns
    An array containing input state value elements or a single state value.

  This function determines if the column index has been passed in to the function. If so it retrieves the specific input state value, otherwise it iterates over all state columns and returns an array containing all input state values. In this instance ''state'' is the state to merge.

- **_setState(list indexOrState, list inState)**
  Sets a single aggregate value or a list of aggregate values.
  - Parameters
    - **@indexOrState**
      Either a variable indicating the state index to be set or a list of states. If the value of the inState parameter has elements, then the first element of this array corresponds to the state that is to be set. If the second parameter does not contain any elements, this indicates a list of state values to be set.
    - **@inState**
      The state of a specific index.
  - Returns
    None.

  This function sets a single aggregate value or a list of aggregate values.

- **getStateValue(scalar index, scalar getMergedState)**
  Gets value of a state variable.
  - Parameters
    - **$index**
      The column index of the state variable.
    - **$getMergedState**
      Optional. If set to 1 the value being returned is the merged state. If set to 0 or undefined the value returned is the state to be merged.
  - Returns
    The state value.

- **getStateString(scalar columnIndex, scalar getInputState)**
  Gets the string state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates state value and 1 indicates input value.
  - Returns
    String value.

- **getStateInt32(scalar columnIndex, scalar getInputState)**
  Gets the Int32 state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**

The column index.

- **$getInputState**
  Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.

- Returns
  Int32 value.

- **getStateInt64(scalar columnIndex, scalar getInputState)**
  Gets the Int64 state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Int64 value.

- **getStateInt8(scalar columnIndex, scalar getInputState)**
  Gets the Int8 state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Int8 value.

- **getStateInt16(scalar columnIndex, scalar getInputState)**
  Gets the Int16 state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Int16 value.

- **getStateFloat(scalar columnIndex, scalar getInputState)**
  Gets the float state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Float value.

- **getStateDouble(scalar columnIndex, scalar getInputState)**
  Gets the double state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Double value.

- **getStateUdsData(scalar columnIndex, scalar getInputState)**
  Gets the generic state variable value at column specified by column index.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$getInputState**
      Indicates whether value to be obtained is state or input value. 0 indicates merged state value and 1 indicates state to be merged value.
  - Returns
    Generic Udsdata value.

- **_setAggregateResult(list result, scalar forFinalResult)**
  Sets the aggregate result value.
  - Parameters
    - **@result**
      The result value or array.
    - **$forFinalResult**
      A Boolean value indicating whether the result is final or not. A value of 1 indicates a final value;

otherwise 0.

- Returns
  None.

This function gets the value resulting from an aggregation operation and, using that result, sets the aggregate value.

- **_setAggregateValue(scalar index, scalar value, scalar forFinalResult)**
  Sets the return value for an aggregate.
  - Parameters
    - **$index**
      The index of the value parameter.
    - **$value**
      The value to be set.
    - **$forFinalResult**
      A Boolean value indicating whether the result is final. A value of 1 indicates a final value; otherwise 0.
  - Returns
    A Boolean value indicating whether the result is final. A value of 1 indicates a final value; otherwise 0.

  This function sets the return value for an aggregate. For the _initializeState(), _accumulate , and _merge functions, this value is the resulting state. For _finalResult the value is the result of the aggregate.

- **setAggregateNull(scalar columnIndex)**
  Sets the aggregate or state value at column specified by column index to null.
  - Parameters
    - **$columnIndex**
      The column index.
  - Returns
    None.

- **setAggregateString(scalar columnIndex, scalar value, scalar length)**
  Sets the aggregate or state value at column specified by column index to a string value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The string aggregate or state value to be set.
    - **$length**

The length of the string value to be set.

- Returns
  None.

- **setAggregateInt32(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to an Int 32 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int 32 aggregate or state value to be set.
  - Returns
    None.

- **setAggregateInt64(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to an Int 64 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int 64 aggregate or state value to be set.
  - Returns
    None.

- **setAggregateInt8(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to an Int 8 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The Int 8 aggregate or state value to be set.
  - Returns
    None.

- **setAggregateInt16(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to an Int 16 value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**

The Int 16 aggregate or state value to be set.

- Returns
  None.

