IBM® Netezza® Analytics
Release 3.3.5.0

Python Analytic Executables API Reference

Part Number 00X6637-00 Rev. 1

Note: Before using this information and the product that it supports, read the information in "Notices and Trademarks" on page 40.

Contents

Preface

1 Class Documentation

Notices and Trademarks

Index

Preface

This guide provides an API reference for Python AE programmers.

Audience for This Guide

The *Python Analytic Executables API Reference* is written for programmers who intend to create Analytic Executables for IBM Netezza Analytics using the Python 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 Python AE API, which is a language adapter provided as part of IBM Netezza Analytics. The Python AE API provides programmatic access to the AE interface for Python 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

The primary AE interface.

General Member Functions

- __init__(self)
  The object constructor.
- didClassRun(Class)
  Determines if the class was run.
- getRequestHandlingStyle(Class)
  Gets the request handling style.
- run(Class)
  Runs the AE Class.
- close(self)
  Optional.
- done(self)
  Optional.
- isLocal(Class)
  Determines if this is a local AE.
- isRemote(Class)
  Determines if this is a remote AE.
- isAggregateAe(self)
  Determines if this is an aggregate AE.
- isFunctionAe(self)
  Determines if this is a function AE.

- isShaperAe(self)
  Determines if this is a shaper/sizer AE.
- isUda(self)
  Determines if this is an aggregate AE.
- isUdf(self)
  Determines if this is a UDF-based AE.
- isUdfSizer(self)
  Determines if this is a UDF sizer.
- isUdtf(self)
  Determines if this is a UDTF-based AE.
- isUdtfShaper(self)
  Determines if this is a UDTF shaper.
- pingNps(self)
  Call to notify the Netezza software that work is still being performed and the AE should not be terminated.

Remote AE Member Functions

- getConnectionPointName(Class)
  Returns the connection point name for this AE.
- getConnectionPointDatasliceId(Class)
  Returns the connection point dataslice ID for this AE.
- getConnectionPointSessionId(Class)
  Returns the connection point session ID for this AE.
- getConnectionPointTransactionId(Class)
  Returns the connection point transaction ID for this AE.
- setConnectionPointName(Class, connectionPointName)
  Sets the connection point name for this AE.
- setConnectionPointDatasliceId(Class, datasliceId)
  Sets the connection point dataslice ID for this AE.
- setConnectionPointSessionId(Class, sessionId)
  Sets the connection point session ID for this AE.
- setConnectionPointTransactionId(Class, transactionId)
  Sets the connection point transaction ID for this AE.

Metadata Member Functions

- describe(self)
  Returns a string representation of the metadata.
- getDataTypeName(self, dataType)
  Gets the name of a specified data type.

- getInputTypes(self)
  Gets a list of input data types.
- getNumberOfInputColumns(self)
  Gets the number of input columns.
- getNumberOfOutputColumns(self)
  Gets the number of output columns.
- getInputPrecision(self, columnIndex)
  Gets the precision of an input numeric.
- getInputScale(self, columnIndex)
  Gets the scale of an input numeric.
- getInputSize(self, columnIndex)
  Gets the size of an input string.
- getInputType(self, columnIndex)
  Gets the data type of an input.
- getOutputPrecision(self, columnIndex)
  Gets the precision of an output numeric.
- getOutputScale(self, columnIndex)
  Gets the scale of an output numeric.
- getOutputSize(self, columnIndex)
  Gets the size of an output string.
- getOutputType(self, columnIndex)
  Gets the data type of an output.
- getUdfReturnType(self)
  Gets the return data type for a UDF.
- isCalledWithOrderByClause(self)
  Determines if the function was called with an ORDER BY clause.
- isCalledWithOverClause(self)
  Determines if the function was called with an OVER clause.
- isCalledWithPartitionByClause(self)
  Determines if the function was called with a PARTITION BY clause.
- isDataInnerCorrelated(self)
  Determines if the function data is inner-correlated.
- isDataLeftCorrelated(self)
  Determines if the function data is left-correlated.
- isDataUncorrelated(self)
  Determines if the function data is uncorrelated.

Run-Time Member Functions

- getCurrentUsername(self)
  Gets the database username of the current AE being run.

