# IBM SPSS Modeler 16 Python Scripting API Reference Guide

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 95.

**Product Information**

This edition applies to version 16, release 0, modification 0 of IBM® SPSS® Modeler and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Chapter 1. General Classes

This provides general classes used by other parts of the API.

## `enum` Objects

This class provides the basis for all enumerated classes in the Modeler API. Note that enum names are not case-sensitive.

```
e.equals(object) : boolean
```

`object (Object)`: the other object.

Returns `True` if the supplied object is equal to this object.

```
e.getName() : string
```

Returns the name of the enumeration. Same as `toString()`.

```
e.hashCode() : int
```

Returns the hash code for this object.

```
e.isUnknown() : boolean
```

Returns `True` if this enumeration value represents an unknown or undefined object or `False` otherwise. An enumeration may contain at most one value that represents an unknown or undefined value.

```
e.toString() : string
```

Returns the name of the enumeration. Same as `getName()`.

## `ModelerException` Objects

A generic Predictive Server exception.

## `VersionInfo` Objects

Default Modeler API version information. The version information is made up of the major version, minor version and patch version. A change in the major version number indicates significant extensions to the functionality; a change in the minor version number indicates small extensions or refinements to the functionality; a change in the patch version number indicates a bug-fix release and that the documented functionality has not changed.

Constants:

```
MAJOR_VERSION (int) :
MINOR_VERSION (int) :
RELEASE_VERSION (int) :
```

```
getBuildDate() : Date
```

Returns the build date and time. If the build time is not available then `None`is returned.

```
getBuildVersion() : string
```

Returns the build version of this library. If the build version is not available then the empty string is returned.

```
getCopyright() : string
```

Returns the copyright message.

`getMajorVersion() : int`

Returns the major version number.

`getMinorVersion() : int`

Returns the minor version number.

`getName() : string`

Returns the name of the library.

`getPatchVersion() : int`

Returns the patch version number.

# Chapter 2. Core Objects

This provides the base objects used by other parts of the API.

## `ASCredentialDescriptor` Objects

Define credential to log in the Analytic Server

```
a.getPassword() : string
```

Get password

```
a.getUserName() : string
```

Get user name

## `ApplicationData` Objects

Some features of the Modeler API include information such as the name and version of the application that built them. This supports the ability to brand the Modeler API. By changing this information held by the `SessionFactory`, an application can modify the apparent source of an attribute. Note however that this does not affect "internal" version information associated with some Modeler API features (e.g., persistence).

```
a.getApplicationName() : string
```

Returns the name of the application e.g., "Predictive Server API".

```
a.getApplicationVersion() : string
```

Returns the version of the application e.g., "16.0.0".

```
a.getCopyright() : string
```

Returns the application copyright information e.g., "Copyright (c) 2004-2013, IBM Corp.".

```
a.getVendorName() : string
```

Returns the name of the vendor e.g., "IBM Corp.".

## `ContentFormat` Objects

Subclass of `Enum`.

Enumerates the valid content formats for content containers.

Constants:

`BINARY (ContentFormat) :` Indicates that the format is binary.

`UTF8 (ContentFormat) :` Indicates that the format is UTF8.

```
getEnum(name) : ContentFormat
```

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or `None` if no enumeration exists for the supplied name.

```
getValues() : ContentFormat[]
```

Returns an array containing all the valid values for this enumeration class.

## `ContentProvider` Objects

Stores and retrieves content for an application. Content is identified using *tags* defined by the application. It is the application's responsibility to ensure that tags are unique, and a class-like naming schme is recommended, e.g. *x.y.z....*

```
c.getContent(tag) : Object
```

`tag (string)` : the content tag

Returns the content identified by the specified tag or None if the tag is unknown to this provider. The type of the result is determined by the content format.

```
c.getContentAsBinary(tag) : byte[]
```

`tag (string)` : the content tag

Returns the binary content identified by the specified tag or None if the tag is unknown to this provider or the content does not have binary format.

```
c.getContentAsUTF8(tag) : string
```

`tag (string)` : the content tag

Returns the string content identified by the specified tag or None if the tag is unknown to this provider or the content does not have string format.

```
c.getContentFormat(tag) : ContentFormat
```

`tag (string)` : the content tag

Returns the format of the content identfied by the specified tag or None if the tag is unknown to this provider. The format determines the content type.

```
c.getContentTagIterator() : Iterator
```

Returns an iterator over the content tags known to this provider.

```
c.isContentCurrent(tag) : boolean
```

`tag (string)` : the content tag

Returns `True` if the specified content is considered current (up-to-date) with respect to its context, e.g. its containing stream. Returns `False` if the content tag is unknown to this provider, or as a hint that the content may be stale.

```
c.putContent(tag, format, content)
```

`tag (string)` : the content tag

`format (ContentFormat)` : the content format

`content (Object)` : the content

Stores content in this provider with the specified tag and format. The content must not be `None` and its type must be compatible with the format. Any existing content with the same tag is replaced.

```
c.putContentAsBinary(tag, content)
```

`tag (string)` : the content tag

content (byte[]) : the content

Stores binary content in this provider under the specified tag. The content must not be None.
c.putContentAsUTF8(tag, content)

tag (string) : the content tag

content (string) : the content

Stores string content in this provider under the specified tag. The content must not be None.
c.removeContent(tag)

tag (string) : the content tag

Removes any content stored by this provider under the specified tag.

## FileFormat Objects

Subclass of Enum.

This class defines constants for the different file formats supported by objects in the Modeler API.

Constants:

BITMAP (FileFormat) : Identifier for bitmap image format.

COMMA_DELIMITED (FileFormat) : Identifier for comma-delimited text.

COMPOSITE_PROCESSOR (FileFormat) : Identifier for the native SuperNode format.

DOCUMENT_OUTPUT (FileFormat) : Identifier for the native DocumentOutput format.

EXTERNAL_MODULE_SPECIFICATION (FileFormat) : Identifier for the native external module specification format.

HTML (FileFormat) : Identifier for HTML format.

HTMLC (FileFormat) : Identifier for HTMLC format.

JAR (FileFormat) : Identifier for JAR (Java archive) format.

JPEG (FileFormat) : Identifier for JPEG image format.

MODEL_OUTPUT (FileFormat) : Identifier for the native ModelOutput format.

MODEL_OUTPUT_SET (FileFormat) : Identifier for the native ModelOutput set format.

MS_EXCEL (FileFormat) : Identifier for MS Excel format.

MS_POWERPOINT (FileFormat) : Identifier for MS PowerPoint format.

MS_WORD (FileFormat) : Identifier for MS Word format.

PASW_RULE (FileFormat) : Identifier for PASW RULE format.

PLAIN_TEXT (FileFormat) : Identifier for plain text format.

PNG (FileFormat) : Identifier for PNG image format.

PROCESSOR (FileFormat) : Identifier for the native Node format.

PROCESSOR_STREAM (FileFormat) : Identifier for the native Stream format.

PROJECT (FileFormat) : Identifier for the native Project format.

PUBLISHED_IMAGE (FileFormat) : Identifier for the native image format produced by a Publisher node.

PUBLISHED_PARAMETERS (FileFormat) : Identifier for the native image parameter format produced by a Publisher node.

RTF (FileFormat) : Identifier for Rich Text Format.

SPSS_APPLICATION_VIEW (FileFormat) : Identifier for SPSS application view format.

SPSS_DATA (FileFormat) : Identifier for SPSS .sav data format.

SPSS_DATA_PROVIDER (FileFormat) : Identifier for SPSS data provider format.

SPSS_SCENARIO (FileFormat) : Identifier for SPSS scenario format.

SPSS_SCENARIO_TEMPLATE (FileFormat) : Identifier for SPSS scenario template format.

SPV (FileFormat) : Identifier for SPV format.

SPW (FileFormat) : Identifier for SPW format.

STATE (FileFormat) : Identifier for the native combined ModelOutput and Stream format.

TAB_DELIMITED (FileFormat) : Identifier for tab-delimited text.

UNKNOWN (FileFormat) : Identifier for an unknown file format.

VIZ (FileFormat) : Identifier for VIZ format.

VIZML (FileFormat) : Identifier for VIZML format.

XML (FileFormat) : Identifier for XML format.

ZIP (FileFormat) : Identifier for Zip (compressed archive) format.

f.equals(object) : boolean

object (Object) :

Returns True if the supplied object is equal to this. Two file formats are considered equal if the names are equal ignoring the case of the name.

f.getDefaultExtension() : string

Returns the default file extension for this file format or an empty string if no file extensions are defined for this file format. The file extension includes the leading ".".

getEnum(name) : FileFormat

name (string) : the enumeration name

Returns the enumeration with the supplied name or `None` if no enumeration exists for the supplied name. The lookup is not case-sensitive.

`f.getExtensionAt(index) : string`

`index (int) :` the file extension index

Returns the file extansion at the supplied index. The first (0'th) extension is assumed to be the default extension. The file extension includes the leading ".".

`f.getExtensionCount() : int`

Returns the number of file extensions associated with this file format.

`f.getFileFilterPattern() : string`

Returns a file filter pattern suitable for use with a file chooser dialog that includes all the file extensions associated with this file type. An empty string is returned if no file extensions are defined for this file type.

`getFileFormatForExtension(extension) : FileFormat`

`extension (string) :` the file extension

Returns the default file format for the supplied extension or None if no matching file format can be found. The extension search is not case-sensitive.

`getFileFormatForMIMEType(mimeType) : FileFormat`

`mimeType (string) :` the mime type to be searched for

Returns the file format for the supplied MIME type or `None` if no matching file format can be found.

`getFileFormatsForExtension(extension) : List`

`extension (string) :` the file extension

Returns a list of all file formats with the supplied extension. The extension search is not case-sensitive.

`f.getMIMEType() : string`

Returns the MIME type associated with this file format or an empty string if no MIME type has been defined.

`getValues() : FileFormat[]`

Returns an array containing all the valid values for this enumeration class.

`f.isUnknown() : boolean`

Returns `True` if this value is `UNKNOWN`.

`f.isValidExtension(extension) : boolean`

`extension (string) :` the file extension

Returns `True` if the supplied extension is valid for this file format or `False` otherwise. `None` values and the empty string always return `False`.

## IncompatibleServerException Objects

Subclass of ServerConnectionException.

An exception thrown while attempting to connect to an incompatible server version.

A message string is always created for this exception.

```
i.getServerMajorVersion() : int
```

Returns the major version returned by the server being connected to.

```
i.getServerMinorVersion() : int
```

Returns the minor version returned by the server being connected to.

```
i.getServerPatchVersion() : int
```

Returns the patch version returned by the server being connected to.

## InvalidPropertyException Objects

Subclass of ModelerException.

An exception thrown when an invalid property is accessed or an invalid value is supplied for a property.

A message string is always created for this exception.

```
i.getProperty() : string
```

Returns the property name.

```
i.getValue() : Object
```

Returns the value being assigned or None.

## ModelFieldRole Objects

Subclass of Enum.

This class enumerates the valid model output roles.

Constants:

ADJUSTED_PROBABILITY (ModelFieldRole) : Indicates a field containing the adjusted probability of the target value.

ENTITY_AFFINITY (ModelFieldRole) : Indicates a field containing the affinity or distance of the value from the entity.

ENTITY_ID (ModelFieldRole) : Indicates a field containing the entity ID associated with the cluster, tree node, neuron or rule.

LOWER_CONFIDENCE_LIMIT (ModelFieldRole) : Indicates a field containing the lower confidence limit of a numeric prediction.

PREDICTED_DISPLAY_VALUE (ModelFieldRole) : Indicates a field containing a displayed-friendly representation of the predicted value.

PREDICTED_VALUE (ModelFieldRole) : Indicates a field containing the predicted value.

PROBABILITY (ModelFieldRole) : Indicates a field containing the probability of the target value.

PRODICTED_SEQUENCE (ModelFieldRole) : Indicates a field containing the predicted sequence.

PRODICTED_SEQUENCE_PROBABILITY (ModelFieldRole) : Indicates a field containing the probability associated with the predicted sequence.

PROPENSITY (ModelFieldRole) : Indicates a field containing the propensity of the prediction.

RESIDUAL (ModelFieldRole) : Indicates a field containing the residual of the predicted numeric value.

STANDARD_DEVIATION (ModelFieldRole) : Indicates a field containing the standard deviation of the target value.

STANDARD_ERROR (ModelFieldRole) : Indicates a field containing the standard error of the predicted numeric value.

SUPPLEMENTARY (ModelFieldRole) : Indicates a field containing some additional information about the value that cannot be categorized into one of the other model field roles.

SUPPORT (ModelFieldRole) : Indicates a field containing the support associated with an association model rule.

UNKNOWN (ModelFieldRole) : Indicates that the role cannot be determined.

UPPER_CONFIDENCE_LIMIT (ModelFieldRole) : Indicates a field containing the upper confidence limit of a numeric prediction.

VALUE (ModelFieldRole) : Indicates a field containing the model output value for non-predictive models.

VARIANCE (ModelFieldRole) : Indicates a field containing the variance of the target value.

```
getEnum(name) : ModelFieldRole
```

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : ModelFieldRole[]
```

Returns an array containing all the valid values for this enumeration class.