- **setAggregateFloat(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to a float value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The float aggregate or state value to be set.
  - Returns
    None.

- **setAggregateDouble(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to a double value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The double aggregate or state value to be set.
  - Returns
    None.

- **setAggregateUdsData(scalar columnIndex, scalar value)**
  Sets the aggregate or state value at the column specified by the column index to a generic value.
  - Parameters
    - **$columnIndex**
      The column index.
    - **$value**
      The generic aggregate or state value to be set.
  - Returns
    None.

- **getNumberOfStateColumns()**
  Gets the number of state columns.

- Returns
  (integer) The number of state columns.

- **getStatePrecision(scalar columnIndex)**
  Gets the precision of a field of state at the specified column index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the state column.
  - Returns
    (integer) The precision of the column.

- **getStateScale(scalar columnIndex)**
  Gets the scale of a field of state at the specified column index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the state column.
  - Returns
    (integer) The scale of the column.

- **getStateSize(scalar columnIndex)**
  Gets the size of a field of state at the specified column index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the state column.
  - Returns
    (integer) The size of the column.

- **getStateType(scalar columnIndex)**
  Gets the data type of state field at specified index.
  - Parameters
    - **$columnIndex**
      (integer) The index of the state column.
  - Returns
    (DATA_TYPE) The data type of the state column at the specified column index.

- **getEnvironment()**
  Returns a hash containing the entire environment.
  - Returns
    (hash<string, string>) Name value pair of AE environment variable and its value.

- **getEnvironmentVariable(scalar variableName, list defaultValue)**
  Gets the value of an AE environment variable.
  - Parameters
    - **$variableName**
      (string) The name of the environment variable to fetch.
    - **@defaultValue**
      (object) The result to be returned if the entry is not found.
  - Returns
    The environment value.

  This function returns the environment value for variableName. If no default value is specified, and the variable is not in the environment, an exception is thrown.

- **getSharedLibraryPath(scalar libraryName, scalar caseSensitive)**
  Gets the path to a shared library.
  - Parameters
    - **$libraryName**
      (string) The name of the shared library.
    - **$caseSensitive**
      (boolean) Optional. Specifies whether the search should be performed in a case-sensitive manner. The default value is 1.
  - Returns
    (string) The full path to the shared library.

  This function throws an AeSharedLibraryNotFoundException if the shared library is not found.

- **yieldSharedLibraries(scalar forProcess)**
  Returns a hash that yields a shared library name and the corresponding full path for each shared library entry.
  - Parameters
    - **$forProcess**
      (boolean) Optional. If 1, specifies the shared library names and full paths for the remote AE process. The default value is 0.
  - Returns
    (hash<string, string>) Each element in the hash yields a library name(as key) and the full path(as value) to that library.

AeEnvironmentVariableNotFoundException Class Reference

The exception class specifically used when an environment variable is not properly set or is not

present.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeEnvironmentVariableNotFoundException error string.
- isFatal()
  Returns the AeEnvironmentVariableNotFoundException fatal value 1.

Detailed Description

The exception class specifically used when an environment variable is not properly set or is not present.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeEnvironmentVariableNotFoundException object.

- **getErrorString()**
  Returns the AeEnvironmentVariableNotFoundException error string.
  - Returns
    (String) The error string of AeEnvironmentVariableNotFoundException .

- **isFatal()**
  Returns the AeEnvironmentVariableNotFoundException fatal value 1.
  - Returns
    (Boolean) AeEnvironmentVariableNotFoundException is a fatal exception. Hence, this function returns 1.

AeEnvironmentVariableNotFoundWarning Class Reference

The exception class specifically used for warnings when an environment variable is not properly set or is not present.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeEnvironmentVariableNotFoundWarning error string.
- isFatal()
  Returns the AeEnvironmentVariableNotFoundWarning fatal value 0.

Detailed Description

The exception class specifically used for warnings when an environment variable is not properly set or is not present.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string
  - Returns
    (object) The new class object.

  This function creates an AeEnvironmentVariableNotFoundWarning object.

- **getErrorString()**
  Returns the AeEnvironmentVariableNotFoundWarning error string.
  - Returns
    (String) The error string of AeEnvironmentVariableNotFoundWarning .