- getDatasliceId(self)
  Determines the ID of the dataslice being serviced by the AE.
- getHardwareId(self)
  Determines the ID of the hardware on which the AE is running.
- getLogMask(self)
  Gets the log mask that the AE is running on.
- getNumberOfDataSlices(self)
  Gets the number of dataslices that are running on the Netezza appliance.
- getNumberOfSpus(self)
  Gets the number of SPUs running on the Netezza appliance.
- getSuggestedMemoryLimit(self)
  Gets the suggested memory limit for this AE.
- getSessionId(self)
  Gets the session ID associated with this AE.
- getTransactionId(self)
  Gets the transaction ID associated with this AE.
- isAUserQuery(self)
  Determines if the AE is handling a user query.
- isLoggingEnabled(self)
  Determines if logging is enabled.
- isRunningInPostgres(self)
  Determines if this AE is running in Postgres.
- isRunningInDbos(self)
  Determines if this AE is running in DBOS.
- isRunningOnSpu(self)
  Determines if this AE is running on the SPU.
- isRunningOnHost(self)
  Determines if this AE is running on the host.

Shared Library Member Functions

- getSharedLibraryPath(self, libraryName, caseSensitive=True)
  Gets the path to a shared library.
- yieldSharedLibraries(self, forProcess=False)
  Returns a generator that yields a shared library name and the corresponding full path for each shared library entry.

Environment Member Functions

- getEnvironment(self)
  Returns a dict containing the entire environment.

- getEnvironmentVariable(self, variableName, defaultValue)
  Gets the value of an entry in the environment.

Error Handling Member Functions

- userError(self, errorString)
  Ends the query and sends the user an error message.

Logging Member Functions

- getLogFilePath(self)
  Gets the log file path.
- log(self, text, logLevel=None)
  Log a message to the AE log file and optionally to stderr and/or the Python AE log.

Input Fetching Member Functions

- __iter__(self)
  Yield rows of data.
- getInputRow(self)
  Returns a list containing the elements of the current input row.
- getInputValue(self, columnIndex)
  Returns the value of the current input at a specified column index.
- getInputString(self, columnIndex)
  Returns the String value of the current input at a specified column index.
- getNext(self)
  Loads the next input row into memory.

Row Outputting Member Functions

- outputCurrentRow(self)
  Outputs the current row.
- outputInputColumn(self, inputColumnIndex, outputColumnIndex)
  Copies an input field into an output field.

Aggregate Member Functions

- getState(self, columnIndex=None)
  Gets a list containing the current state.
- getInputState(self, columnIndex=None)
  Gets a list containing the input state, that is, the state to merge.
- setState(self, indexOrState, state)
  Sets the current state.
- getNumberOfStateColumns(self)
  Gets the number of columns in the state.
- getStateDataType(self, columnIndex)

Gets the data type of a field of state at the specified column index.

- getStatePrecision(self, columnIndex)
  Gets the precision of a field of state at the specified column index.
- getStateScale(self, columnIndex)
  Gets the scale of a field of state at the specified column index.
- getStateSize(self, columnIndex)
  Gets the size of a field of state at the specified column index.

Shaper/Sizer Member Functions

- addOutputColumn(self, columnName, dataType)
  Add an output column.
- addOutputColumnNumeric(self, columnName, dataType, precision, scale)
  Add a numeric output column.
- addOutputColumnString(self, columnName, dataType, size)
  Add a string output column.
- isInputValueAvailable(self, columnIndex)
  Determines if an input value is available for a shaper/sizer.
- isShaperSystemCatalogUpperCase(self)
  Determines if the Netezza system catalog is upper case.
- isUdfSizer(self)
  Determines if the AE is a sizer for a UDF.
- isUdtfShaper(self)
  Determines if the AE is a shaper for a UDTF.

Static Public Attributes

- STRING_DATA_TYPES
  String Data Types.
- INTEGER_DATA_TYPES
  Integer Data Types.
- NUMERIC_DATA_TYPES
  Numeric Data Types.
- REQUEST_HANDLING_STYLE__USE_THREADS
  Use threads.
- REQUEST_HANDLING_STYLE__FORK
  Fork.
- REQUEST_HANDLING_STYLE__SINGLE_THREADED
  Single-threaded.
- DEFAULT_REQUEST_HANDLING_STYLE
  The default handling style defaults to use-threads.

- CONNECTION_POINT_NAME
- DATA_TYPE__UNKNOWN
  Unknown data type.
- DATA_TYPE__FIXED
  Fixed (CHAR) data type.
- DATA_TYPE__VARIABLE
  Variable (VARCHAR) data type.
- DATA_TYPE__NATIONAL_FIXED
  National Fixed data type.
- DATA_TYPE__NATIONAL_VARIABLE
  National Variable data type.
- DATA_TYPE__BOOLEAN
  Boolean data type.
- DATA_TYPE__DATE
  Date data type.
- DATA_TYPE__TIME
  Time data type.
- DATA_TYPE__TIMETZ
  Time Zone data type.
- DATA_TYPE__NUMERIC32
  Numeric 32 data type.
- DATA_TYPE__NUMERIC64
  Numeric 64 data type.
- DATA_TYPE__NUMERIC128
  Numeric 128 data type.
- DATA_TYPE__FLOAT
  Float data type.
- DATA_TYPE__DOUBLE
  Double data type.
- DATA_TYPE__INTERVAL
  Interval data type.
- DATA_TYPE__INT8
  Int 8-bit data type.
- DATA_TYPE__INT16
  Int 16-bit data type.
- DATA_TYPE__INT32
  Int 32-bit data type.
- DATA_TYPE__INT64
  Int 64-bit data type.
- DATA_TYPE__TIMESTAMP
  Timestamp data type.