```
m.isUnknown() : boolean
```

Returns True if this value is UNKNOWN.

## ObjectCreationException Objects

Subclass of ModelerException.

An exception thrown when an instance of an object cannot be created.

No message string is set for this exception.

## ObjectLockedException Objects

Subclass of ModelerException.

An exception thrown when an attempt is made to lock or modify an object that is already locked. Locking an object prevents an updates to that object while the lock is in effect. A `Stream` and all its `Nodes` are locked by the `Session` during execution.

No message string is set for this exception.

`o.getObject() : Object`

Returns the object.

## `OwnerException` Objects

Subclass of `ModelerException`.

An exception thrown when an object owner requires an owned object.

No message string is set for this exception.

`o.getOwner() : Object`

Returns the object owner.

`o.getUnownedObject() : Object`

Returns the object that is not owned by the owner.

## `ParameterDefinition` Objects

Describes a parameter which affects the behaviour of some Modeler API objects. A parameter definition is obtained from a `ParameterProvider`. The parameter name is unique within the provider.

Parameters are modified through the provider. A `ParameterDefinition` instance may be a snapshot of a parameter definition at the time it was obtained from the provider and need not reflect subsequent modifications.

`p.getFalseFlag() : Object`

Returns the *false* indicator for this flag parameter. Returns `None` if the parameter type is other than `ParameterType.FLAG` or if no false value has been declared for the flag.

`p.getLowerBound() : Object`

Returns a lower bound on the valid values of this parameter. Returns `None` if the parameter type is other than `ParameterType.RANGE` or if no lower bound has been declared for the range.

`p.getParameterLabel() : string`

Returns a label for this parameter. The default label is the empty string.

`p.getParameterName() : string`

Returns the name of this parameter. The name is unique within the parameter provider.

`p.getParameterStorage() : ParameterStorage`

Returns the storage type of this parameter.

`p.getParameterType() : ParameterType`

Returns the measure of this parameter.

`p.getParameterValue() : Object`

Returns the value of this parameter.

`p.getSetValues() : Object[]`

Returns the valid values of this parameter. Returns `None` if the parameter type is other than `ParameterType.SET` or if no values have been declared for the set.

`p.getTrueFlag() : Object`

Returns the *true* indicator for this flag parameter. Returns `None` if the parameter type is other than `ParameterType.FLAG` or if no true value has been declared for the flag.

`p.getUpperBound() : Object`

Returns an upper bound on the valid values of this parameter. Returns `None` if the parameter type is other than `ParameterType.RANGE` or if no upper bound has been declared for the range.

`p.isValidValue(value) : boolean`

`value (Object) :` the value

Returns `True` if the specified value is valid for this parameter. The value must be compatible with the parameter storage type and with the parameter values when they are specified.

## `ParameterProvider` Objects

Identifies objects that contain parameters. Parameters do not control the behaviour of the implementing class directly but provide a look-up mechanism for values that may affect the behaviour of other objects.

`p.getParameterDefinition(parameterName) : ParameterDefinition`

`parameterName (string) :` the parameter name

Returns the parameter definition for the parameter with the specified name or `None` if no such parameter exists in this provider. The result may be a snapshot of the definition at the time the method was called and need not reflect any subsequent modifications made to the parameter through this provider.

`p.getParameterLabel(parameterName) : string`

`parameterName (string) :` the parameter name

Returns the label of the named parameter or `None` if no such parameter exists.

`p.getParameterStorage(parameterName) : ParameterStorage`

`parameterName (string) :` the parameter name

Returns the storage of the named parameter or `None` if no such parameter exists.

`p.getParameterType(parameterName) : ParameterType`

`parameterName (string) :` the parameter name

Returns the type of the named parameter or `None` if no such parameter exists.

`p.getParameterValue(parameterName) : Object`

`parameterName (string) :` the parameter name

Returns the value of the named parameter or `None` if no such parameter exists.

`p.parameterIterator() : Iterator`

Returns an iterator of parameter names for this object.

`p.setParameterLabel(parameterName, label)`

parameterName (string) : the parameter name

label (string) : the parameter label

Sets the label of the named parameter.

Exceptions:

ObjectLockedException : if the parameter provider is locked
p.setParameterStorage(parameterName, storage)

parameterName (string) : the parameter name

storage (ParameterStorage) : the parameter storage

Sets the storage of the named parameter.

Exceptions:

ObjectLockedException : if the parameter provider is locked
p.setParameterType(parameterName, type)

parameterName (string) : the parameter name

type (ParameterType) : the parameter type

Sets the type of the named parameter.

Exceptions:

ObjectLockedException : if the parameter provider is locked
p.setParameterValue(parameterName, value)

parameterName (string) : the parameter name

value (Object) : the parameter value

Sets the value of the named parameter.

Exceptions:

ObjectLockedException : if the parameter provider is locked

## ParameterStorage Objects

Subclass of Enum.

This class enumerates the valid storage types for columns in a DataModel or parameters.

Constants:

DATE (ParameterStorage) : Indicates that the storage type is a date type.

INTEGER (ParameterStorage) : Indicates that the storage type is an integer.

REAL (ParameterStorage) : Indicates that the storage type is a floating point number.

STRING (ParameterStorage) : Indicates that the storage type is a string.

TIME (ParameterStorage) : Indicates that the storage type is a time type.

TIMESTAMP (ParameterStorage) : Indicates that the storage type is a combined time and date type.

UNKNOWN (ParameterStorage) : Indicates that the storage type is unknown.
getEnum(name) : ParameterStorage

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.
p.getStorageClass() : Class

Returns the Java class that parameter values with this storage will be returned as.
getValues() : ParameterStorage[]

Returns an array containing all the valid values for this enumeration class.

## ParameterType Objects

Subclass of Enum.

This class enumerates the valid types of parameter support by a ParameterProvider.

Constants:

FLAG (ParameterType) : Indicates that the parameter value is one of two values.

RANGE (ParameterType) : Indicates that the parameter value lies between two points on a scale.

SET (ParameterType) : Indicates that the parameter value is one of a (small) set of discrete values.

TYPELESS (ParameterType) : Indicates that the parameter can have any value compatible with its storage.
getEnum(name) : ParameterType

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.
getValues() : ParameterType[]

Returns an array containing all the valid values for this enumeration class.

## PropertiedObject Objects

This encapsulates the functionality of objects that contain settable properties. Properties are split into two sub-groups:

- **Simple properties**: these are basic name/value pairs
- **Keyed properties**: these are properties associated with specific objects

An example of a simple property is a Boolean value that specifies to a data file reader whether column names are included in the first line of a file:

```
myDataReader.setPropertyValue("read_field_names", Boolean.TRUE);
```

An example of a keyed property is used in an object that allows fields or columns to be removed from the data model. Here the setting is made up of both the property name and the key which is the column name:

```
// Remove "DateOfBirth" column
myColumnFilter.setKeyedPropertyValue("include", "DateOfBirth", Boolean.FALSE);
```

Finally note that properties which correspond with Enum values must be converted to strings by calling getName() before being passed to one of the setter methods:

```
typeProcessor.setKeyedPropertyValue("role", "Drug", ModelingRole.OUT.getName());
```

Similarly, the getter methods will return Strings which can then be converted to the appropriate Enum value:

```
String value = (String) typeProcessor.getKeyedPropertyValue("role", "Drug");
ModelingRole role = ModelingRole.getValue(value);
```

```
p.getKeyedPropertyKeys(propertyName) : List
```

propertyName (string) : the property name

Returns the keys currently defined for the supplied keyed property name.

```
p.getKeyedPropertyValue(propertyName, keyName) : Object
```

propertyName (string) : the property name

keyName (string) : the key name

Returns the value of the named property and key or None if no such property or key exists.

```
p.getLabel() : string
```

Returns the object's display label. The label is the value of the property "custom_name" if that is a non-empty string and the "use_custom_name" property is not set; otherwise, it is the value of getName().

```
p.getName() : string
```

Returns the object's name.

```
p.getPropertyValue(propertyName) : Object
```

propertyName (string) : the property name

Returns the value of the named property or None if no such property exists.

```
p.getSavedByVersion() : double
```

Return the object's saved by version.

Exceptions:

Exception : if the information cannot be accessed for some reason

```
p.isKeyedProperty(propertyName) : boolean
```

propertyName (string) : the property name

Returns True if the supplied property name is a keyed property.

```
p.isServerConnectionRequiredProperty(propertyName) : boolean
```

propertyName (string) : the property name

Returns True if the supplied property name should only be set if there is a valid server connection.

```
p.propertyIterator() : Iterator
```

Returns an iterator of property names for this object.

```
p.setKeyedPropertyValue(propertyName, keyName, value)
```

`propertyName (string) :` the property name

`keyName (string) :` the key name

`value (Object) :` the property value

Sets the value of the named property and key.

Exceptions:

`ObjectLockedException :` if the propertied object is locked

`InvalidPropertyException :` if the property name is unknown or the supplied value is not valid for the property

```
p.setLabel(label)
```

`label (string) :` the object's display label

Sets the object's display label. If the new label is a non-empty string it is assigned to the property `"custom_name"`, and `False` is assigned to the property `"use_custom_name"` so that the specified label takes precedence; otherwise, an empty string is assigned to the property `"custom_name"`, and `True` is assigned to the property .

```
p.setPropertyValue(propertyName, value)
```

`propertyName (string) :` the property name

`value (Object) :` the property value

Sets the value of the named property.

Exceptions:

`ObjectLockedException :` if the propertied object is locked

`InvalidPropertyException :` if the property name is unknown or the supplied value is not valid for the property

```
p.setPropertyValues(properties)
```

`properties (Map) :` the property/value pairs to be assigned

Sets the values of the named properties. Each entry in the Map consists of a key representing the property name and the value which should be assigned to that property.

Exceptions:

`ObjectLockedException :` if the propertied object is locked

`InvalidPropertyException :` if one of the property names is unknown or one the supplied values is not valid for that property

## RepositoryConnectionDescriptor Objects

Defines the basic data elements required to connect to a content repository.

```
r.getDomainName() : string
```

Returns the domain name to be used when logging into the content repository (if applicable) or the empty string.

```
r.getHostName() : string
```

Returns the host name or IP address of the content repository machine.

```
r.getPassword() : string
```

Returns the user's password in plain text. An empty string indicates that there is no password.

```
r.getPortNumber() : int
```

Returns the port number which the content repository is listening on.

```
r.getUseSSL() : boolean
```

Returns `True` if the connection should use a secure socket.

```
r.getUserName() : string
```

Returns the user's login name.

## RepositoryConnectionDescriptor2 Objects

Subclass of `RepositoryConnectionDescriptor`.

Extends the repository connection details for applications which are aware of PASW platform services.

```
r.getSsoToken() : Object
```

Returns the SSO security token or `None` if there is no token.

```
r.getSubject() : Object
```

Returns the platform security subject or `None` if there is no subject.

## RepositoryConnectionDescriptor3 Objects

Subclass of `RepositoryConnectionDescriptor2`.

```
r.getContextRoot() : string
```

Return the context root of the content repository service.

## ServerConnectionDescriptor Objects

Subclass of `SystemServerConnectionDescriptor`.

Defines the basic data elements required to connect to a remote server.

```
s.getPassword() : string
```

Returns the user's password in plain text. An empty string indicates that there is no password.

```
s.isSSOConnect() : boolean
```

Returns `True` if the connection should use SSO.

```
s.isUseSSL() : boolean
```

Returns `True` if the connection should use SSL.

## `ServerConnectionException` Objects

Subclass of `ModelerException`.

An exception thrown while connecting or using a `ServerConnectionDescriptor`.

A message string is always created for this exception.

## `ServerVersionInfo` Objects

Version information for a connected server.

A server version has four components, in decreasing order of priority:

- major
- minor
- release
- fix pack

```
s.getServerVersionFixPack() : int
```

Returns the server fix pack number.

```
s.getServerVersionMajor() : int
```

Returns the server major version number.

```
s.getServerVersionMinor() : int
```

Returns the server minor version number.

```
s.getServerVersionRelease() : int
```

Returns the server release number.

## `StructureAttributeType` Objects

Subclass of `Enum`.

This class enumerates the valid types of attribute supported by a `ParameterProvider`.

Constants:

```
BOOLEAN (StructureAttributeType) :

DATE (StructureAttributeType) :

DOUBLE (StructureAttributeType) :

INTEGER (StructureAttributeType) :

STRING (StructureAttributeType) :
getEnum(name) : StructureAttributeType
```

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or `None` if no enumeration exists for the supplied name.

s.getStorageClass() : Class

Returns the basic Java class used to represent this attribute's value. One of:

- string
- int
- float
- boolean
- Date

getValues() : StructureAttributeType[]

Returns an array containing all the valid values for this enumeration class.

## StructuredValue Objects

This interface defines a structured value. A structured value consists of a number of simple attributes.

s.changeAttributeValue(index, value) : StructuredValue

index (int) : the index

value (Object) : the new value

Returns a new structured value with the attribute value at the specified index changed to the new value. An attribute value cannot be None -- if the supplied value is None, it will be ignored.

s.getAttributeCount() : int

Returns the number of attribute values in the structure.

s.getAttributeValue(index) : Object

index (int) : the index

Returns the attribute value at the specified index.

## SystemServerConnectionDescriptor Objects

Defines the basic data elements that describes a server connection. Note that for security reasons, the user's password is not accessible through this interface.

s.getDataDirectory() : string

Returns the data directory for this connection descriptor.

s.getDomainName() : string

Returns the domain name to be used when logging into the server (if applicable) or the empty string.

s.getHostName() : string

Returns the host name or IP address of the server machine.

s.getPortNumber() : int

Returns the port number which the server is listening on.

s.getServerMajorVersion() : int

Returns the major version of server required by this descriptor.

s.getServerMinorVersion() : int

Returns the minor version of server required by this descriptor.

```
s.getServerPatchVersion() : int
```

Returns the patch version of server required by this descriptor.

```
s.getUserName() : string
```

Returns the user's login name.

# Chapter 3. Data and Metadata

This provides support for metadata such as the names, values and types of data.

## `Column` Objects

This defines the properties of a set of columns. Applications that wish to construct `Column` instances must do so using the `DataModelFactory` rather than implementing the interface directly.

```
c.getColumnLabel() : string
```

Returns the label of the column or an empty string if there is no label associated with the column.

```
c.getColumnName() : string
```

Returns the name of the column.

```
c.getFalseFlag() : Object
```

Returns the "false" indicator value for the column, or `None` if either the value is not known or the column is not a flag.

```
c.getLowerBound() : Object
```

Returns the lower bound value for the values in the column, or `None` if either the value is not known or the column is not continuous.

```
c.getMeasureType() : MeasureType
```

Returns the measure type for the column.

```
c.getMissingValueDefinition() : MissingValueDefinition
```

Returns the missing value definition for the column or `None`.

```
c.getModelOutputMetadata() : ModelOutputMetadata
```

Returns the model output column role for the column if it is a model output column or `None`.

```
c.getModelingRole() : ModelingRole
```

Returns the modeling role for the column.

```
c.getSetValues() : Object[]
```

Returns an array of valid values for the column, or `None` if either the values are not known or the column is not a set.

```
c.getStorageType() : StorageType
```

Returns the storage type for the column.

```
c.getTrueFlag() : Object
```

Returns the "true" indicator value for the column, or `None` if either the value is not known or the column is not a flag.

```
c.getUpperBound() : Object
```

Returns the upper bound value for the values in the column, or `None` if either the value is not known or the column is not continuous.

```
c.getValueLabel(value) : string
```

value (Object) : the value

Returns the label for the value in the column or an empty string if there is no label associated with the value.

c.hasMissingValueDefinition() : boolean

Returns True if the column has a missing value definition.

c.isMeasureDiscrete() : boolean

Returns True if the column is discrete. Columns that are either a set or a flag are considered discrete.

c.isModelOutputColumn() : boolean

Returns True if this is a model output column.

c.isStorageDatetime() : boolean

Returns True if the column's storage is a time, date or timestamp value.

c.isStorageNumeric() : boolean

Returns True if the column's storage is an integer or a real number.

c.isValidValue(value) : boolean

value (Object) : the value

Returns True if the specified value is valid for this storage and valid when the valid column values are known.

## ColumnCountException Objects

Subclass of DataModelException.

A ColumnCountException is thrown when the number of columns in the data did not match the number of columns specified in the data model.

No message string is set for this exception.

c.getValues() : List

Returns the list of values that were being converted.

## ColumnGroup Objects

This defines the properties of a column group. A column group is typically used to identify related columns e.g. that represent a multi response set.

c.getColumnGroupLabel() : string

Returns the label of the column group or an empty string if there is no label associated with the column group.

c.getColumnGroupName() : string

Returns the name of the column group.

c.getColumnGroupType() : ColumnGroupType

Returns the type of column group.

c.getColumnNames() : List

Returns the list of column names in this group. The list is a copy of the list in the group and can be modified without affecting the group definition.