- **isFatal()**
  Returns the AeEnvironmentVariableNotFoundWarning fatal value 0.

- Returns
  (Boolean) AeEnvironmentVariableNotFoundWarning is a non fatal warning. Hence, this function returns 0.

AeException Class Reference

The base Perl AE Exception class.

Constructor Member Functions

- new()
  The object constructor.

General Member Functions

- getType()
  Returns the exception type string.
- getErrStr()
  Returns the exception error string.
- getIsFatal()
  Returns the exception fatal value.

Detailed Description

The base Perl AE Exception class.

This class is the base Perl AE Exception class.

Public Member Function Documentation

- **new()**
  The object constructor.
  - Returns
    (object) The new class object.

  This function creates an AeException object.

- **getType()**
  Returns the exception type string.
  - Returns
    (String) The type string of the exception.

  This function calls the getType method of the calling class and returns the value to the user.

- **getErrStr()**

Returns the exception error string.

▲ Returns
(String) The error string of the exception.

This function calls the getErrorString() method of the calling class and returns the value to the user.

► **getIsFatal()**
Returns the exception fatal value.

▲ Returns
(Boolean) The fatal value of the exception. 1 indicates the exception is fatal and 0 indicates the exception is non fatal.

This function calls the isFatal() method of the calling class and returns the value to the user.

AeInfo Class Reference

A class storing information about the AE.

Constructor and Destructor Member Functions

► new(scalar class)
The object constructor.

► DESTROY()
The object destructor.

General Member Functions

► isLocal()
Determines if the AE is a local AE.

► isAggregate()
Determines if the AE is an aggregate AE.

► isFunction()
Determines if the AE is a function AE.

► isShaper()
Determines if the AE is a shaper/sizer AE.

Detailed Description

A class storing information about the AE.

The class generally contains basic information about a Remote AE.

Public Member Function Documentation

- **new(scalar class)**
  The object constructor.
  - Parameters
    - **$class**
      The package name.
  - Returns
    (object) The new class object.

  This function calls the _AeInfo constructor.

- **DESTROY()**
  The object destructor.
  - Returns
    None.

  This function is a finalization method that cleans up the class state when there are no more references to an object or the program exits.

- **isLocal()**
  Determines if the AE is a local AE.
  - Returns
    (boolean) Returns 1 if the AE is a Local AE; 0 otherwise.

- **isAggregate()**
  Determines if the AE is an aggregate AE.
  - Returns
    (boolean) Returns 1 if the AE is an Aggregate AE; 0 otherwise.

- **isFunction()**
  Determines if the AE is a function AE.
  - Returns
    (boolean) Returns 1 if the AE is a Function AE; 0 otherwise.

- **isShaper()**
  Determines if the AE is a shaper/sizer AE.
  - Returns
    (boolean) Returns 1 if the AE is a Shaper/Sizer AE; 0 otherwise.

AeInitialiationFailedException Class Reference

An exception class specific for failures that occur during the initialization of the AE.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInitialiationFailedException error string.
- isFatal()
  Returns the AeInitialiationFailedException fatal value 1.

Detailed Description

An exception class specific for failures that occur during the initialization of the AE.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInitialiationFailedException object.

- **getErrorString()**
  Returns the AeInitialiationFailedException error string.
  - Returns
    (String) The AeInitialiationFailedException's error string.

- **isFatal()**
  Returns the AeInitialiationFailedException fatal value 1.
  - Returns
    (Boolean) AeInitialiationFailedException is a fatal exception. Hence, this function returns 1.

AeInitialiationFailedWarning Class Reference

The exception class specifically used for warnings that occur during initialization of the AE.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInitialiationFailedWarning error string.
- isFatal()
  Returns the AeInitialiationFailedWarning fatal value 0.

Detailed Description

The exception class specifically used for warnings that occur during initialization of the AE.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInitialiationFailedWarning object.

- **getErrorString()**
  Returns the AeInitialiationFailedWarning error string.
  - Returns
    (String) The error string of AeInitialiationFailedWarning .

- **isFatal()**
  Returns the AeInitialiationFailedWarning fatal value 0.
  - Returns
    (Boolean) AeInitialiationFailedWarning is a non fatal warning. Hence, this function returns 0.