- DATA_TYPE__GEOMETRY
  Geometry data type.
- DATA_TYPE__VARBINARY
  Varbinary data type.
- AGGREGATION_TYPE__ERROR
  Error state.
- AGGREGATION_TYPE__END
  End of aggregation state.
- AGGREGATION_TYPE__INITIALIZE
  Initialize state aggregation state.
- AGGREGATION_TYPE__ACCUMULATE
  Accumulate aggregation state.
- AGGREGATION_TYPE__MERGE
  Merge aggregation state.
- AGGREGATION_TYPE__FINAL_RESULT
  Final result aggregation state.
- LOG_LEVEL__TRACE
  Log level: Trace.
- LOG_LEVEL__DEBUG
  Log level: Debug.
- CONNECTION_POINT_NAME
  Override this member variable to set or override the connection point name.

Overridable Member Functions

- _accumulate(self, state, row)
  Override this function to handle the accumulate process state of aggregates.
- _cleanUp(self)
  Override this function for one-time cleanup of the AE.
- _finalResult(self, state)
  Override this function to handle the Finalize processing state for aggregates.
- _getFunctionResult(self, row)
  Override this to implement UDF or UDTF functionality where there is exactly one output per input.
- _getRemoteProtocolStatus(Class)
  Override this function to provide a different status string to the remote protocol status call-back.
- _handleRemoteProtocol(Class, code, data)
  Override this function to handle generic remote protocol functionality.
- _handleRemoteProtocolControlData(Class, data)

Override this function to handle generic remote protocol control data.

- _handleRemoteProtocolRequest(Class, request)
  Override this function to handle generic remote protocol requests.
- _handleRemoteProtocolStopCommand(Class)
  Override this function to handle the remote protocol stop command in a non-standard way.
- _initializeState(self)
  Override this function to handle the Initialize processing state for aggregates.
- _merge(self, state, inputState)
  Override this function to handle the Merge processing state for aggregates.
- _onIdle(Class)
  Override this function to handle idle time for remote AEs.
- _run(self)
  Override this function to handle generic running of AEs.
- _runInstance(Class, instance)
  Override this function to handle generic running of an AE instance.
- _runLocal(Class)
  Override this function to handle running only local AEs.
- _runRemote(Class)
  Override this function to handle how remote AEs are run.
- _runShaper(self)
  Override this function to handle running a shaper.
- _runSizer(self)
  Override this function to handle running a sizer.
- _runUda(self)
  Override this function to handle running an aggregate.
- _runUdf(self)
  Override this function to handle running a UDF.
- _runUdtf(self)
  Override this function to handle running a UDTF.
- _setup(self)
  Override this for one-time setup of the AE instance.

Detailed Description

The primary AE interface.

Public Member Function Documentation

- **__init__(self)**
  The object constructor.

  Calls the _AeInternal constructor; also tracks the number of instances of AEs that have been created.

- **didClassRun(Class)**
  Determines if the class was run.
  - Returns
    (boolean) TRUE if the class was run; FALSE otherwise.

- **getRequestHandlingStyle(Class)**
  Gets the request handling style.
  - Returns
    (REQUEST_HANDLING_STYLE) The request handling style.

  This function returns either REQUEST_HANDLING_STYLE__USE_THREADS, REQUEST_HANDLING_STYLE__FORK, or REQUEST_HANDLING_STYLE__SINGLE_THREADED based on the current request handling style.

- **run(Class)**
  Runs the AE Class.

  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.

- **close(self)**
  Optional.

  Called when no more calls are to be made to the Netezza software.

- **done(self)**
  Optional.

  Called when no more output is to be written to the Netezza software.

- **isLocal(Class)**
  Determines if this is a local AE.
  - Returns
    (boolean) Returns TRUE if this is a local AE; FALSE otherwise.

- **isRemote(Class)**
  Determines if this is a remote AE.
  - Returns
    (boolean) Returns TRUE if this is a remote AE; FALSE otherwise.

- **isAggregateAe(self)**
  Determines if this is an aggregate AE.
  - Returns
    (boolean) Returns TRUE if this is an aggregate AE; FALSE otherwise.