```
c.getCountedValue() : string
```

Returns the counted value for this column group. The counted value only has meaning for multi dichotomy sets and represents the "true" or "selected" value of the column. Note that the value is represented as a string even if the columns named by the column group have storage other than string and it is the responsibility of the calling application to convert the string representation into a valid value.

## `ColumnGroupType` Objects

Subclass of `Enum`.

This class enumerates the valid types for a ColumnGroup.

Constants:

- `GENERAL (ColumnGroupType) :` Indicates that the there are no special semantics associated with this column group.
- `MULTI_CATEGORY_SET (ColumnGroupType) :` Indicates that this column group represents a multi category set.
- `MULTI_DICHOTOMY_SET (ColumnGroupType) :` Indicates that this column group represents a multi dichotomy set.
- `SPLIT_GROUP (ColumnGroupType) :` A group representing the split fields. Ideally the application will keep the fields in the split group consistent with the fields whose role is "split". However, components will use the group to determine the order for the split fields and will typically ignore the "split" role. There should be at most one split group in the data model.

```
getEnum(name) : ColumnGroupType
```

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : ColumnGroupType[]
```

Returns an array containing all the valid values for this enumeration class.

## `DataModel` Objects

This defines the properties of a set of columns. Applications that wish to construct data model instances must do so using the `DataModelFactory` rather than implementing the interface directly.

```
d.columnGroupIterator() : Iterator
```

Returns an iterator that returns each column group in turn.

```
d.columnIterator() : Iterator
```

Returns an iterator that returns each column in the "natural" insert order. The iterator returns instances of Column.

```
d.contains(name) : boolean
```

`name (string) :` the column name

Returns `True` if a column with the supplied name exists in this `DataModel`, `False` otherwise.

```
d.getColumn(name) : Column
name (string) :
```

Returns the column with the specified name.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.getColumnCount() : int
```

Returns the number of columns in this set.

```
d.getColumnGroup(name) : ColumnGroup
name (string) :
```

Returns the named column group or `None` if no such column group exists.

```
d.getColumnGroupCount() : int
```

Returns the number of column groups in this data model.

```
d.getColumnLabel(name) : string
```

`name (string)` : the column name

Returns the label of the named column or an empty string if there is no label associated with the column.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.getFalseFlag(name) : Object
```

`name (string)` : the column name

Returns the "false" indicator value for the column, or `None` if either the value is not known or the column is not a flag.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.getLowerBound(name) : Object
```

`name (string)` : the column name

Returns the lower bound value for the values in the named column, or `None` if either the value is not known or the column is not continuous.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.getMeasureType(name) : MeasureType
```

`name (string)` : the column name

Returns the measure type for values in the named column.

Exceptions:

DataModelException : if the named column does not exist
d.getMissingValueDefinition(name) : MissingValueDefinition

name (string) : the column name

Returns the missing value definition for the column or None.

Exceptions:

DataModelException : if the named column does not exist
d.getModelingRole(name) : ModelingRole

name (string) : the column name

Returns the modeling role for the named column.

Exceptions:

DataModelException : if the named column does not exist
d.getSetValues(name) : Object[]

name (string) : the column name

Returns an array of valid values for the column, or None if either the values are not known or the column is not a set.

Exceptions:

DataModelException : if the named column does not exist
d.getStorageType(name) : StorageType

name (string) : the column name

Returns the storage type for values in the named column.

Exceptions:

DataModelException : if the named column does not exist
d.getTrueFlag(name) : Object

name (string) : the column name

Returns the "true" indicator value for the column, or None if either the value is not known or the column is not a flag.

Exceptions:

DataModelException : if the named column does not exist
d.getUpperBound(name) : Object

name (string) : the column name

Returns the upper bound value for the values in the named column, or None if either the value is not known or the column is not continuous.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.getValueLabel(name, value) : string
```

`name (string)` : the column name

`value (Object)` : the value

Returns the label for the value in the named column or an empty string if there is no label associated with the value.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.hasMissingValueDefinition(name) : boolean
```

`name (string)` : the column name

Returns `True` if the column has a missing value definition.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.isMeasureDiscrete(name) : boolean
```

`name (string)` : the column name

Returns `True` if the column is discrete. Columns that are either a set or a flag are considered discrete.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.isModelOutputColumn(name) : boolean
```

`name (string)` : the column name

Returns `True` if this is a model output column.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.isStorageDatetime(name) : boolean
```

`name (string)` : the column name

Returns `True` if the column's storage is a time, date or timestamp value.

Exceptions:

`DataModelException` : if the named column does not exist

```
d.isStorageNumeric(name) : boolean
```

`name (string)` : the column name

Returns True if the column's storage is an integer or a real number.

Exceptions:

DataModelException : if the named column does not exist

d.isValidValue(name, value) : boolean

name (string) : the column name

value (Object) : the value

Returns True if the specified value is valid for this storage and valid when the valid column values are known.

Exceptions:

DataModelException : if the named column does not exist

d.nameIterator() : Iterator

Returns an iterator that returns the name of each column in the "natural" insert order.

d.toArray() : Column[]

Returns the data model as an array of columns. The columns are ordered in their natural/insert order.

## DataModelException Objects

Subclass of ModelerException.

An exception thrown when there is a mismatch between physical set of data and its expected data model.

No message string is set for this exception.

d.getDataModel() : DataModel

Returns the data model that was being used to convert the values.

## GlobalValues Objects

This defines the set of global values which are usually computed by a stream via a "setglobals" node.

g.fieldNameIterator() : Iterator

Returns an iterator for each field name with at least one global value.

g.getValue(type, fieldName) : Object

type (GlobalValues.Type) : the type of value

fieldName (string) : the field name

Returns the global value for the specified type and field name or None if no value can be located. The returned value is generally expected to be a number although future functionality may return different value types.

g.getValues(fieldName) : Map
fieldName (string) :

Returns a map containing the known entries for the specified field name or None of no entries for the field exist.

## GlobalValues.Type Objects

This defines the set of global value types than can be accessed from the global values table.

Constants:

- MAX (GlobalValues.Type) :
- MEAN (GlobalValues.Type) :
- MIN (GlobalValues.Type) :
- STDDEV (GlobalValues.Type) :
- SUM (GlobalValues.Type) :

```
g.getFunctionName() : string
valueOf(name) : GlobalValues.Type
name (string) :
values() : GlobalValues.Type[]
```

## InvalidColumnExceptionValues Objects

Subclass of DataModelException.

An InvalidColumnValueException is thrown when the value supplied for a column is not consistent with its storage type.

No message string is set for this exception.

```
i.getColumnName() : string
```

Returns the name of the unknown column.

```
i.getValue() : Object
```

Returns the invalid value.

## MeasureType Objects

Subclass of Enum.

This class enumerates the valid measures for columns in a DataModel.

Constants:

- AUTOMATIC (MeasureType) : Indicates that the column value will be auto-selected as RANGE or DISCRETE according to its storage.
- DISCRETE (MeasureType) : Indicates that the column value is like SET but can be treated as FLAG if there are only two values.
- FLAG (MeasureType) : Indicates that the column value is one of two values.
- ORDERED_SET (MeasureType) : Indicates that the column value is like SET but with an implied order on the values.
- RANGE (MeasureType) : Indicates that the column value lies between two points on a scale.
- SET (MeasureType) : Indicates that the column value is one of a (small) set of discrete values.
- TYPELESS (MeasureType) : Indicates that the column can have any value compatible with its storage.

```
getEnum(name) : MeasureType
```

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

`getValues() : MeasureType[]`

Returns an array containing all the valid values for this enumeration class.

## `MissingValueDefinition` Objects

This defines the attributes associated with missing values.

`m.getValueCount() : int`

Returns the number of values specified as missing in this definition, excluding `None` and whitespace values. The result is the length of the list returned by `getValues`.

`m.getValues() : List`

Returns the list of values specified as missing in this definition. The list cannot be modified; the mutator methods on the list throw `UnsupportedOperationException`.

`m.isEnabled() : boolean`

Returns `True` if this definition is enabled. A definition which is not enabled recognizes no values as missing.

`m.isNullIncluded() : boolean`

Returns `True` if the `None` value is recognized as missing by this definition.

`m.isWhitespaceIncluded() : boolean`

Returns `True` if whitespace values are recognized as missing by this definition.

## `ModelOutputMetadata` Objects

This defines the metadata associated with a Column generated by a model.

Constants:

- `ADJUSTED (string) :` Used by binary classifiers as the tag value indicating that the column indicates adjusted propensity.
- `RAW (string) :` Used by binary classifiers as the tag value indicating that the column indicates raw propensity.
- `X (string) :` Used by Kohonen models as the tag value indicating the X component of the cluster id.
- `Y (string) :` Used by Kohonen models as the tag value indicating the Y component of the cluster id.

`m.getRole() : ModelFieldRole`

Returns the model output column role for the column if it is a model output column or `None`.

`m.getTag() : string`

Returns the tag associated with this model output column or `None` if the column is not a model output column or the tag has not been defined.

`m.getTargetColumn() : string`

Returns the target column name associated with this model output column or `None` if the column is not a model output column or the target column has not been defined.

`m.getValue() : Object`

Returns the value associated with this model output column or `None` if the column is not a model output column or no value has been defined. The value is typically used in combination with the target column e.g. when a model can generate probabilities for each value specified by the target field.

## ModelingRole Objects

Subclass of Enum.

This class enumerates the valid modeling roles for columns in a DataModel.

Constants:

- BOTH (ModelingRole) : Indicates that this column can be either an antecendent or a consequent.
- FREQ_WEIGHT (ModelingRole) : Indicates that this column is used to be as frequency weight but won't be showed for user
- IN (ModelingRole) : Indicates that this column is a predictor or an antecendent.
- NONE (ModelingRole) : Indicates that this column is not used directly during modeling.
- OUT (ModelingRole) : Indicates that this column is predicted or a consequent.
- PARTITION (ModelingRole) : Indicates that this column is used to identify the data partition.
- RECORD_ID (ModelingRole) : Indicates that this column is used to indentify the record id.
- SPLIT (ModelingRole) : Indicates that this column is used to split the data.

```
getEnum(name) : ModelingRole
```

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : ModelingRole[]
```

Returns an array containing all the valid values for this enumeration class.

## RowSet Objects

This defines the properties of a tabular dataset.

```
r.getColumnClass(columnIndex) : Class
```

columnIndex (int) : the column index

Returns the class of objects in a specified column of this dataset. The column index must be in the range:

```
0 <= index < getColumnCount()
```

Exceptions:

IndexOutOfBoundsException : unless the column index is in range

```
r.getColumnCount() : int
```

Returns the number of columns in this dataset.

```
r.getColumnName(columnIndex) : string
```

columnIndex (int) : the column index

Returns the name of a specified column in this dataset. Returns the empty string if the column does not have a name. The column index must be in the range:

```
0 <= index < getColumnCount()
```

Exceptions:

IndexOutOfBoundsException : unless the column index is in range

```
r.getRowCount() : int
```

Returns the number of rows in this dataset.

```
r.getValueAt(rowIndex, columnIndex) : Object
```

`rowIndex (int) :` the row index

`columnIndex (int) :` the column index

Returns the value from a specified row and column in this dataset. The row and column indexes must be in the range:

```
0 <= rowIndex < getRowCount() && 0 <= columnIndex < getColumnCount()
```

The returned value will either be `None` or an instance of the class returned by `getColumnClass` for the same column index.

Exceptions:

`IndexOutOfBoundsException :` unless the row and column indexes are in range

## `StorageType` Objects

Subclass of `Enum`.

This class enumerates the valid storage types for columns in a `DataModel` or parameters.

Constants:

- `DATE (StorageType) :` Indicates that the storage type is a date type.
- `INTEGER (StorageType) :` Indicates that the storage type is an integer.
- `REAL (StorageType) :` Indicates that the storage type is a floating point number.
- `STRING (StorageType) :` Indicates that the storage type is a string.
- `TIME (StorageType) :` Indicates that the storage type is a time type.
- `TIMESTAMP (StorageType) :` Indicates that the storage type is a combined time and date type.
- `UNKNOWN (StorageType) :` Indicates that the storage type is unknown.

```
getEnum(name) : StorageType
```

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or `None` if no enumeration exists for the supplied name.

```
getValues() : StorageType[]
```

Returns an array containing all the valid values for this enumeration class.

## `UnknownColumnException` Objects

Subclass of `DataModelException`.

An `UnknownColumnException` is thrown when a column name is used that does not exist in the data model.

No message string is set for this exception.

```
u.getColumnName() : string
```

Returns the name of the unknown column.

# Chapter 4. com.spss.psapi.extensions.common

This provides definitions that are exposed to CLEF implementors using the CLEF Java API.

## DocumentOutputManager Objects

A document output manager contains all document outputs managed by the application session.

```
d.addDocumentOutput(documentOutput, source)
```

documentOutput (DocumentOutput) : the document output to be added

source (Object) : the object that generated or supplied the document output (may be None)

Adds the supplied document output object to the manager.

```
d.getOutputAt(index) : BuiltObject
index (int) :
```

Returns the document or interactive output at the specified index.

```
d.getOutputCount() : int
```

Returns the number of outputs being managed.

```
d.getOutputForID(id) : BuiltObject
```

id (string) : the output ID

Returns the document or interactive output with the specified ID or None if no such output can be found.

```
d.removeDocumentOutput(documentOutput)
```

documentOutput (DocumentOutput) : the document output to be removed

Remove the supplied document output object from the manager.

## ManagedSession Objects

Subclass of SystemSession.

A managed session provides access to Modeler manager objects that manage streams, models and outputs.

```
m.getDocumentOutputManager() : DocumentOutputManager
```

Returns the document output manager for this session.

```
m.getModelOutputManager() : ModelOutputManager
```

Returns the model output manager for this session.

```
m.getProcessorStreamManager() : StreamManager
```

Returns the stream manager for this session.