AeInternalError Class Reference

The AeInternalError class is used for more generic internal causes.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInternalError error string.
- isFatal()
  Returns the AeInternalError fatal value 1.

Detailed Description

The AeInternalError class is used for more generic internal causes.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInternalError object.

- **getErrorString()**
  Returns the AeInternalError error string.
  - Returns
    (String) The error string of AeInternalError .

- **isFatal()**
  Returns the AeInternalError fatal value 1.

- Returns
  (Boolean) AeInternalError is a fatal exception. Hence, this function returns 1.

AeInternalWarning Class Reference

The exception class used for more generic warnings due to internal causes.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInternalWarning error string.
- isFatal()
  Returns the AeInternalWarning fatal value 0.

Detailed Description

The exception class used for more generic warnings due to internal causes.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInternalWarning object.

- **getErrorString()**
  Returns the AeInternalWarning error string.
  - Returns
    (String) The error string of AeInternalWarning .

- **isFatal()**
  Returns the AeInternalWarning fatal value 0.

- Returns
  (Boolean) AeInternalWarning is a non fatal warning. Hence, this function returns 0.

AeInvalidStateException Class Reference

The AeInvalidStateException class is specific for failures caused by an invalid state.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInvalidStateException error string.
- isFatal()
  Returns the AeInvalidStateException fatal value 1.

Detailed Description

The AeInvalidStateException class is specific for failures caused by an invalid state.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInvalidStateException object.

- **getErrorString()**
  Returns the AeInvalidStateException error string.
  - Returns
    (String) The error string of AeInvalidStateException .

- **isFatal()**
  Returns the AeInvalidStateException fatal value 1.
  - Returns
    (Boolean) AeInvalidStateException is a fatal exception. Hence, this function returns 1.

AeInvalidStateWarning Class Reference

The exception class specifically used for warnings caused by an invalid state.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeInvalidStateWarning error string.
- isFatal()
  Returns the AeInvalidStateWarning fatal value 0.

Detailed Description

The exception class specifically used for warnings caused by an invalid state.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeInvalidStateWarning object.

- **getErrorString()**
  Returns the AeInvalidStateWarning error string.
  - Returns
    (String) The error string of AeInvalidStateWarning .

- **isFatal()**
  Returns the AeInvalidStateWarning fatal value 0.
  - Returns
    (Boolean) AeInvalidStateWarning is a non fatal warning. Hence, this function returns 0.

AeSharedLibraryNotFoundException Class Reference

The exception class specifically used when a shared library is not properly linked or is not present.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeSharedLibraryNotFoundException error string.
- isFatal()
  Returns the AeSharedLibraryNotFoundException fatal value 1.

Detailed Description

The exception class specifically used when a shared library is not properly linked or is not present.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeSharedLibraryNotFoundException object.

- **getErrorString()**
  Returns the AeSharedLibraryNotFoundException error string.
  - Returns
    (String) The error string of AeSharedLibraryNotFoundException .

- **isFatal()**
  Returns the AeSharedLibraryNotFoundException fatal value 1.
  - Returns
    (Boolean) AeSharedLibraryNotFoundException is a fatal exception. Hence, this function returns 1.

AeSharedLibraryNotFoundWarning Class Reference

The exception class specifically used for warning when a shared library is not properly linked or is not present.

Constructor Member Functions

- new(scalar errStr)
  The object constructor.

General Member Functions

- getErrorString()
  Returns the AeSharedLibraryNotFoundWarning error string.
- isFatal()
  Returns the AeSharedLibraryNotFoundWarning fatal value 0.

Detailed Description

The exception class specifically used for warning when a shared library is not properly linked or is not present.

This class inherits from AeException .

Public Member Function Documentation

- **new(scalar errStr)**
  The object constructor.
  - Parameters
    - **$errStr**
      Optional. The exception string.
  - Returns
    (object) The new class object.

  This function creates an AeSharedLibraryNotFoundWarning object.

- **getErrorString()**
  Returns the AeSharedLibraryNotFoundWarning error string.

- Returns
  (String) The error string of AeSharedLibraryNotFoundWarning .