  This returns the same value as isUda .

- **isFunctionAe(self)**
  Determines if this is a function AE.
  - Returns
    (boolean) Returns TRUE if this is a function AE; FALSE otherwise.

- **isShaperAe(self)**
  Determines if this is a shaper/sizer AE.
  - Returns
    (boolean) Returns TRUE if this is a shaper/sizer AE; FALSE otherwise.

- **isUda(self)**
  Determines if this is an aggregate AE.
  - Returns
    (boolean) Returns TRUE if this is an aggregate AE; FALSE otherwise.

  This returns the same value as isAggregateAe .

- **isUdf(self)**
  Determines if this is a UDF-based AE.
  - Returns
    (boolean) Returns TRUE if this is a UDF-based AE; FALSE otherwise.

- **isUdfSizer(self)**
  Determines if this is a UDF sizer.
  - Returns
    (boolean) Returns TRUE if this is a UDF sizer AE; FALSE otherwise.

- **isUdtf(self)**
  Determines if this is a UDTF-based AE.
  - Returns
    (boolean) Returns TRUE if this is a UDTF-based AE; FALSE otherwise.

- **isUdtfShaper(self)**
  Determines if this is a UDTF shaper.

- Returns
  (boolean) Returns TRUE if this is a UDTF shaper AE; FALSE otherwise.

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

  When there are long periods of inactivity with the AE system, this function should be called to notify the Netezza software that the AE is still working.

- **getConnectionPointName(Class)**
  Returns the connection point name for this AE.

  - Returns
    (string) The connection point name.

- **getConnectionPointDatasliceId(Class)**
  Returns the connection point dataslice ID for this AE.

  - Returns
    (integer) The dataslice ID.

  The value returned is -1 if the AE is not running in remote mode by dataslice.

- **getConnectionPointSessionId(Class)**
  Returns the connection point session ID for this AE.

  - Returns
    (integer) The session ID.

  The value returned is -1 if the AE is not running in remote mode by session.

- **getConnectionPointTransactionId(Class)**
  Returns the connection point transaction ID for this AE.

  - Returns
    (integer) The transaction ID.

  The value returned is -1 if the AE is not running in remote mode by transaction.

- **setConnectionPointName(Class, connectionPointName)**
  Sets the connection point name for this AE.

  - Parameters
    - **connectionPointName**
      (string) The name of the connection point.

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 using this function, the deriving class may override the class variable, CONNECTION_POINT_NAME.

- **setConnectionPointDatasliceId(Class, datasliceId)**
  Sets the connection point dataslice ID for this AE.
  - Parameters
    - **datasliceId**
      (integer) 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(Class, sessionId)**
  Sets the connection point session ID for this AE.
  - Parameters
    - **sessionId**
      (integer) 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(Class, transactionId)**
  Sets the connection point transaction ID for this AE.
  - Parameters
    - **transactionId**
      (integer) The connection point transaction ID.

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

- **describe(self)**
  Returns a string representation of the metadata.
  - Returns
    (string) A string representation of the metadata.

- **getDataTypeName(self, dataType)**
  Gets the name of a specified data type.
  - Parameters
    - **dataType**
      (DATA_TYPE) The data type.
  - Returns
    (string) The name of the specified data type.

- **getInputTypes(self)**
  Gets a list of input data types.
  - Returns
    (list<DATA_TYPE>) The list of input data types.

- **getNumberOfInputColumns(self)**
  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(self)**
  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(self, 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(self, 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 'accumu-

late' processing state.

- **getInputSize(self, columnIndex)**
  Gets the size of an input string.
  - Parameters
    - **columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The size 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.

- **getInputType(self, columnIndex)**
  Gets the data type of an input.
  - Parameters
    - **columnIndex**
      (integer) The index of the column.
  - Returns
    (DATA_TYPE) The data type of the input 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(self, 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(self, 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(self, columnIndex)**
  Gets the size of an output string.
  - Parameters
    - **columnIndex**
      (integer) The index of the column.
  - Returns
    (integer) The size 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.

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

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

- **getUdfReturnType(self)**
  Gets the return data type for a UDF.
  - Returns
    (DATA_TYPE) The data type of the output field.

  This function is only valid for UDF sizer AEs.

- **isCalledWithOrderByClause(self)**
  Determines if the function was called with an ORDER BY clause.
  - Returns
    (boolean) TRUE if the function was called with an ORDER BY clause; FALSE otherwise.

  This function is only valid for function AEs.

- **isCalledWithOverClause(self)**
  Determines if the function was called with an OVER clause.
  - Returns
    (boolean) TRUE if the function was called with an OVER clause; FALSE otherwise.

  This function is only valid for function AEs.