## ModelOutputManager Objects

A model output manager contains all model outputs managed by the application session.

```
m.addModelOutput(modelOutput, source)
```

`modelOutput (ModelOutput) :` the model output to be added

`source (Object) :` the object that generated or supplied the model output (may be `None`)

Adds the supplied model output object to the manager.

```
m.getModelOutputAt(index) : ModelOutput
index (int) :
```

Returns the model output at the specified index.

```
m.getModelOutputCount() : int
```

Returns the number of model outputs being managed.

```
m.getModelOutputForID(id) : ModelOutput
```

`id (string) :` the model output ID

Returns the model output with the specified ID or `None` if no such model output can be found.

```
m.removeModelOutput(modelOutput)
```

`modelOutput (ModelOutput) :` the model output to be removed

Remove the supplied model output object from the manager.

## StreamManager Objects

A stream manager contains all streams being managed by the session.

```
s.getCurrentProcessorDiagram() : Diagram
```

Returns the current diagram or `None` if there are no streams currently being managed. The current diagram will always be the same object as, or a child of, the current stream.

```
s.getCurrentProcessorStream() : Stream
```

Returns the current stream or `None`. Note that the current stream may not be the same as the current diagram.

```
s.getProcessorStreamAt(index) : Stream
index (int) :
```

Returns the stream at the specified index.

```
s.getProcessorStreamCount() : int
```

Returns the number of streams being managed.

```
s.getProcessorStreamForID(id) : Stream
```

`id (string) :` the stream ID

Returns the stream with the specified ID or `None` if no such stream can be found.

# Chapter 5. Model information

This provides support for accessing information about data mining models.

## `CompositeModelDetail` Objects

Subclass of `ModelDetail`.

This interface encapsulates the representation detail of a composite model.

```
c.canUseMultipleModels() : boolean
```

Returns `True` if multiple models can be used by the composite model.

```
c.getIndividualModelResults() : Iterator
```

Returns the individual model results of this composite model.

## `ModelDetail` Objects

This interface encapsulates the representation detail of a data mining model.

```
m.getAlgorithmName() : string
```

Returns the name of the model builder algorithm.

```
m.getApplicationName() : string
```

Returns the name of the model builder application.

```
m.getApplicationVersion() : string
```

Returns the version of the model builder application.

```
m.getBuildDate() : Date
```

Returns the date this model was built.

```
m.getCopyright() : string
```

Returns the model copyright.

```
m.getInputDataModel() : DataModel
```

Returns the input data model required by the model. Note that the `ModelingRole` for each field in the input data model is `ModelingRole.IN`, even for fields that had been specified as `ModelingRole.BOTH` for association or sequence models.

```
m.getModelID() : string
```

Returns the model ID. For models within a composite model, it is assumed that each model has a unique ID.

```
m.getModelType() : ModelType
```

Returns the type of the model.

```
m.getOutputDataModel() : DataModel
```

Returns the output data model produced by the model. Note that the `ModelingRole` for each field in the output data model is `ModelingRole.OUT`, even for fields that had been specified as `ModelingRole.BOTH` for association or sequence models.

It is also important to note that this is not same as the output column set produced by the ModelApplier that applies the model to data. The transformer output column set will usually include additional fields from the input and may include property settings that modify the number or type of outputs produced (for example, a transformer that applies a clustering model may produce the cluster ID as an integer or a string depending on the property settings).

m.getPMMLModelType() : PMMLModelType

Returns the PMML type of the model, or None if the model does not correspond to any PMML type.

m.getUserName() : string

Returns the name of the account used to build the model.

m.isOutputColumnAuxiliary(name) : boolean

name (string) : the column name

Returns True if the supplied column name is an auxiliary column in the output data model or False. An auxiliary output column is a column that provides additional information about the output of the model e.g., the confidence of the prediction or distance from the cluster center.

m.getPMMLText() : string

For PMML-based models, this returns a string representing the PMML generated from the model building algorithm. For non-PMML models, this returns None.For PMML-based models, this returns a string representing the PMML generated from the model building algorithm. For non-PMML models, this returns None.

m.isPMMLModel() : boolean

Returns True if this model is represented using PMML, False otherwise.

## ModelType Objects

Subclass of Enum.

This class enumerates the valid model types. These are largely based on the JDM 1.1 and draft JDM 2.0 data mining functions definition.

Constants:

- ANOMALY_DETECTION (ModelType) : Indicates an anomaly detection model.
- APPROXIMATION (ModelType) : Indicates that the model predicts a continous value.
- ASSOCIATION (ModelType) : Indicates that the model identifies frequently occurring sets of items.
- ATTRIBUTE_IMPORTANCE (ModelType) : Indicates an attribute importance model.
- CATEGORIZE (ModelType) : Indicates a text categorize model.
- CLASSIFICATION (ModelType) : Indicates that the model predicts a discrete value.
- CLUSTERING (ModelType) : Indicates that the model groups similar rows of data together.
- CONCEPT_EXTRACTION (ModelType) : Indicates that the model identifies concepts from textual data.
- REDUCTION (ModelType) : Indicates that the model reduces the complexity of data.
- SEQUENCE (ModelType) : Indicates that the model identifies frequently occurring sequences of events.
- SUPERVISED_MULTITARGET (ModelType) : Indicates a supervised multi-target model.
- SURVIVAL_ANALYSIS (ModelType) : Indicates a survival analysis model.
- TIME_SERIES (ModelType) : Indicates a time series model.
- UNKNOWN (ModelType) : Indicates that the model type cannot be determined. This value should never be returned for any built-in model.

`getEnum(name) : ModelType`

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or `None` if no enumeration exists for the supplied name.

`getValues() : ModelType[]`

Returns an array containing all the valid values for this enumeration class.

`m.isUnknown() : boolean`

Returns `True` if this value is `ModelType.UNKNOWN`.

## `PMMLModelType` Objects

Subclass of `Enum`.

Enumerates the model types specified by PMML 4.0.

Constants:

- `ASSOCIATION (PMMLModelType) :` PMML AssociationModel
- `CLUSTERING (PMMLModelType) :` PMML ClusteringModel
- `GENERAL_REGRESSION (PMMLModelType) :` PMML GeneralRegressionModel
- `MINING (PMMLModelType) :` PMML MiningModel
- `NAIVE_BAYES (PMMLModelType) :` PMML NaiveBayesModel
- `NEURAL_NETWORK (PMMLModelType) :` PMML NeuralNetwork
- `REGRESSION (PMMLModelType) :` PMML RegressionModel
- `RULE_SET (PMMLModelType) :` PMML RuleSetModel
- `SEQUENCE (PMMLModelType) :` PMML SequenceModel
- `SUPPORT_VECTOR_MACHINE (PMMLModelType) :` PMML SupportVectorMachineModel
- `TEXT (PMMLModelType) :` PMML TextModel
- `TIME_SERIES (PMMLModelType) :` PMML TimeSeriesModel
- `TREE (PMMLModelType) :` PMML TreeModel

`getEnum(name) : PMMLModelType`

`name (string) :` the PMML model type name

Returns the `PMMLModelType` corresponding to the specified PMML model type name, or `None` if there is no such model type. The name must be one specified by PMML 4.0.

`getValues() : PMMLModelType[]`

Returns an array containing all the valid values for this enumeration class.

`p.isUnknown() : boolean`

Returns `False` always. All the values in this class represent known model types; there is no "unknown".

# Chapter 6. Server resources

This provides access to server-side resources such as the server file system and database connections.

## ServerDatabaseConnection Objects

This encapsulates the functionality of an object that represents a connection to a database.

```
s.dropTable(tableName)
tableName (string) :
s.getDBQueryColumns(queryText) : RowSet
```

queryText (string) : the text of the query

Returns a list of the columns returned by a specified database query. The list is returned in the form of a dataset where the first column contains the column name as a string.

Exceptions:

ServerResourceException : if the server resource cannot be accessed

```
s.getDBTableColumns(catalogName, schemaName, tableName) : RowSet
```

catalogName (string) : the catalog name

schemaName (string) : the schema name

tableName (string) : the base table name

Returns a list of the columns in a specified database table. The list is returned in the form of a dataset where the first column contains the column name as a string and the second column contains StorageType constants that provide the closest match for the storage types for the columns in the database table.

Exceptions:

ServerResourceException : if the server resource cannot be accessed

```
s.getDBTables(catalogName, schemaName, tableName, includeUserTables, includeSystemTables, includeViews,
includeSynonyms) : RowSet
```

catalogName (string) : pattern for the catalog name

schemaName (string) : pattern for the schema name

tableName (string) : pattern for the table name

includeUserTables (boolean) : indicates whether user tables should be included

includeSystemTables (boolean) : indicates whether system tables should be included

includeViews (boolean) : indicates whether views should be included

includeSynonyms (boolean) : indicates whether synonyms should be included

Returns a summary of the tables available on this database connection which match the specified arguments. The summary is returned in the form of a dataset with at least two columns, of which the first two columns are the schema name and table name, both as strings.

Exceptions:

`ServerResourceException` : if the server resource cannot be accessed

```
s.isDBTableVisible(catalogName, schemaName, tableName) : boolean
```

`catalogName (string)` : the catalog name

`schemaName (string)` : the schema name

`tableName (string)` : the table name

Returns `True` if the specified table is visible to this connection.

Exceptions:

`ServerResourceException` : if the server resource cannot be accessed

## `ServerFile` Objects

This encapsulates the representation of a file on the server file system. A server file always uses / to separate directory paths since this is valid for both UNIX and Windows file systems.

```
s.getName() : string
```

Returns the name of the file.

```
s.getParent() : string
```

Returns the pathname of the parent directory.

```
s.getParentServerFile() : ServerFile
```

Returns a file representing the parent directory.

```
s.getPath() : string
```

Returns the pathname of the file.

```
s.isAbsolute() : boolean
```

Returns whether the pathname is absolute.

```
s.isDirectory() : boolean
```

Returns whether this file is a directory.

## `ServerFileSystem` Objects

This encapsulates the functionality for representing the file system on the data mining server's host.

```
s.createTemporaryFile(name) : ServerFile
```

`name (string)` : the suggested name of the new file

Creates a new file with a unique name in the server's temporary file space. The file is empty and can be written by the creator. The specified name will be used as the starting point for generating a unique name for the file, and if the name has an extension it will be left unchanged. The file must be deleted when it is no longer needed.

Exceptions:

`ServerResourceException` : if the file cannot be created for any reason

```
s.deleteFile(file)
```

`file (ServerFile)` : the file to delete

Deletes the specified file from this file system. The caller must have permission to delete the file.

Calling this method has the same effect as creating and running a delete file task.

Exceptions:

`ServerResourceException` : if the file cannot be deleted

```
s.directoryOrFileExists(path, isDirectory) : boolean
```

`path (ServerFile)` : Pathname to check

`isDirectory (boolean)` : Set to True if the path to be tested should be a directory

Returns whether a directory or file exists.

Exceptions:

`ServerResourceException` : if the server file system cannot be accessed

```
s.exists(file) : boolean
file (ServerFile) :
```

Returns whether a file exists.

Exceptions:

`ServerResourceException` : if the server file system cannot be accessed

```
s.getAbsolutePath(file) : string
```

`file (ServerFile)` : a `ServerFile` object

Returns the absolute pathname or `None` if the absolute path cannot be determined.

Exceptions:

`ServerResourceException` : if the server file system cannot be accessed

```
s.getAbsoluteServerFile(file) : ServerFile
```

`file (ServerFile)` : a `ServerFile` object

Returns a file representing the absolute pathname or `None` if the absolute path cannot be determined.

Exceptions:

ServerResourceException : if the server file system cannot be accessed

```
s.getChild(parent, filename) : ServerFile
```

parent (ServerFile) : a ServerFile object representing a directory or special folder

filename (string) : a name of a file or folder which exists in parent

Returns the named child in the supplied parent.

Exceptions:

ServerResourceException : if the server file system cannot be accessed

```
s.getDefaultDirectory() : ServerFile
```

Returns the server's default directory.

Exceptions:

ServerResourceException : if the server file system cannot be accessed

```
s.getFiles(directory) : ServerFile[]
```

directory (ServerFile) : the directory

Returns the list of files in the specified directory of the file system.

Exceptions:

ServerResourceException : if the server file system cannot be accessed

```
s.getParentDirectory(file) : ServerFile
```

file (ServerFile) : a ServerFile object

Returns the parent directory of a file in the file system.

```
s.getPathSeparator() : string
```

Returns a string version of the path separator character.

```
s.getPathSeparatorChar() : char
```

Returns the path separator character for this file system. On Windows, this is ; and on UNIX, this is :.

```
s.getRoots() : ServerFile[]
```

Returns the roots of the file system.

Exceptions:

ServerResourceException : if the server file system cannot be accessed

```
s.getSeparator() : string
```

Returns a string version of the separator character.

```
s.getSeparatorChar() : char
```

Returns the separator character for this file system. On Windows, this is \ and on UNIX, this is /.

```
s.getServerFile(filename) : ServerFile
```

`filename (string) :` a name of a file or folder

Returns a server file for the corresponding file name.

`s.getSize(file) : long`

`file (ServerFile) :` the file

Returns the size, in bytes, of the specified file in this file system. Returns `0` if the file does not exist or is not accessible, or `-1` if the file is accessible but the server does not support the file size operation. The result is undefined when the file denotes a directory.

Exceptions:

`ServerResourceException :` if the file system cannot be accessed

---

## `ServerResourceException` Objects

Subclass of `ModelerException`.

An exception thrown while accessing a server resource.

A message string is always created for this exception.

# Chapter 7. Server resources

This provides support for the construction and use of data mining sessions.

## LocaleInfo Objects

This interface defines locale-sensitive information associated with a Session.

```
l.getLocale() : Locale
```

Returns the locale associated with this locale information object.

```
l.getLocalizedProcessorDescription(nodeType) : string
```

nodeType (string) : the node type

Returns a locale-sensitive string describing the supplied node type name or None if a description cannot be found.

```
l.getLocalizedProcessorName(nodeType) : string
```

nodeType (string) : the node type name

Returns a locale-sensitive string representing the supplied node type name or None if a name cannot be found.

## Repository Objects

This defines the basic functionality for a content repository. Unlike the file persistence tasks defined by TaskFactory, these are executed synchronously through the repository object rather than indirectly via the Session.

```
r.createFolder(parentFolder, newFolder) : string
```

parentFolder (string) : the path to the parent folder

newFolder (string) : the new folder name

Creates a new folder with the specified name.

Exceptions:

SessionException : if the folder cannot be created

```
r.createRetrieveURI(path, version, label) : URI
```

path (string) : the full path to the object to be retrieved

version (string) : the version marker or None

label (string) : the label or None

A utility function to create a repository URI that is valid for retrieving an object from the specified location. Either a version or label may be specified if a specific version is required. If both are None then the LATEST version will be returned.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

```
r.createStoreURI(path, label) : URI
```

path (string) : the full path to the store location

label (string) : the label to be applied to the object or None

A utility function to create a repository URI that is valid for storing an object at the specified location and with an optional label to be applied to the object when it is stored.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

```
r.deleteFolder(folder)
```

folder (string) : the folder to be deleted

Deletes the specified folder and any content within it.