- **isFatal()**
  Returns the AeSharedLibraryNotFoundWarning fatal value 0.
  - Returns
    (Boolean) AeSharedLibraryNotFoundWarning is a non fatal warning. Hence, this function returns 0.

Debug Class Reference

A Perl extension for debugging and logging in the INZA Perl AE.

Constructor Member Functions

- new(scalar class, hash args)
  The object constructor.

General Member Functions

- getExceptionInfo(scalar exception)
  Gets a user-readable version of an exception.
- getExceptionUserString(scalar exception)
  Gets a user-readable version of an exception.
- getStack(scalar exception)
  Gets a user-readable version of an exception.
- yieldExceptionInfo(scalar exception)
  Converts a nzae exception into a readable stack trace.
- printStack(scalar exception)
  Converts and prints a nzae exception.

Detailed Description

A Perl extension for debugging and logging in the INZA Perl AE.

Public Member Function Documentation

- **new(scalar class, hash args)**
  The object constructor.
  - Parameters
    - **$class**

The package name.

- **%args**
  A hash of optional arguments.

- Returns
  (object) The new class object.

This function creates a new Debug object.

- **getExceptionInfo(scalar exception)**
  Gets a user-readable version of an exception.

  - Parameters
    - **$exception**
      The exception object.

  - Returns
    A user-parseable exception stack.

  This function is included to be consistent with other AE API implementations. While other language APIs include specific methods for returning the exception information, user string and stack information for the exception, Perl's exception handling is not as granular. Like getExceptionUserString and getStack, this function calls yieldExceptionInfo to get a cleaned stack trace. As a result, all three methods return the same information.

- **getExceptionUserString(scalar exception)**
  Gets a user-readable version of an exception.

  - Parameters
    - **$exception**
      The exception object.

  - Returns
    A user-parseable exception stack.

  This function is included to be consistent with other AE API implementations. While other language APIs include specific methods for returning the exception information, user string and stack information for the exception, Perl's exception handling is not as granular. Like getExceptionInfo and getStack, this function calls yieldExceptionInfo to get a cleaned stack trace. As a result, all three methods return the same information.

- **getStack(scalar exception)**
  Gets a user-readable version of an exception.

  - Parameters
    - **$exception**
      The exception object.

  - Returns
    A user-parseable exception stack.

  This function is included to be consistent with other AE API implementations. While other language APIs

include specific methods for returning the exception information, user string and stack information for the exception, Perl's exception handling is not as granular. Like getExceptionInfo and getExceptionUserString, this function calls yieldExceptionInfo to get a cleaned stack trace. As a result, all three methods return the same information.

- **yieldExceptionInfo(scalar exception)**
  Converts a nzae exception into a readable stack trace.
  - Parameters
    - **$exception**
      The exception object.
  - Returns
    A user-parseable exception stack.

  This function returns a stack trace appropriate for logging or sending to the user.

- **printStack(scalar exception)**
  Converts and prints a nzae exception.
  - Parameters
    - **$exception**
      The exception object.
  - Returns
    None.

  This function processes a stack trace into an appropriate form and prints to STDOUT.

Exceptions Class Reference

The interface class for exceptions to inherit from.

Constructor Member Functions

- new()
  The object constructor.

Detailed Description

The interface class for exceptions to inherit from.

The parent class from which AeException inherits.

Public Member Function Documentation

- **new()**
  The object constructor.
  - Returns
    (object) The new class object.

  This function creates an exceptions object.

RemoteListener Class Reference

Perl extension for the Remote AE API.

Constructor Member Functions

- new(scalar class)
  The object constructor.

Destructor Member Functions

- DESTROY()
  The object destructor.

General Member Functions

- init(scalar connectionPointName, scalar datasliceId, scalar sessionId, scalar transactionIdStr)
  Initializes the RemoteListener .
- remoteCallback()
  Requests are made to this function by the AE remote protocol layer.
- setRemoteProtocolCallbackObject(scalar callbackRecipient)
  Sets the recipient of the remote protocol callbacks.

Detailed Description

Perl extension for the Remote AE API.

This class, along with the AE class, provides an interface that allows the user to write Remote AEs in Perl.