- **isCalledWithPartitionByClause(self)**
  Determines if the function was called with a PARTITION BY clause.
  - Returns
    (boolean) TRUE if the function was called with a PARTITION BY clause; FALSE otherwise.

  This function is only valid for function AEs.

- **isDataInnerCorrelated(self)**
  Determines if the function data is inner-correlated.
  - Returns
    (boolean) TRUE if the function data is inner-correlated; FALSE otherwise.

  This function is only valid for function AEs.

- **isDataLeftCorrelated(self)**
  Determines if the function data is left-correlated.
  - Returns
    (boolean) TRUE if the function data is left-correlated; FALSE otherwise.

  This function is only valid for function AEs.

- **isDataUncorrelated(self)**
  Determines if the function data is uncorrelated.
  - Returns
    (boolean) TRUE if the function data is uncorrelated; FALSE otherwise.

  This function is only valid for function AEs.

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

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

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

- **getLogMask(self)**
  Gets the log mask that the AE is running on.
  - Returns
    (LOG_MASK) The log mask on which the AE is running.

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

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

- **getSuggestedMemoryLimit(self)**
  Gets the suggested memory limit for this AE.
  - Returns
    (integer) The suggested memory limit for this AE.

- **getSessionId(self)**
  Gets the session ID associated with this AE.
  - Returns
    (integer) The session ID associated with this AE.

- **getTransactionId(self)**
  Gets the transaction ID associated with this AE.
  - Returns
    (integer) The transaction ID associated with this AE.

- **isAUserQuery(self)**
  Determines if the AE is handling a user query.
  - Returns
    (boolean) TRUE if the AE is handling a user query; FALSE otherwise.

- **isLoggingEnabled(self)**
  Determines if logging is enabled.

- Returns
  (boolean) TRUE if logging is enabled; FALSE otherwise.

- **isRunningInPostgres(self)**
  Determines if this AE is running in Postgres.
  - Returns
    (boolean) TRUE if this AE is running in Postgres; FALSE otherwise.

- **isRunningInDbos(self)**
  Determines if this AE is running in DBOS.
  - Returns
    (boolean) TRUE if this AE is running in DBOS; FALSE otherwise.

- **isRunningOnSpu(self)**
  Determines if this AE is running on the SPU.
  - Returns
    (boolean) TRUE if this AE is running on the SPU; FALSE otherwise.

- **isRunningOnHost(self)**
  Determines if this AE is running on the host.
  - Returns
    (boolean) TRUE if this AE is running on the host; FALSE otherwise.

- **getSharedLibraryPath(self, libraryName, caseSensitive=True)**
  Gets the path to a shared library.
  - Parameters
    - **libraryName**
      (string) The name of the shared library.
    - **[caseSensitive=True]**
      (boolean) Specifies whether to search in a case-sensitive manner.
  - Returns
    (string) The full path to the shared library.

  Throws an AeSharedLibraryNotFoundException if the shared library is not found.

- **yieldSharedLibraries(self, forProcess=False)**
  Returns a generator that yields a shared library name and the corresponding full path for each shared library entry.
  - Parameters
    - **[forProcess=False]**
      (boolean) If set this yields shared library names and full paths for the remote AE process.

- Returns
  (generator -> tuple<string, string>) Each iteration over the generator yields a library name and the full path to that library.

NOTE: This function is generally called by using a FOR clause to retrieve the non-process shared library entries, for example: "FOR name, path IN self.yieldSharedLibraries(): ...".

- **getEnvironment(self)**
  Returns a dict containing the entire environment.
  - Returns
    (dict<string, string>) The environment.

- **getEnvironmentVariable(self, variableName, defaultValue)**
  Gets the value of an entry in the environment.
  - 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 for <variableName>. If no default value is specified, and the variable is not in the environment, an exception is thrown.

- **userError(self, errorString)**
  Ends the query and sends the user an error message.
  - Parameters
    - **errorString**
      (string) The error message to send to the user.

- **getLogFilePath(self)**
  Gets the log file path.
  - Returns
    (string) The log file path.

- **log(self, text, logLevel=None)**
  Log a message to the AE log file and optionally to stderr and/or the Python AE log.
  - Parameters
    - **text**
      (string) The message to log.
    - **logLevel**

(LOG_LEVEL) The level at which to log; None defaults to LOG_LEVEL__TRACE.

If the registered LOG_LEVEL is set above 0, the message also logs to stderr. If the registered LOG_LEVEL is set above 4, the message also logs to the log file found at /nz/export/ae/pythonDebugLogs.

- **__iter__(self)**
  Yield rows of data.

  Only valid for function AEs. This is an overloaded standard iterator that yield rows of input data. The Python value None is returned in the list if a NULL database value is encountered.