Exceptions:

SessionException : if the folder cannot be deleted

```
r.getRepositoryHandle() : Object
```

Returns the underlying repository handle. The handle will be an instance of the Java class com.spss.repository.client.application.Repository.

```
r.renameFolder(folder, newName)
```

folder (string) : the path of the folder to be renamed

newName (string) : the new folder name

Renames the specified folder.

Exceptions:

SessionException : if the folder cannot be renamed

```
r.retrieveDocument(path, version, label, autoManage) : DocumentOutput
```

path (string) : the full path to the object

version (string) : the version marker or None

label (string) : the label or None

autoManage (boolean) : whether the document should be added to the output manager

Retrieves a document output from the specified path. Either a version or label may be specified if a specific version is required. If both are None then the LATEST version is returned. Code that needs to open documents privately without having them made visible to the user should set the autoManage flag to False.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the document output cannot be retrieved for some reason
r.retrieveModel(path, version, label, autoManage) : ModelOutput

path (string) : the full path to the object

version (string) : the version marker or None

label (string) : the label or None

autoManage (boolean) : whether the model should be added to the model manager

Retrieves a model output from the specified path. Either a version or label may be specified if a specific version is required. If both are None then the LATEST version is returned. Code that needs to open models privately without having them made visible to the user should set the autoManage flag to False.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the model output cannot be retrieved for some reason
r.retrieveProcessor(path, version, label, diagram) : Node

path (string) : the full path to the object

version (string) : the version marker or None

label (string) : the label or None

diagram (Diagram) : the diagram that the node should be added to

Retrieves a node from the specified path and inserts it into the supplied diagram. Either a version or label may be specified if a specific version is required. If both are None then the LATEST version is returned.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the node cannot be retrieved for some reason
r.retrieveStream(path, version, label, autoManage) : Stream

path (string) : the full path to the object

version (string) : the version marker or None

label (string) : the label or None

autoManage (boolean) : whether the stream should be added to the stream manager

Retrieves a stream from the specified path. Either a version or label may be specified if a specific version is required. If both are None then the LATEST version is returned. Code that needs to open streams privately without having them made visible to the user should set the autoManage flag to False.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the stream cannot be retrieved for some reason

```
r.storeDocument(documentOutput, path, label) : string
```

documentOutput (DocumentOutput) : the document output to be stored

path (string) : the path

label (string) : the label or None

Stores a document output to the specified location. If the label is provided then it is applied to the new version.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the document output cannot be stored for some reason

```
r.storeModel(modelOutput, path, label) : string
```

modelOutput (ModelOutput) : the model output to be stored

path (string) : the path

label (string) : the label or None

Stores a model output to the specified location. If the label is provided then it is applied to the new version.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the model output cannot be stored for some reason

```
r.storeProcessor(node, path, label) : string
```

node (Node) : the node to be stored

path (string) : the path

label (string) : the label or None

Stores a node to the specified location. If the label is provided then it is applied to the new version.

Exceptions:

URISyntaxException : if the method cannot construct a valid URI

SessionException : if the node cannot be stored for some reason

```
r.storeStream(stream, path, label) : string
```

stream (Stream) : the stream to be stored

path (string) : the path

`label (string) :` the label or `None`

Stores a stream to the specified location. If the label is provided then it is applied to the new version.

Exceptions:

`URISyntaxException :` if the method cannot construct a valid URI

`SessionException :` if the stream cannot be stored for some reason

## `Session` Objects

Subclass of `SystemSession`.

A session is the main interface through which users of the Modeler API access the API features. Each session has its own connection to a data mining server.

A data mining server is typically a remote server identified by a `ServerConnectionDescriptor`. From Modeler API 2.0 a server can also be a local server instance spawned as a child process of the Modeler API host process (see `connect()`). This local server mode requires a local server installation, such as a Modeler client installation, on the local machine. The installation directory is located through the system property

`com.spss.psapi.session.serverInstallationDirectory`

This property must point to a valid installation directory and must be set before the session factory is instantiated otherwise connections to the local server will fail.

`s.close()`

Closes this session. This automatically interrupts any task currently running, closes any current `Stream` and invalidates any other objects created by this session.

`s.connect(serverDescriptor)`

`serverDescriptor (ServerConnectionDescriptor) :` identifies a remote data mining server

Connects this session to the remote server identified by the specified server descriptor.

Exceptions:

`ServerConnectionException :` if a connection to the server cannot be established or if the Session already has a connection

`s.connect()`

Connects this session to a local server instance. Spawns a new local server process.

Exceptions:

`ServerConnectionException :` if a local server instance cannot be created or if the session already has a connection

`s.connect(stream)`

`stream (Stream) :` the stream to be connected

Connects the stream to the session's server.

Exceptions:

ServerConnectionException : if the Session is not connected or if the stream is already connected

OwnerException : if the stream was not created by this session

```
s.createServerDatabaseConnection(datasourceName, userName, password, catalogName) :
  ServerDatabaseConnection
```

datasourceName (string) : the datasource name

userName (string) : the user name

password (string) : the password

catalogName (string) : the catalog name

Creates a ServerDatabaseConnection. The datasource name must visible to the data mining server.

Exceptions:

ServerConnectionException : if the session is not connected to a server

ServerResourceException : if the connection task fails.

```
s.getASCredentialDescriptor() : ASCredentialDescriptor
```

Returns the credential descriptor used to log in to the Analytic Server.

```
s.getAttribute(name) : Object
```

name (string) : an attribute name

Returns the value of the specified attribute in this session, or None if there is no such attribute. Attributes are created by the application, not by the Modeler API.

```
s.getAttributeNames() : Collection
```

Returns the names of the attributes defined in this session.

```
s.getParameters() : ParameterProvider
```

Returns the parameter provider for this session. The parameter provider provides access to session parameters.

```
s.getRepository() : Repository
```

Returns the repository object that provides simple mechanisms for storing and retrieving objects.

```
s.getRepositoryConnectionDescriptor() : RepositoryConnectionDescriptor
```

Returns the repository connection descriptor.

```
s.getServerConnectionDescriptor() : ServerConnectionDescriptor
```

Returns the ServerConnectionDescriptor used to connect this session to a remote server. Returns None if the session is connected to a local server or has not yet been connected.

```
s.getServerDataSourceNames() : RowSet
```

Returns a row set that lists the available system DSNs visible on the data mining server host. The list is returned in the form of a row set where the first column contains the data source name as a string and the second column contains a string description.

Exceptions:

`ServerConnectionException` : if the session is not connected to a server

`ServerResourceException` : if access to the server data sources was denied

`s.getServerFileSystem() : ServerFileSystem`

Returns the server file system.

Exceptions:

`ServerConnectionException` : if the session is not connected to a server

`ServerResourceException` : if access to the server file system was denied

`s.getServerVersionInfo() : ServerVersionInfo`

Returns information about the connected server version or `None` if the session is not connected.

`s.getTaskFactory() : TaskFactory`

Returns the `TaskFactory` for this session.

`s.getTaskRunner() : TaskRunner`

Returns the `TaskRunner` for this session.

`s.getUIResources() : UIResources`

Returns the interface that provides access to UI-related resources.

`s.interrupt()`

Interrupts any synchronous task which is currently executing. The call returns immediately while the task is interrupted. It has no effect if the session is not busy.

This method may be called from any thread.

`s.isBusy() : boolean`

Returns `True` if this session is currently executing a synchronous task (even if the task is in the process of being interrupted).

This method may be called from any thread.

`s.isClosed() : boolean`

Returns `True` if this has been closed.

`s.isConnected() : boolean`

Returns `True` if this session has a server connection.

`s.isValidASCredentialDescriptor(asConnectionDescriptor) : boolean`

`asConnectionDescriptor (ASCredentialDescriptor)` : the credential descriptor

Check whether the credential descriptor is a valid.

`s.isValidRepositoryConnectionDescriptor(descriptor) : boolean`

`descriptor (RepositoryConnectionDescriptor)` : the repository connection descriptor

Returns `True` if the supplied repository connection descriptor is valid i.e. can connect to a content repository.

`s.publish(node) : ExecutionHandle`

node (DataWriter) : the DataWriter to be published

Executes the specified DataWriter in publish mode to obtain a PublishedImage. This overrides the execution mode set in the node and retrieves the published image from the server on completion. Execution is synchronous. The result is an ExecutionHandle which can be used to determine the exit status of the task and, if the task completes successfully, to obtain the result of the task which is a PublishedImage.

Exceptions:

OwnerException : if the node was not created by this session

ObjectLockedException : if the node or its containing stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running another task, or cannot execute the task for some other reason, or if execution completes in a state other than SUCCESS

```
s.publish(node, inline) : ExecutionHandle
```

node (TerminalNode) : the TerminalNode to be published. Must be a DataWriter if not publishing inline

inline (boolean) : True to prepare the image for inline scoring

Executes the specified node in publish mode to obtain a PublishedImage ready for inline scoring.

If inline is False, this is equivalent to calling publish(DataWriter) and the node must be a data writer. Otherwise, the node may be any terminal node and the stream is modified in place to replace the input and output nodes with ones suitable for inline scoring.

Execution is synchronous. The result is an ExecutionHandle which can be used to determine the exit status of the publishing task and, if the task completes successfully, to obtain the result of the task which is a PublishedImage.

Exceptions:

OwnerException : if the node was not created by this session

ObjectLockedException : if the node or its containing stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running another task, or cannot execute the task for some other reason, or if the stream cannot be prepared for inline scoring, or if execution completes in a state other than SUCCESS

```
s.removeAttribute(name)
```

name (string) : an attribute name

Deletes the specified attribute from this session. If there is no such attribute, the call has no effect.

```
s.run(stream, results) : ExecutionHandle
```

stream (Stream) : the Stream to be executed

`results (Collection) :` an empty collection that will contain any built objects once execution has completed

Executes the supplied stream synchronously and waits for it to complete. Returns an `ExecutionHandle` which can be used to access the exit status and any result from the task.

Exceptions:

`OwnerException :` if the stream was not created by this session

`ObjectLockedException :` if the stream is locked

`ServerConnectionException :` if the stream is not connected to a server

`SessionException :` if the session is already running another task, cannot execute the task or if execution completes in a state other than `SUCCESS`

```
s.run(nodes, results) : ExecutionHandle
```

`nodes (Node[]) :` the array of `Node` objects to be executed

`results (Collection) :` an empty collection that will contain any built objects once execution has completed

Executes the supplied array of nodes synchronously and waits for them to complete. There must be at least one node in the array. Returns an `ExecutionHandle` which can be used to access the exit status and any result from the task.

Exceptions:

`OwnerException :` if the nodes' stream was not created by this session or the nodes are not all owned by the same stream

`ObjectLockedException :` if the nodes' owner stream is locked

`ServerConnectionException :` if the nodes' stream is not connected to a server

`SessionException :` if the session is already running another task, cannot execute the task or if execution completes in a state other than `SUCCESS`

`IllegalArgumentException :` if the array is empty

```
s.runTask(task) : ExecutionHandle
```

`task (Task) :` the `Task` to be executed

Executes the supplied task synchronously and waits for it to complete. Returns an `ExecutionHandle` which can be used to access the exit status and any result from the task.

Exceptions:

`OwnerException :` if the task was not created by this session's `TaskFactory`

`ObjectLockedException :` if the task is already executing or any object referenced by the task is locked for updating

`SessionException :` if the session cannot execute the task or if execution completes in a state other than `SUCCESS`

```
s.setASCredentialDescriptor(asConnectionDescriptor)
```

asConnectionDescriptor (ASCredentialDescriptor) : the credential descriptor

Sets the credential descriptor used to log in Analytic Server.

```
s.setAttribute(name, value)
```

name (string) : an attribute name

value (Object) : the attribute value

Assigns a value to the specified attribute in this session. If the attribute already has a value, it is replaced. If the value is None, the attribute is deleted from the session.

```
s.setRepositoryConnectionDescriptor(descriptor)
```

descriptor (RepositoryConnectionDescriptor) : the new repository connection descriptor

Sets the repository connection descriptor to be used when a connection to a content repository is required. Any existing connection is closed although a new connection will not be created until it is required.

```
s.spawn(stream, results) : ExecutionHandle
```

stream (Stream) : the Stream to be executed

results (Collection) : an empty collection that will contain any built objects once execution has completed

Executes the supplied stream asynchronously. Returns an ExecutionHandle which can be used to monitor and control the progress of the task.

Exceptions:

OwnerException : if the stream was not created by this session

ObjectLockedException : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

```
s.spawn(nodes, builtObjects) : ExecutionHandle
```

nodes (Node[]) : the array of Node objects to be executed

builtObjects (Collection) : an empty collection that will be populated by built objects created by the execution

Executes the supplied array of nodes asynchronously. Returns an ExecutionHandle which can be used to monitor and control the progress of the task. There must be at least one node in the array.

Exceptions:

OwnerException : if the nodes' stream was not created by this session or the nodes are not all owned by the same stream

ObjectLockedException : if the nodes' stream is locked

ServerConnectionException : if the nodes' stream is not connected to a server

IllegalArgumentException : if the array is empty

s.spawnPublish(node) : ExecutionHandle

node (DataWriter) : the DataWriter to be published

Executes the specified DataWriter asynchronously in publish mode to obtain a PublishedImage. This overrides the execution mode set in the node and retrieves the published image from the server on completion. Returns an ExecutionHandle which can be used to monitor and control the progress of the publishing task and, if it completes successfully, to obtain the task result which is a PublishedImage.

Exceptions:

OwnerException : if the node was not created by this session

ObjectLockedException : if the node or its containing stream is locked

ServerConnectionException : if the stream is not connected to a server

s.spawnPublish(node, inline) : ExecutionHandle

node (TerminalNode) : the TerminalNode to be published. Must be a DataWriter if not publishing inline

inline (boolean) : True to prepare the image for inline scoring

Executes the specified node asynchronously in *publish* mode to obtain a PublishedImage ready for inline scoring.

If inline is False, this is equivalent to calling publish(DataWriter) and the node must be a data writer. Otherwise, the node may be any terminal node and the stream is modified in place to replace the input and output nodes with ones suitable for inline scoring.

Returns an ExecutionHandle which can be used to monitor and control the progress of the publishing task and, if it completes successfully, to obtain the task result which is a PublishedImage.

Exceptions:

OwnerException : if the node was not created by this session

ObjectLockedException : if the node or its containing stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the stream cannot be prepared for inline scoring

s.spawnTask(task) : ExecutionHandle

task (Task) : the Task to be executed

Executes the supplied task asynchronously. Returns an ExecutionHandle which can be used to monitor and control the progress of the task.