Public Member Function Documentation

- **new(scalar class)**
  The object constructor.
  - Parameters
    - **$class**
      The package name.
  - Returns
    (object) A new class object.

  This function calls the _RemoteListenerInternal constructor.

- **DESTROY()**
  The object destructor.
  - Returns
    None.

  This function is a finalization method that cleans up the class state when there are no more references to an object or the program exits.

- **init(scalar connectionPointName, scalar datasliceId, scalar sessionId, scalar transactionId-Str)**
  Initializes the RemoteListener .
  - Parameters
    - **$connectionPointName**
      The connection name.
    - **$datasliceId**
      The Dataslice ID on the SPU.
    - **$sessionId**
      The Session ID on the SPU.
    - **$transactionIdStr**
      Optional. The Transaction ID on the SPU.
  - Returns
    None.

- **remoteCallback()**
  Requests are made to this function by the AE remote protocol layer.
  - Returns
    None.

- **setRemoteProtocolCallbackObject(scalar callbackRecipient)**
  Sets the recipient of the remote protocol callbacks.
  - Parameters
    - **$callbackRecipient**
      An AE object.
  - Returns
    None.

Stack Class Reference

Perl extension for generating a stack trace for log output.

Constructor Member Functions

- new(scalar class, hash args)
  The object constructor.

General Member Functions

- getStackString(scalar exception, scalar exceptionType, scalar shortenFilePaths)
  Converts a nzae exception into a readable stack trace.

Detailed Description

Perl extension for generating a stack trace for log output.

Public Member Function Documentation

- **new(scalar class, hash args)**
  The object constructor.
  - Parameters
    - **$class**
      The package name.
    - **%args**
      Optional. A list of arguments.
  - Returns
    (object) The new class object.

- **getStackString(scalar exception, scalar exceptionType, scalar shortenFilePaths)**
  Converts a nzae exception into a readable stack trace.
  - Parameters
    - **$exception**
      The exception object
    - **$exceptionType**
      Optional. The exception type.
    - **$shortenFilePaths**
      Optional. If TRUE, the stack trace reports the location of the file causing the exception with the common path elements removed.
  - Returns
    A user-parseable exception stack.

  This function creates a stack trace appropriate for logging or sending to the user.

Notices and Trademarks

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
26 Forest Street
Marlborough, MA 01752 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement

or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

Trademarks

IBM, the IBM logo, ibm.com and Netezza are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™),these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

NEC is a registered trademark of NEC Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and/or other countries.

D-CC, D-C++, Diab+, FastJ, pSOS+, SingleStep, Tornado, VxWorks, Wind River, and the Wind River logo are trademarks, registered trademarks, or service marks of Wind River Systems, Inc. Tornado patent pending.

APC and the APC logo are trademarks or registered trademarks of American Power Conversion Corporation.

Other company, product or service names may be trademarks or service marks of others.

Regulatory and Compliance

Regulatory Notices

Install the NPS system in a restricted-access location. Ensure that only those trained to operate or service the equipment have physical access to it. Install each AC power outlet near the NPS rack that plugs into it, and keep it freely accessible. Provide approved 30A circuit breakers on all power sources.

Product may be powered by redundant power sources. Disconnect ALL power sources before servicing. High leakage current. Earth connection essential before connecting supply. Courant de fuite élevé. Raccordement à la terre indispensable avant le raccordement au réseau.

Homologation Statement

This product may not be certified in your country for connection by any means whatsoever to interfaces of public telecommunications networks. Further certification may be required by law prior to making any such connection. Contact an IBM representative or reseller for any questions.

FCC - Industry Canada Statement

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio-frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case users will be required to correct the interference at their own expense.

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

CE Statement (Europe)

This product complies with the European Low Voltage Directive 73/23/EEC and EMC Directive 89/336/EEC as amended by European Directive 93/68/EEC.

Warning: This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

VCCI Statement

この装置は、情報処理装置等電波障害自主規制協議会 (VCCI) の基準に基づくクラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起越すことがあります。この場合には使用者が適切な対策を講ずるう要求されることがあります。

Index

Symbols

A

C

D

E

G

I

L

N

O

P

R

S

W

Y