- **getInputRow(self)**
  Returns a list containing the elements of the current input row.

  - Returns
    (list<object>) The input row.

  The Python value None is returned in the list if a NULL database value is encountered.

- **getInputValue(self, columnIndex)**
  Returns the value of the current input at a specified column index.

  - Parameters
    - **columnIndex**
      (integer) The column index of the value to fetch.
  - Returns
    (object) The value of the current input at the specified column index.

  The Python value None is returned if a NULL database value is encountered.

- **getInputString(self, columnIndex)**
  Returns the String value of the current input at a specified column index.

  - Parameters
    - **columnIndex**
      (integer) The column index of the value to fetch.
  - Returns
    (object) The String value of the current input at the specified column index.

  The Python value None is returned if a NULL database value is encountered.

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

- **outputCurrentRow(self)**
  Outputs the current row.

- **outputInputColumn(self, inputColumnIndex, outputColumnIndex)**
  Copies an input field into an output field.
  - Parameters
    - **inputColumnIndex**
      (integer) The column index of the input row to copy.
    - **outputColumnIndex**
      (integer) The column index of the output to write to.

- **getState(self, columnIndex=None)**
  Gets a list containing the current state.
  - Parameters
    - **[columnIndex]**
      (integer) The column index of state to retrieve. If not provided, the function returns a list containing the whole row of state.
  - Returns
    (list<object> or object) If no index is specified, a list of the current state. Otherwise, the state at the specified column index.

  Throws an exception if called when the aggregate state is not available. The Python value None is returned where the state is NULL in the database.

- **getInputState(self, columnIndex=None)**
  Gets a list containing the input state, that is, the state to merge.
  - Parameters
    - **[columnIndex]**
      (integer) The column index of state to retrieve. If not specified the function returns a list containing the whole row of state.
  - Returns
    (list<object>) A list of the state to merge.

  Throws an exception if not called during the "merge" or "finalize" process state of an aggregate.

  Throws an exception if called when aggregate input state is not available. The Python value None is returned where the state is NULL in the database.

- **setState(self, indexOrState, state)**
  Sets the current state.
  - Parameters
    - **indexOrState**
      (any) If two arguments are specified, the column index of the state to set. Otherwise, a list used to set the entire state.

- **[state]**
  (object) The state to set if setting a single column of the state with the function.

The Python value None should be used if the state is to be set to NULL in the database.

- **getNumberOfStateColumns(self)**
  Gets the number of columns in the state.
  - Returns
    (integer) The number of columns in the state.

- **getStateDataType(self, columnIndex)**
  Gets the data type of a field of state at the specified column index.
  - Parameters
    - **columnIndex**
      (string) The index of the column of state.
  - Returns
    (integer) The data type of the column.

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

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

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

- **addOutputColumn(self, columnName, dataType)**
  Add an output column.
  - Parameters
    - **columnName**
      (string) The name of the column to add.
    - **dataType**
      (DATA_TYPE) The data type of the column.

- **addOutputColumnNumeric(self, columnName, dataType, precision, scale)**
  Add a numeric output column.
  - Parameters
    - **columnName**
      (string) The name of the column to add.
    - **dataType**
      (DATA_TYPE) The data type of the column.
    - **precision**
      (integer) The precision of the numeric to add.
    - **scale**
      (integer) The scale of the numeric to add.

- **addOutputColumnString(self, columnName, dataType, size)**
  Add a string output column.
  - Parameters
    - **columnName**
      (string) The name of the column to add.
    - **dataType**
      (DATA_TYPE) The data type of the column.
    - **size**
      (integer) The scale of the numeric to add.

- **isInputValueAvailable(self, columnIndex)**
  Determines if an input value is available for a shaper/sizer.
  - Parameters
    - **columnIndex**
      (integer) The index of the column.
  - Returns
    (boolean) TRUE if the input is available; FALSE otherwise.

- **isShaperSystemCatalogUpperCase(self)**
  Determines if the Netezza system catalog is upper case.
  - Returns
    (boolean) TRUE if the system catalog is upper case; FALSE otherwise.

- **isUdfSizer(self)**
  Determines if the AE is a sizer for a UDF.
  - Returns
    (boolean) TRUE if the AE is a UDF sizer and FALSE otherwise.

- **isUdtfShaper(self)**
  Determines if the AE is a shaper for a UDTF.
  - Returns
    (boolean) TRUE if the AE is a UDTF shaper; FALSE otherwise.