Exceptions:

OwnerException : if the task was not created by this session's TaskFactory

ObjectLockedException : if the task is already executing or any object referenced by the task is locked for updating

```
s.waitForAllTasksToFinish()
```

Waits for all tasks in progress on this session to complete.

## SessionException Objects

Subclass of ModelerException.

An exception thrown while creating a Session or performing a task within a Session.

A message string is always created for this exception.

## SystemSession Objects

Defines the functionality of a system session. Sessions define the basic context that other operations work within.

```
s.getLocale() : Locale
```

Returns the locale specified for this session.

```
s.getLocaleInfo() : LocaleInfo
```

Returns the LocaleInfo for this session.

```
s.isLocalSession() : boolean
```

Returns True if the session is connected to a local or desktop execution engine, False otherwise.

## UIResources Objects

This interface provides access to UI-related resources.

```
u.getBaseProcessorIcon(nodeType) : ImageIcon
```

nodeType (string) : the node type

Returns the base icon used to represent this node type or None. The base icon does not include any background.

```
u.getProcessorIcon(nodeType) : ImageIcon
```

nodeType (string) : the node type

Returns the default icon used to represent this node type or None.

# Chapter 8. Tasks and execution

This provides objects that create and represent data mining tasks.

## ExecutionFeedbackEvent Objects

Feedback received from an execution task. Feedback is only received from tasks executing a Node or Stream.

Constants:

- MESSAGE_ID_EXECUTION_STARTED (int) : Message ID: execution started.
- MESSAGE_ID_EXECUTION_STOPPED (int) : Message ID: execution stopped.
- MESSAGE_ID_INTERRUPTED (int) : Message ID: interrupted.
- MESSAGE_ID_OPTIMIZATION_STARTED (int) : Message ID: optimization started.
- MESSAGE_ID_OPTIMIZATION_STOPPED (int) : Message ID: optimization stopped.
- MESSAGE_ID_OTHER (int) : Message ID: other.
- MESSAGE_ID_PREPARATION_STARTED (int) : Message ID: preparation started.
- MESSAGE_ID_PREPARATION_STOPPED (int) : Message ID: preparation stopped.
- MESSAGE_ID_PROCESS_EXECUTION_STARTED (int) : Message ID: external process execution started.
- MESSAGE_ID_PROCESS_EXECUTION_STOPPED (int) : Message ID: external process execution stopped.
- MESSAGE_ID_SQL_EXECUTION_STARTED (int) : Message ID: SQL execution started.
- MESSAGE_ID_SQL_EXECUTION_STOPPED (int) : Message ID: SQL execution stopped.
- SEVERITY_ERROR (int) : Severity level: error.
- SEVERITY_INFORMATION (int) : Severity level: information.
- SEVERITY_WARNING (int) : Severity level: warning.
- TYPE_DIAGNOSTIC (int) : Event type: diagnostic message.
- TYPE_PROGRESS (int) : Event type: percentage progress.
- TYPE_RECORD_COUNT (int) : Event type: record count.

```
createExecutionFeedbackEvent(handle, type, severity, message) : ExecutionFeedbackEvent
```

handle (ExecutionHandle) : the ExecutionHandle which originated the event

type (int) : the type of event

severity (int) : the severity level of the event

message (string) : the message associated with the event

Creates a new execution feedback event. getMessageId() returns MESSAGE_ID_OTHER.

```
createExecutionFeedbackEvent(handle, type, severity, messageId, message) : ExecutionFeedbackEvent
```

handle (ExecutionHandle) : the ExecutionHandle which originated the event

type (int) : the type of event

severity (int) : the severity level of the event

messageId (int) : the message ID associated with the event

message (string) : the message associated with the event

Creates a new execution feedback event. If the value of messageId is not recognised, getMessageId() returns MESSAGE_ID_OTHER.

e.getExecutionHandle() : ExecutionHandle

Returns the ExecutionHandle that raised the event. This is the source of the event cast to type ExecutionHandle.

e.getMessage() : string

Returns the message associated with this event.

For an event of type TYPE_DIAGNOSTIC the result is the diagnostic message localized for the session.

For an event of type TYPE_RECORD_COUNT the result consists of two decimal integers separated by a space, representing, respectively, the number of records read and written. For example:

"15000 0"

(15000 records read, none written).

For an event of type TYPE_PROGRESS the result consists of a single floating point number in the range

0.0 <= n <= 100.0

representing (an estimate of) the percentage of work completed. For example:

"26.3"

(26.3% completed).

e.getMessageId() : int

Returns the message ID associated with this event. This is one of the MESSAGE_ID_ constants.

e.getSeverity() : int

Returns the severity level of this event. The result is one of the SEVERITY_ constants declared above. The severity levels SEVERITY_WARNING and SEVERITY_ERROR are associated with the event type TYPE_DIAGNOSTIC and indicate a warning or error condition on the server. An error event ultimately causes execution to fail.

e.getType() : int

Returns the type of feedback represented by this event. The result is one of the TYPE_ constants declared above.

## ExecutionFeedbackListener Objects

Subclass of EventListener.

Listener for ExecutionFeedbackEvent.

```
e.executionFeedback(event)
event (ExecutionFeedbackEvent) :
```

Called when execution feedback is produced.

## ExecutionHandle Objects

Monitors and controls execution of a Task by a Session. A new handle is created for each execution.

```
e.getErrorMessage() : string
```

Returns an error message if execution terminated with an error and a message is available; returns None otherwise.

```
e.getExecutionState() : ExecutionState
```

Returns the latest ExecutionState of the associated task.

```
e.getExitCode() : int
```

Returns the exit code from executing the task. Typically this is either 0 to indicate success or 1 to indicate failure although some tasks may use different conventions. The result of querying the exit code before the task has finished executing is undefined.

```
e.getResult() : Object
```

Returns the result of the task if execution terminated with success and the task produced a result. Returns None if the task is still executing, or if it terminated in a state other than SUCCESS, or if it did not return a result.

```
e.getTask() : Task
```

Returns the Task being executed.

```
e.terminate() : ExecutionState
```

Terminates execution of the associated task and returns its final ExecutionState. The call blocks until the task has finished. The return value may not be TERMINATED if execution had completed before the termination request was sent.

```
e.terminate(milliseconds) : ExecutionState
```

`milliseconds (long) :` the maximum time to wait for the task to complete

Terminates execution of the associated task, waits until the task has finished or until the specified timeout has expired (whichever is sooner) and returns the execution state of the task. The result may not be TERMINATED if execution had completed before the termination request was sent or if execution had not completed before the timeout expired. A timeout of 0 or less means to wait forever.

```
e.waitForCompletion() : ExecutionState
```

Waits until the associated task has finished executing and returns its final ExecutionState.

```
e.waitForCompletion(milliseconds) : ExecutionState
```

`milliseconds (long) :` the maximum time to wait for completion

Waits until the associated task has finished executing, or until the specified timeout has expired (whichever is sooner) and returns the ExecutionState of the task. A timeout of 0 or less means to wait forever.

## ExecutionState Objects

Subclass of Enum.

This class enumerates the task execution states.

Constants:

- ERROR (ExecutionState) : The task has completed with an error.
- EXECUTING (ExecutionState) : The task is still executing.
- SUBMITTED (ExecutionState) : The task has been submitted for execution but not yet started.
- SUCCESS (ExecutionState) : The task has completed successfully.
- TERMINATED (ExecutionState) : The task has completed prematurely as the result of a terminate request.
- TERMINATING (ExecutionState) : A terminate request has been issued but the task has not yet completed.

```
getEnum(name) : ExecutionState
```

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : ExecutionState[]
```

Returns an array containing all the valid values for this enumeration class.

```
e.isCompletedState() : boolean
```

Returns True if this state represents a task that is no longer executing. This true for SUCCESS, TERMINATED and ERROR states.

## ExecutionStateEvent Objects

Indicates a state change in a task. An event is fired when a task moves into a new state, and the new state is obtained from the event by calling getExecutionState.

```
createExecutionStateEvent(handle, state) : ExecutionStateEvent
```

handle (ExecutionHandle) : the ExecutionHandle for the event

state (ExecutionState) : the new execution state.

Creates a new execution state event.

```
e.getExecutionHandle() : ExecutionHandle
```

Returns the ExecutionHandle that raised the event.

```
e.getExecutionState() : ExecutionState
```

Returns the new execution state which caused this event to be raised. The return value is one of the constants defined by ExecutionState.

**Note:** This may not correspond with the state that the task is now in since a subsequent state change may have occurred.

## ExecutionStateListener Objects

Subclass of EventListener.

Listener for ExecutionStateEvent.

```
e.executionStateChanged(event)
event (ExecutionStateEvent) :
```

Called when the execution state changes.

## Task Objects

A Task represents an operation that can be performed by the Modeler API.

```
t.getResult() : Object
```

Returns any object produced as a result of executing the task. This may be None if the task has not been executed yet, or there was an error during execution or if the task does not produce a result.

## TaskFactory Objects

The TaskFactory is used to create instances of Task. A task can be executed synchronously or asynchronously by the Session. Session-owned objects that are passed to TaskFactory methods must be executed by the session that owns them.

```
t.createDeleteFileTask(filePath) : Task
```

filePath (string) :- The path of file on server.

Create a task that delete the file using the specified file path.

```
t.createDownloadFileTask(path, outputStream) : Task
```

path (string) : the path of the remote file on server

outputStream (OutputStream) : the output stream

Creates a task that copies the content of a file on server to the specified output stream. The task will fail if the file cannot be read.

```
t.createDropTableTask(conn, tableName) : Task
```

conn (ServerDatabaseConnection) : - The ServerDatabaseConnection used for connect to the database.

tableName (string) : - The name of the table which you want to delete.

Create a task that drop the table using the specified table name.

```
t.createExportDocumentTask(document, outputStream, fileFormat) : Task
```

document (DocumentOutput) : the DocumentOutput object

outputStream (OutputStream) : the output stream

fileFormat (FileFormat) : the FileFormat to be used

Creates a task that exports a DocumentOutput object to an output stream using the specified FileFormat name. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the output is not owned by the same session that owns the task factory

ExportFormatException : if the document does not support the export format

```
t.createExportModelTask(model, outputStream, fileFormat) : Task
```

model (ModelOutput) : the ModelOutput object

outputStream (OutputStream) : the output stream

fileFormat (FileFormat) : the FileFormat to be used

Creates a task that exports a ModelOutput object to an output stream using the specified FileFormat name. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the output is not owned by the same session that owns the task factory

ExportFormatException : if the model does not support the export format

```
t.createExportStreamTask(stream, outputStream, fileFormat) : Task
```

stream (Stream) : the stream to be exported

outputStream (OutputStream) : the output stream

fileFormat (FileFormat) : the export file format

Creates a task that export a stream description to an output stream using specified file format.

Exceptions:

OwnerException : if the node is not owned by the same session that owns the task factory

ExportFormatException : if the stream does not support the export format

```
t.createImportPMMLModelTask(inputStream) : Task
```

inputStream (InputStream) : the input stream

Creates a task that imports a ModelOutput object from an input stream. Calling getResult() on the completed task returns an instance of ModelOutput.

```
t.createOpenDocumentTask(inputStream) : Task
```

inputStream (InputStream) : the input stream

Creates a task that reads a DocumentOutput object from an input stream. Calling getResult() on the completed task returns an instance of DocumentOutput. The document is not added to the output manager.

```
t.createOpenDocumentTask(inputStream, autoManage) : Task
```

inputStream (InputStream) : the input stream

autoManage (boolean) : whether the document should be added to the output manager

Creates a task that reads a DocumentOutput object from an input stream. Calling getResult() on the completed task returns an instance of DocumentOutput. Code that needs to open documents privately without having them made visible to the user should set the autoManage flag to False.

t.createOpenModelTask(inputStream) : Task

inputStream (InputStream) : the input stream

Creates a task that reads a ModelOutput object from an input stream. Calling getResult() on the completed task returns an instance of ModelOutput. The model is not added to the model manager.
t.createOpenModelTask(inputStream, autoManage) : Task

inputStream (InputStream) : the input stream

autoManage (boolean) : whether the model should be added to the model manager

Creates a task that reads a ModelOutput object from an input stream. Calling getResult() on the completed task returns an instance of ModelOutput. Code that needs to open models privately without having them made visible to the user should set the autoManage flag to False.
t.createOpenProcessorTask(inputStream, stream) : Task

inputStream (InputStream) : the input stream

stream (Stream) : the Stream

Creates a task that reads a Node object from an input stream and inserts it into the supplied Stream. Calling getResult() on the completed task returns an instance of Node.

Exceptions:

OwnerException : if the stream is not owned by the same session that owns the task factory
t.createOpenStreamTask(inputStream) : Task

inputStream (InputStream) : the input stream

Creates a task that reads a Stream object from an input stream. Calling getResult() on the completed task returns an instance of Stream.
t.createOpenStreamTask(inputStream, autoManage) : Task

inputStream (InputStream) : the input stream

autoManage (boolean) : whether the opened stream should be added to the stream manager

Creates a task that reads a Stream object from an input stream. Calling getResult() on the completed task returns an instance of Stream. Code that needs to open streams privately without having them made visible to the user should set the autoManage flag to False.
t.createSaveDocumentTask(document, outputStream) : Task

document (DocumentOutput) : the DocumentOutput object

outputStream (OutputStream) : the output stream

Creates a task that saves a DocumentOutput object to an output stream. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the document output is not owned by the same session that owns the task factory
t.createSaveModelTask(model, outputStream) : Task

model (ModelOutput) : the ModelOutput object

outputStream (OutputStream) : the output stream

Creates a task that saves a ModelOutput object to an output stream. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the model output is not owned by the same session that owns the task factory

```
t.createSaveProcessorTask(node, outputStream) : Task
```

node (Node) : the Node object

outputStream (OutputStream) : the output stream

Creates a task that saves a Node object to an output stream. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the node is not owned by the same session that owns the task factory

```
t.createSaveStreamTask(stream, outputStream) : Task
```

stream (Stream) : the Stream object

outputStream (OutputStream) : the output stream

Creates a task that saves a Stream object to an output stream. Calling getResult() on the completed task returns None.

Exceptions:

OwnerException : if the stream is not owned by the same session that owns the task factory

```
t.createUpdatePMMLModelTask(modelOutput, inputStream) : Task
modelOutput (ModelOutput) :
inputStream (InputStream) :
```

Creates a task that updates the ModelOutput object with PMML read from an input stream. The task will fail if the ModelOutput is not based on a PMML model or the ModelOutput and the PMML algorithms are not the same.

Exceptions:

OwnerException : if the model output is not owned by the same session that owns the task factory

```
t.createUploadFileTask(path, inputStream) : Task
```

path (string) : the path of the remote file on server

inputStream (InputStream) : the input stream

Creates a task that copies the content of the specified input stream to a file on server, replacing any existing file content. The task will fail if the file cannot be written.

## TaskRunner Objects

The TaskRunner provides a convenient way of creating and running tasks synchronously.
t.createStream(name, autoConnect, autoManage) : Stream

name (string) : the object's name

autoConnect (boolean) : whether the stream should be auto-connected to the server

autoManage (boolean) : whether the stream should be added to the stream manager

Creates and returns a new Stream. Note that code that needs to create streams privately without having them made visible to the user should set the autoManage flag to False.

Exceptions:

ServerConnectionException : if the Session is already connected and the auto-connect flag is True but a new connection could not be created for the stream
t.exportDocumentToFile(documentOutput, filename, fileFormat)

documentOutput (DocumentOutput) : the document to be exported

filename (string) : the exported file path

fileFormat (FileFormat) : the export file format

Exports the stream description to a file using the specified file format.

Exceptions:

OwnerException : if the document is not owned by the task runner session

SessionException : if the document cannot be exported for some reason

ExportFormatException : if the document does not support the export format
t.exportModelToFile(modelOutput, filename, fileFormat)

modelOutput (ModelOutput) : the model to be exported

filename (string) : the exported file path

fileFormat (FileFormat) : the export file format

Exports the model to a file using the specified file format.

Exceptions:

OwnerException : if the model is not owned by the task runner session

SessionException : if the model cannot be exported for some reason

ExportFormatException : if the model does not support the export format
t.exportStreamToFile(stream, filename, fileFormat)

stream (Stream) : the stream to be exported

filename (string) : the exported file path

fileFormat (FileFormat) : the export file format

Exports the stream description to a file using the specified file format.

Exceptions:

OwnerException : if the stream is not owned by the task runner session

SessionException : if the stream cannot be exported for some reason

ExportFormatException : if the stream does not support the export format

```
t.insertNodeFromFile(filename, diagram) : Node
```

filename (string) : the file path

diagram (Diagram) : the diagram that the node should be inserted into

Reads and returns a node from the specified file, inserting it into the supplied diagram. Note that this can be used to read both Node and SuperNode objects.

Exceptions:

OwnerException : if the diagram is not owned by the task runner session

SessionException : if the node cannot be loaded for some reason

ObjectLockedException : if the stream that the node is being added to is locked

```
t.openDocumentFromFile(filename, autoManage) : DocumentOutput
```

filename (string) : the document file path

autoManage (boolean) : whether the document should be added to the output manager

Reads and returns a document from the specified file.

Exceptions:

SessionException : if the document cannot be loaded for some reason

```
t.openModelFromFile(filename, autoManage) : ModelOutput
```

filename (string) : the model file path

autoManage (boolean) : whether the model should be added to the model manager

Reads and returns a model from the specified file.

Exceptions:

SessionException : if the model cannot be loaded for some reason

```
t.openStreamFromFile(filename, autoManage) : Stream
```

filename (string) : the stream file path

`autoManage (boolean) :` whether the stream should be added to the stream manager

Reads and returns a stream from the specified file.

Exceptions:

`SessionException :` if the stream cannot be loaded for some reason

```
t.saveDocumentToFile(documentOutput, filename)
```

`documentOutput (DocumentOutput) :` the document to be saved

`filename (string) :` the document file path

Saves the document to the specified file location.

Exceptions:

`OwnerException :` if the document is not owned by the task runner session

`SessionException :` if the document cannot be saved for some reason

```
t.saveModelToFile(modelOutput, filename)
```

`modelOutput (ModelOutput) :` the model to be saved

`filename (string) :` the model file path

Saves the model to the specified file location.

Exceptions:

`OwnerException :` if the model is not owned by the task runner session

`SessionException :` if the model cannot be saved for some reason

```
t.saveStreamToFile(stream, filename)
```

`stream (Stream) :` the stream to be saved

`filename (string) :` the stream file path

Saves the stream to the specified file location.

Exceptions:

`OwnerException :` if the stream is not owned by the task runner session

`SessionException :` if the stream cannot be saved for some reason

# Chapter 9. Streams and SuperNodes

This provides objects that perform data processing and model building.

## BuiltObject Objects

Subclass of PropertiedObject.

This encapsulates the concrete results of executing ObjectBuilder nodes.

```
b.close()
```

Closes this built object and releases its external resources. Closing an object which is already closed has no effect, otherwise the behaviour of a method applied to a closed object is undefined except where stated.

```
b.getBuilderProcessorID() : string
```

Returns the ID of the object builder node that built this object. The empty string is returned if this information is not available.

```
b.getBuilderStreamID() : string
```

Returns the temporary session ID of the stream containing the object builder node that built this object. If the stream is closed and subsequently re-opened, the re-opened stream's ID will be different from the value returned here.

The empty string is returned if this information is not available.

```
b.getID() : string
```

Returns the temporary session ID of this object. A new ID is allocated each time a new built object is created or opened and the ID is not persisted. This means that if the same object is re-opened multiple times, each object will include a different ID.

```
b.isClosed() : boolean
```

Returns True if this object has been closed.

```
b.isExportable(fileFormat) : boolean
```

fileFormat (FileFormat) : the FileFormat

Returns True if this object can be exported using the supplied FileFormat or False otherwise.

## CFNode Objects

Subclass of Node.

This identifies Node objects that are implemented via the Component Framework.

```
c.getFeature() : Object
```

Return the underlying CF feature of the CF node.

## CompositeModelApplier Objects

Subclass of ModelApplier, CompositeModelOwner.

This encapsulates the auto-model applier nodes ModelApplier that can ensemble the scores from multiple models into a single score.

## CompositeModelBuilder Objects

Subclass of ModelBuilder.

This encapsulates the auto-model builder nodes ModelBuilder that can build and evaluate multiple models using different modeling algorithms and settings.

```
c.getAllModelAlgorithms() : List
```

Returns the model builder node ids that are available in this CompositeModelBuilder.

```
c.isAlgorithmEnabled(modelBuilderId) : boolean
```

modelBuilderId (string) : build model id

Returns whether the supplied algorithm is enabled. If the supplied algorithm ID is not one of the algorithms available for this model builder, the method returns False. An algorithm may be enabled in the model builder, but invalid due to other settings (see isAlgorithmInvalid(String)).

```
c.isAlgorithmInvalid(algorithmID) : boolean
algorithmID (string) :
```

Returns whether the supplied algorithm is invalid. An algorithm may be enabled (see isAlgorithmEnabled(String)) in this model builder, but be invalid due to other settings. For instance, the target measure may not be valid for the algorithm. If the algorithm is invalidated in this way it will not be built, even if it is enabled.

```
c.setAlgorithmEnabled(modelBuilderId, value)
```

modelBuilderId (string) : build model id

value (boolean) : the enabled value

Sets whether the supplied algorithm is enabled. If the supplied algorithm ID is not one of the algorithms available for this model builder, the method call will have no effect.

## CompositeModelOutput Objects

Subclass of ModelOutput, CompositeModelOwner.

This encapsulates the concrete results of executing ModelBuilder nodes or by opening or importing Composite models. This will be the results of the auto models.

```
c.getModelDetail() : CompositeModelDetail
```

Returns the underlying composite model representation.

## CompositeModelOwner Objects

This encapsulates objects that own auto-built models. These models can be ensembled and the scores from multiple models combined into a single score.

```
c.getCompositeModelDetail() : CompositeModelDetail
```

Returns the underlying composite model representation.

```
c.getModelResult(resultId) : CompositeModelResult
resultId (string) :
```

Returns the model results for the specified ID or None if no such model result exists.

```
c.getModelResultIDs() : List
```

Returns the list of model result IDs. The result ID is the same as the name of the individual composite model result.

```
c.isModelResultUsed(resultId) : boolean
resultId (string) :
```

Returns whether the specified model result ID is enabled/active.

```
c.removeModelResults(resultIds)
```

`resultIds (List) :` the list of result IDs to be removed

Permanently removes the supplied model results from this composite model object owner.

```
c.setModelResultUsed(resultId, value)
```

`resultId (string) :` the result ID

`value (boolean) :` `True` if the model should be enabled, `False` if not

Sets whether the specified model result ID is active. Note that supplying model results that have previously been removed will have no effect i.e. they will not be re-added into the composite model object owner.

## `CompositeModelResults` Objects

This encapsulates the concrete results for the individual models forming the Composite Model.

```
c.getModelMeasureLabel(measure) : string
```

`measure (string) :` the evaluation measure name

Returns a descriptive label for the specified evaluation measure or `None` if the measure is not valid for this result.

```
c.getModelMeasurePartitionCount() : int
```

Returns the number of partitions for which evaluation measures exist. Partitions (where they exist) are numbered: 0=Training, 1=Testing, 2=Validation.

```
c.getModelMeasureType(measure) : StorageType
```

`measure (string) :` the evaluation measure name

Returns the data type of the specified evaluation measure or `None` if the measure is not valid for this result.

```
c.getModelMeasureValue(measure, partition) : Object
```

`measure (string) :` the evaluation measure name

`partition (int) :` the partition index

Returns the value of the specified evaluation measure in the specified partition, or `None` if either the measure or the partition is not valid for this result.

```
c.getModelMeasures() : List
```

Returns the list of measures used to evaluate this model result.

c.getModelName() : string

Returns the name of this model result. The model name is expected to be unique within a set of composite model result objects.

c.getModelOutput() : ModelOutput

Returns the model output of the composite model result.

c.isUsed() : boolean

Returns True if this composite model result is used.

## SuperNode Objects

Subclass of Node, ParameterProvider.

This encapsulates the functionality of any SuperNode whose behaviour is determined by its constituent nodes.

s.getChildDiagram() : SuperNodeDiagram

Returns the diagram containing the nodes encapsulated by this SuperNode.

s.getCompositeProcessorType() : SuperNodeType

Returns the type of this SuperNode.

s.isPasswordNeeded() : boolean

If the SuperNode is not locked or a password has been entered then return False, otherwise return True.

s.setCompositeProcessorType(type)

type (SuperNodeType) : the SuperNode type

Sets the type of this SuperNode.

Exceptions:

ObjectLockedException : if the diagram is currently locked

s.verifyPassword(password) : boolean

password (string) : the password to be tested

Check whether the supplied password is valid or not.

## SuperNodeDiagram Objects

Subclass of Diagram.

This is the diagram used by SuperNode objects to store nodes. It allows connections between the two pseudo nodes, (the input connector and output connector) and other nodes in the diagram to be edited.

s.disconnectInputConnector()

Removes any direct links between the diagram's input connector and other nodes in the diagram. If the SuperNode is an initial node and therefore doesn't have an input connector, this method does nothing.

Exceptions:

ObjectLockedException : if the diagram is currently locked

```
s.disconnectOutputConnector()
```

Removes any direct links between the diagram's output connector and other nodes in the diagram. If the SuperNode is an terminal node and therefore doesn't have an input connector, this method does nothing.

Exceptions:

`ObjectLockedException` : if the diagram is currently locked

```
s.getInputConnector() : Node
```

Returns the input connector from this diagram, or `None` if the diagram belongs to an initial node and has no input connector.

```
s.getOutputConnector() : Node
```

Returns the output connector from this diagram, or `None` if the diagram belongs to a terminal node and has no output connector.

```
s.linkFromInputConnector(node)
node (Node) :
```

Creates a link from the input connector to the supplied node.

Exceptions:

`OwnerException` : if any objects in the path are not owned by this diagram

`ObjectLockedException` : if the diagram is currently locked

`InvalidEditException` : if the connection would be invalid

```
s.linkToOutputConnector(node)
node (Node) :
```

Creates a link from the supplied node to the output connector.

Exceptions:

`OwnerException` : if any objects in the path are not owned by this diagram

`ObjectLockedException` : if the diagram is currently locked

`InvalidEditException` : if the connection would be invalid

```
s.runAll(results) : ExecutionHandle
```

`results (Collection)` : an empty collection that will contain any built objects once execution has completed

Executes the executable nodes within this sub-stream synchronously and waits for it to complete. Returns an `ExecutionHandle` which can be used to access the exit status and any result from the task.

Note that this should only be called on a terminal `SuperNode`.

Exceptions:

`OwnerException` : if the stream was not created by this session

`ObjectLockedException` : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running another task, cannot execute the task or if execution completes in a state other than SUCCESS

```
s.runSelected(nodes, results) : ExecutionHandle
```

nodes (Node[]) : the array of Node objects to be executed

results (Collection) : an empty collection that will contain any built objects once execution has completed

Executes the supplied array of nodes synchronously and waits for them to complete. There must be at least one node in the array. Returns an ExecutionHandle which can be used to access the exit status and any result from the task.

Note that this should only be called on a terminal SuperNode.

Exceptions:

OwnerException : if the nodes are not all owned by this stream

ObjectLockedException : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running a stream, another task, cannot execute the task or if execution completes in a state other than SUCCESS

IllegalArgumentException : if the array is empty

```
s.unlinkFromInputConnector(node)
node (Node) :
```

Removes any direct link from the input connector to the supplied node.

Exceptions:

OwnerException : if any objects in the path are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

InvalidEditException : if the connection would be invalid

```
s.unlinkToOutputConnector(node)
node (Node) :
```

Removes any direct link from the supplied node to the output connector.

Exceptions:

OwnerException : if any objects in the path are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

InvalidEditException : if the connection would be invalid

## SuperNodeType Objects

Subclass of Enum.

This class enumerates the types of SuperNodes.

Constants:

- INITIAL (SuperNodeType) : Represents a SuperNode that begins a sequence of nodes.
- PROCESS (SuperNodeType) : Represents a SuperNode that allows data to flow through it.
- TERMINAL (SuperNodeType) : Represents a SuperNode that ends a sequence of nodes.

```
getEnum(name) : SuperNodeType
```

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : SuperNodeType[]
```

Returns an array containing all the valid values for this enumeration class.

## DataReader Objects

Subclass of InitialNode.

This encapsulates the functionality of a data node that reads records e.g. from a file or database table.

## DataTransformer Objects

Subclass of ProcessNode.

This encapsulates the functionality of a standard inline data node.

## DataWriter Objects

Subclass of TerminalNode.

This encapsulates the functionality of a node that writes records e.g. to a file or database table.

```
d.getExportDataModel() : DataModel
```

Returns the DataModel being exported by this node.

## DiagramConnector Objects

Subclass of ProcessNode.

A node which provides a connection point between a SuperNodeDiagram and its containing diagram. A connector routes data between diagrams.

## DocumentBuilder Objects

Subclass of ObjectBuilder.

This encapsulates the functionality of a node that builds a viewable output object such as a table, graph or report.

## DocumentOutput Objects

Subclass of BuiltObject.

This encapsulates the concrete results of executing DocumentBuilder nodes that produce generic document-type outputs.

```
d.getDocumentOutputType() : DocumentOutputType
```

Returns the DocumentOutputType of this output. The returned value corresponds with one of the output-type constants defined by DocumentOutputType.

```
d.getTypeName() : string
```

Returns the name of this type of output object.