Static Member Data Documentation

- list STRING_DATA_TYPES=[_AeInternal.DATA_TYPE__FIXED, _AeInternal.DATA_TYPE__VARIABLE, _AeInternal.DATA_TYPE__NATIONAL_FIXED, _AeInternal.DATA_TYPE__NATIONAL_VARIABLE, _AeInternal.DATA_TYPE__GEOMETRY, _AeInternal.DATA_TYPE__VARBINARY]
  String Data Types.

- list INTEGER_DATA_TYPES=[_AeInternal.DATA_TYPE__INT8, _AeInternal.DATA_TYPE__INT16, _AeInternal.DATA_TYPE__INT32, _AeInternal.DATA_TYPE__INT64]
  Integer Data Types.

- list NUMERIC_DATA_TYPES=[_AeInternal.DATA_TYPE__NUMERIC32, _AeInternal.DATA_TYPE__NUMERIC64, _AeInternal.DATA_TYPE__NUMERIC128]
  Numeric Data Types.

- int REQUEST_HANDLING_STYLE__USE_THREADS=1
  Use threads.

- int REQUEST_HANDLING_STYLE__FORK=2
  Fork.

- int REQUEST_HANDLING_STYLE__SINGLE_THREADED=3
  Single-threaded.

- DEFAULT_REQUEST_HANDLING_STYLE=REQUEST_HANDLING_STYLE__USE_THREADS
  The default handling style defaults to use-threads.

- tuple CONNECTION_POINT_NAME=_AeInternal._getConnectionPointNameFromEnvironment()

- int DATA_TYPE__UNKNOWN=1
  Unknown data type.

- int DATA_TYPE__FIXED=0
  Fixed (CHAR) data type.

- int DATA_TYPE__VARIABLE=1
  Variable (VARCHAR) data type.

- int DATA_TYPE__NATIONAL_FIXED=2
  National Fixed data type.

- int DATA_TYPE__NATIONAL_VARIABLE=3
  National Variable data type.

- int DATA_TYPE__BOOLEAN=4
  Boolean data type.

- int DATA_TYPE__DATE=5
  Date data type.

- int DATA_TYPE__TIME=6
  Time data type.

- int DATA_TYPE__TIMETZ=7
  Time Zone data type.

- int DATA_TYPE__NUMERIC32=8
  Numeric 32 data type.

- int DATA_TYPE__NUMERIC64=9
  Numeric 64 data type.

- int DATA_TYPE__NUMERIC128=10
  Numeric 128 data type.

- int DATA_TYPE__FLOAT=11
  Float data type.

- int DATA_TYPE__DOUBLE=12
  Double data type.

- int DATA_TYPE__INTERVAL=13
  Interval data type.

- int DATA_TYPE__INT8=14
  Int 8-bit data type.

- int DATA_TYPE__INT16=15
  Int 16-bit data type.

- int DATA_TYPE__INT32=16
  Int 32-bit data type.

- int DATA_TYPE__INT64=17
  Int 64-bit data type.

- int DATA_TYPE__TIMESTAMP=18
  Timestamp data type.

- int DATA_TYPE__GEOMETRY=19
  Geometry data type.

- int DATA_TYPE__VARBINARY=20
  Varbinary data type.

- int AGGREGATION_TYPE__ERROR=1
  Error state.

- int AGGREGATION_TYPE__END=0

End of aggregation state.

- int AGGREGATION_TYPE__INITIALIZE=1
  Initialize state aggregation state.

- int AGGREGATION_TYPE__ACCUMULATE=2
  Accumulate aggregation state.

- int AGGREGATION_TYPE__MERGE=3
  Merge aggregation state.

- int AGGREGATION_TYPE__FINAL_RESULT=4
  Final result aggregation state.

- int LOG_LEVEL__TRACE=1
  Log level: Trace.

- int LOG_LEVEL__DEBUG=2
  Log level: Debug.

- CONNECTION_POINT_NAME=None
  Override this member variable to set or override the connection point name.

Overridable Member Function Documentation

- **_accumulate(self, state, row)**
  Override this function to handle the accumulate process state of aggregates.

  - Parameters
    - **state**
      (list<object>) The current row of state.
    - **row**
      (list<object>) A row of input.

  This function should call setState to change the current state for the input row. The Python value None is passed in where the database value is NULL.

- **_cleanUp(self)**
  Override this function for one-time cleanup of the AE.

- **_finalResult(self, state)**
  Override this function to handle the Finalize processing state for aggregates.
  - Parameters
    - **state**
      (list<object>) The current row of state.
  - Returns
    (object) The final result of the aggregation. Return the Python value None to return NULL in the database.

- **_getFunctionResult(self, row)**
  Override this to implement UDF or UDTF functionality where there is exactly one output per input.
  - Parameters
    - **row**
      (list) The input row to the function.
  - Returns
    row (list or single value) The result of the function. For UDF AEs, this must be either a list with one value or only the value. In any case, this result must match the output signature for the UDF/UDTF.

  To output rows for a UDTF where there is not a one-to-one mapping of input to output, override _runUdtf instead of this function. The Python value None is passed in where the database value is NULL, and None should be returned to return a NULL database value.

- **_getRemoteProtocolStatus(Class)**
  Override this function to provide a different status string to the remote protocol status callback.
  - Returns
    (string) The current status of the executable.

- **_handleRemoteProtocol(Class, code, data)**
  Override this function 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 O K, and the return data.

  In general, an AE writer does override this function, but this capability is provided for the advanced AE writer. Normally, an AE writer overrides one of: _getRemoteProtocolStatus , _handleRemoteProtocolControlData , _handleRemoteProtocolRequest , or _handleRemoteProtocolStopRequest().

- **_handleRemoteProtocolControlData(Class, data)**

Override this function 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.

- **_handleRemoteProtocolRequest(Class, request)**
  Override this function 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.

- **_handleRemoteProtocolStopCommand(Class)**
  Override this function to handle the remote protocol stop command in a non-standard way.

  The default implementation calls Class._cleanUpClass() and sys.exit().

- **_initializeState(self)**
  Override this function to handle the Initialize processing state for aggregates.

- **_merge(self, state, inputState)**
  Override this function to handle the Merge processing state for aggregates.
  - Parameters
    - **state**
      (list<object>) The current state.
    - **inputState**
      (list<object>) The state to merge with the current state.

  This function should call setState to change the current state given the input state. The Python value None is passed in where the database value is NULL.

- **_onIdle(Class)**
  Override this function to handle idle time for remote AEs.

- **_run(self)**
  Override this function to handle generic running of AEs.

  The ability to override this function is provided for advanced AE writers. The _run function is

called for all AEs. In general, an AE writer should derive from the Ae class and override _runUdtf , _runUdf, _runUda , _runShaper , or _runSizer as appropriate.

- **_runInstance(Class, instance)**
  Override this function to handle generic running of an AE instance.

  - Parameters
    - **instance**
      (Ae) The Ae instance to run.

  This function calls _setup , _run , _cleanUp , done and close for a specified AE instance, and handles exceptions appropriately. Typically, an AE writer does not override this function, but the capability is provided for advanced AE writers.

- **_runLocal(Class)**
  Override this function to handle running only local AEs.

  This function instantiates the AE and then calls _runInstance . Typically, an AE writer does not override this function, but the capability is provided for advanced AE writers.

- **_runRemote(Class)**
  Override this function to handle how remote AEs are run.

  This function creates an AE remote listener, accepts requests from the Netezza software, forks or starts a thread as appropriate, creates an instance of the AE, and calls _runInstance . It also handles exceptions appropriately. Typically, an AE writer does not override this function, but the capability is provided for advanced AE writers.

- **_runShaper(self)**
  Override this function to handle running a shaper.

- **_runSizer(self)**
  Override this function to handle running a sizer.

- **_runUda(self)**
  Override this function to handle running an aggregate.

  In general, if the AE is an aggregate, it should override _initializeState , _aggregate(), _merge , and _finalResult . However, more fine tuning might be performed by overriding this function instead.

- **_runUdf(self)**
  Override this function to handle running a UDF.

  By default, this function fetches input rows and calls _getFunctionResult . It then outputs the result of the function call. Consequently, an AE writer can override _getFunctionResult for a simpler interface to UDF functionality.

- **_runUdtf(self)**
  Override this function to handle running a UDTF.

  By default, this function fetches input rows and calls _getFunctionResult . It then outputs the result of the function call. This function should be overridden (as compared to _getFunction-Result) if an AE writer wants to output greater than or less than 1 row of output for each row of input.

- **_setup(self)**
  Override this for one-time setup of the AE instance.

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

AeEnvironmentVariableNotFoundException Class Reference

Raised when an AE environment variable is asked for, but not found.

Inherits AeException

Detailed Description

Raised when an AE environment variable is asked for, but not found.

AeException Class Reference

The base AE Exception class.

Detailed Description

The base AE Exception class.

AeInitialiationFailedException Class Reference

Raised when AE Initialization fails.

Inherits AeException

Detailed Description

Raised when AE Initialization fails.

AeInvalidStateException Class Reference

Raised when the AE Class is in an invalid state.

Inherits AeException

Detailed Description

Raised when the AE Class is in an invalid state.

AeSharedLibraryNotFoundException Class Reference

Raised when an AE shared library is asked for, but not found.

Inherits AeException

Detailed Description

Raised when an AE shared library is asked for, but not found.

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

G

I

L

N

O

P

R

S

U

Y