## DocumentOutputType Objects

Subclass of Enum.

This class enumerates the types of document outputs that are generated by the different NodeType that are DocumentBuilders or which can be imported.

Constants:

- ANALYSIS (DocumentOutputType) :
- COLLECTION (DocumentOutputType) :
- CUSTOM_TABLE (DocumentOutputType) :
- DATA_AUDIT (DocumentOutputType) :
- DISTRIBUTION (DocumentOutputType) :
- ENSEMBLE (DocumentOutputType) :
- EVALUATION (DocumentOutputType) :
- GRAPH_BOARD (DocumentOutputType) :
- HISTOGRAM (DocumentOutputType) :
- MATRIX (DocumentOutputType) :
- MEANS (DocumentOutputType) :
- MULTI_PLOT (DocumentOutputType) :
- QUALITY (DocumentOutputType) :
- REPORT (DocumentOutputType) :
- REPORT_DOCUMENT (DocumentOutputType) :
- SCATTER_PLOT (DocumentOutputType) :
- SPSS_PROCEDURE (DocumentOutputType) :
- STATISTICS (DocumentOutputType) :
- TABLE (DocumentOutputType) :
- TIME_PLOT (DocumentOutputType) :
- TRANSFORM (DocumentOutputType) :
- UNKNOWN (DocumentOutputType) : Represents a document output object whose type cannot be determined.
- WEB (DocumentOutputType) :

```
addDocumentOutputType(identifier) : DocumentOutputType
identifier (string) :
```

Adds a the type with the supplied name. Returns the new type or None if a type with the supplied name already exists. This method is for system use only and is only public as an implementation detail.

```
getEnum(name) : DocumentOutputType
```

`name (string) :` the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

```
getValues() : DocumentOutputType[]
```

Returns an array containing all the valid values for this enumeration class.

```
d.isUnknown() : boolean
```

Returns True if this value is UNKNOWN.

```
removeDocumentOutputType(identifier)
```

`identifier (string) :` the type to be removed

Removes the specified type. The type should have been created with addDocumentOutputType(). This method is for system use only and is only public as an implementation detail.

## ExportFormatException Objects

Subclass of ModelerException.

Indicates that an attempt was made to export a built object using an unsupported file format.

No message string is set for this exception.

```
e.getFileFormat() : FileFormat
```

Returns the unsupported export format.

```
e.getObject() : BuiltObject
```

Returns the built object.

## GraphBuilder Objects

Subclass of DocumentBuilder.

This encapsulates the functionality of a node that builds a graph output.

```
g.getGraphOutput() : GraphOutput
```

Returns the most recent GraphOutput produced by this node, or None if this GraphBuilder has either not be executed before or failed to produce an object during the most recent execution.

## GraphOutput Objects

Subclass of DocumentOutput.

This encapsulates the concrete results of executing DocumentBuilder nodes that produce objects that are based on graphs.

## InitialNode Objects

Subclass of Node.

This encapsulates the functionality of any node that begins a sequence of nodes. These typically import data from a data source such as a database or file.

## ProcessNode Objects

Subclass of Node.

This encapsulates the functionality of any node that allows data to flow through it i.e., a Node that is neither a InitialNode nor a TerminalNode.

## InvalidEditException Objects

Subclass of ModelerException.

An exception thrown when an invalid attempt is made to edit the links between two or more nodes.

No message string is set for this exception.

```
i.getProcessorDiagram() : Diagram
```

Returns the Diagram that generated the exception.

```
i.getProcessorStream() : Stream
```

Returns the Stream that generated the exception.

## ModelApplier Objects

Subclass of ProcessNode.

This encapsulates the functionality of an inline node that applies models.

```
m.getBuilderProcessorID() : string
```

Returns the ID of the model builder node that built the underlying model. The empty string is returned if this information is not available.

```
m.getBuilderStreamID() : string
```

Returns the ID of the stream of the model builder node that built the underlying model. The empty string is returned if this information is not available.

```
m.getLocalizedAlgorithmName() : string
```

Returns the localized algorithm name.

```
m.getModelDetail() : ModelDetail
```

Returns the underlying model representation.

## ModelBuilder Objects

Subclass of ObjectBuilder.

This encapsulates the functionality of a node that builds a model output object.

```
m.getModelOutputType() : ModelOutputType
```

Returns the type of ModelOutput that is produced by this type of model builder. The returned value corresponds to one of the model-type constants defined by ModelOutputType.

## ModelOutput Objects

Subclass of BuiltObject.

This encapsulates the concrete results of executing ModelBuilder nodes or by opening or importing models.

```
m.getLocalizedAlgorithmName() : string
```

Returns the localized algorithm name.

```
m.getModelApplierID() : string
```

Returns the ID of ModelApplier that corresponds with this model output or None if this model output cannot be applied directly to data.

```
m.getModelDetail() : ModelDetail
```

Returns the underlying model representation.

```
m.getModelOutputType() : ModelOutputType
```

Returns the ModelOutputType of this model output. The returned value corresponds with one of the model-type constants defined by ModelOutputType.

```
m.getTypeName() : string
```

Returns the name of this type of output object.

## ModelOutputType Objects

Subclass of Enum.

This class enumerates the types of model outputs that are generated by the different NodeType that are ModelBuilders or which can be imported.

Constants:

- ANOMALY_DETECTION (ModelOutputType) :
- APRIORI (ModelOutputType) :
- ASSOCIATION (ModelOutputType) :
- BACK_PROPAGATION (ModelOutputType) :
- C50 (ModelOutputType) :
- CARMA (ModelOutputType) :
- CART (ModelOutputType) :
- CATEGORIZE (ModelOutputType) :
- CHAID (ModelOutputType) :
- DB2IM_ASSOCIATION (ModelOutputType) :
- DB2IM_CLUSTER (ModelOutputType) :
- DB2IM_REGRESSION (ModelOutputType) :
- DB2IM_SEQUENCE (ModelOutputType) :
- DB2IM_TREE (ModelOutputType) :
- DECISION_LIST (ModelOutputType) :
- FACTOR (ModelOutputType) :
- FEATURE_SELECTION (ModelOutputType) :
- KMEANS (ModelOutputType) :

- KOHONEN (ModelOutputType) :
- LINEAR_REGRESSION (ModelOutputType) :
- LOGISTIC_REGRESSION (ModelOutputType) :
- MS_ASSOCIATION (ModelOutputType) :
- MS_CLUSTERING (ModelOutputType) :
- MS_LOGISTIC (ModelOutputType) :
- MS_NAIVE_BAYES (ModelOutputType) :
- MS_NEURAL_NETWORK (ModelOutputType) :
- MS_REGRESSION (ModelOutputType) :
- MS_SEQUENCE_CLUSTERING (ModelOutputType) :
- MS_TIME_SERIES (ModelOutputType) :
- MS_TREE (ModelOutputType) :
- NUMERIC_PREDICTOR (ModelOutputType) :
- ORACLE_ADAPTIVE_BAYES (ModelOutputType) :
- ORACLE_AI (ModelOutputType) :
- ORACLE_DECISION_TREE (ModelOutputType) :
- ORACLE_GLM (ModelOutputType) :
- ORACLE_KMEANS (ModelOutputType) :
- ORACLE_NAIVE_BAYES (ModelOutputType) :
- ORACLE_NMF (ModelOutputType) :
- ORACLE_OCLUSTER (ModelOutputType) :
- ORACLE_SVM (ModelOutputType) :
- QUEST (ModelOutputType) :
- RULE (ModelOutputType) :
- SEQUENCE (ModelOutputType) :
- SPSS_MODEL (ModelOutputType) :
- TEXT_EXTRACTION (ModelOutputType) :
- TIME_SERIES (ModelOutputType) :
- TWOSTEP (ModelOutputType) :
- UNKNOWN (ModelOutputType) : Represents a model output object whose type cannot be determined.

addModelOutputType(identifier) : ModelOutputType
identifier (string) :

Adds a the type with the supplied name. Returns the new type or None if a type with the supplied name already exists. This method is for system use only and is only public as an implementation detail.

getEnum(name) : ModelOutputType

name (string) : the enumeration name

Returns the enumeration with the supplied name or None if no enumeration exists for the supplied name.

getValues() : ModelOutputType[]

Returns an array containing all the valid values for this enumeration class.

m.isUnknown() : boolean

Returns True if this value is UNKNOWN.

removeModelOutputType(identifier)

identifier (string) : the type to be removed

Removes the specified type. The type should have been created with addModelOutputType(). This method is for system use only and is only public as an implementation detail.

## ObjectBuilder Objects

Subclass of TerminalNode.

This encapsulates the functionality of a node that builds an output object such as a model or graph.

o.getBuiltObject() : BuiltObject

Returns the most recent BuiltObject produced by this node, or None if this ObjectBuilder has either not be executed before or failed to produce an object during the most recent execution.

## Node Objects

Subclass of PropertiedObject.

This is the base interface for all nodes.

n.flushCache()

Flushes the cache of this object. Has no effect if the cache is not enabled or is not full.

n.getID() : string

Returns the ID of this object. A new ID is created each time a new node is created. The ID is persisted with the node when it is saved as part of a stream so that when the stream is opened, the node IDs are preserved. However, if a saved node is inserted into a stream, the insert node is considered to be a new object and will be allocated a new ID.

n.getInputDataModel() : DataModel

Returns the DataModel coming into this node.

n.getOutputDataModel() : DataModel

Returns the DataModel output by this node.

n.getProcessorDiagram() : Diagram

Returns the Diagram that owns this node.

n.getProcessorStream() : Stream

Returns the Stream that owns this node.

n.getTypeName() : string

Returns the name of this type of node.

n.getXPosition() : int

Returns the x position offset of the node in the Diagram.

n.getYPosition() : int

Returns the y position offset of the node in the Diagram.

n.isCacheEnabled() : boolean

Returns True if the cache is enabled, False otherwise.

n.isCacheFull() : boolean

Returns True if the cache is full, False otherwise.

```
n.isInitial() : boolean
```

Returns True if this is an initial node i.e. one that occurs at the start of a stream.

```
n.isInline() : boolean
```

Returns True if this is an in-line node i.e. one that occurs mid-stream.

```
n.isTerminal() : boolean
```

Returns True if this is a terminal node i.e. one that occurs at the end of a stream.

```
n.run(results) : ExecutionHandle
results (Collection) :
```

Executes this node synchronously and waits for execution to complete. Returns an ExecutionHandle which can be used to access the exit status and any result from the task. Equivalent to calling:

```
node.getProcessorStream().runSelected(new Processor[] {node}, results);
```

Exceptions:

OwnerException : if there is inconsistent ownership

ObjectLockedException : if the owner stream is locked for some reason (for example, it is already executing)

ServerConnectionException : if the connection to the server cannot be established

SessionException : if some other exception occurs

```
n.setCacheEnabled(val)
val (boolean) :
```

Enables or disables the cache for this object. If the cache is full and the caching becomes disabled, the cache is flushed.

```
n.setPositionBetween(source, target)
```

source (Node) : the predecessor

target (Node) : the successor

Sets the position of the node in the Diagram so it is positioned between the supplied nodes.

```
n.setXYPosition(x, y)
```

x (int) : the x offset

y (int) : the y offset

Sets the position of the node in the Diagram.

## Diagram Objects

This is the container used to assemble Node objects into a connected flow or well-formed sequence of nodes.

```
d.clear()
```

Deletes all nodes from this diagram.

Exceptions:

`ObjectLockedException` : if the diagram is currently locked

```
d.create(nodeType, name) : Node
```

`nodeType (string)` : the node type name

`name (string)` : the object's name

Creates a `Node` of the specified type and adds it to this diagram.

Exceptions:

`ObjectCreationException` : if the node cannot be created for some reason

`ObjectLockedException` : if the node cannot be added to the stream

```
d.createAt(nodeType, name, x, y) : Node
```

`nodeType (string)` : the node type name

`name (string)` : the object's name

`x (int)` : the x location

`y (int)` : the y location

Creates a `Node` of the specified type and adds it to this diagram at the specified location. If either $x < 0$ or $y < 0$, the location is not set.

Exceptions:

`ObjectCreationException` : if the node cannot be created for some reason

`ObjectLockedException` : if the node cannot be added to the stream

```
d.createModelApplier(modelOutput, name) : Node
```

`modelOutput (ModelOutput)` : the model output object

`name (string)` : the new object's name

Creates a `ModelApplier` derived from the supplied model output object.

Exceptions:

`ObjectCreationException` : if the node cannot be created for some reason

`ObjectLockedException` : if the node cannot be added to the stream

```
d.delete(node)
```

`node (Node)` : the node to be removed

Deletes the specified node from this diagram. The node must be owned to this diagram.

Exceptions:

OwnerException : if the node is not owned by this diagram

ObjectLockedException : if the diagram is currently locked

```
d.deleteAll(nodes)
```

nodes (Collection) : the collection of nodes to be removed

Deletes all the specified nodes from this diagram. All nodes in the collection must belong to this diagram.

Exceptions:

OwnerException : if any node parameters are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

ClassCastException : if any item in the collection is not of type Node

```
d.disconnect(node)
```

node (Node) : the node to be disconnected

Removes any links between the supplied node and any other nodes in this diagram.

Exceptions:

OwnerException : if any objects in the path are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

```
d.findAll(type, label) : Collection
```

type (string) : the node type

label (string) : the node label

Returns a list of all nodes with the specified type and name. Either the type or name may be None in which case the other parameter is used.

```
d.findAll(filter, recursive) : Collection
```

filter (NodeFilter) : the node filter

recursive (boolean) : if True then composite nodes should be recursively searched

Returns a collection of all nodes accepted by the specified filter. If the recursive flag is True then any SuperNodes within this diagram are also searched.

```
d.findByID(id) : Node
```

id (string) : the node ID

Returns the node with the supplied ID or None if no such node exists. The search is limited to the current diagram.

```
d.findByType(type, label) : Node
```

type (string) : the node type

label (string) : the node label

Returns the node with the supplied type and/or label. Either the type or name may be None in which case the other parameter is used. If multiple nodes match then an arbitrary one is chosen and returned. If no nodes match then the return value is None.

```
d.findDownstream(fromNodes) : List
```

fromNodes (List) : the start point of the search

Searches from the supplied list of nodes and returns the set of nodes downstream of the supplied nodes. The returned list includes the originally supplied nodes.

```
d.findUpstream(fromNodes) : List
```

fromNodes (List) : the start point of the search

Searches from the supplied list of nodes and returns the set of nodes upstream of the supplied nodes. The returned list includes the originally supplied nodes.

```
d.flushCaches()
```

Flushes the caches of any cache-enabled Node objects in the diagram. Has no effect if caches are not enabled or are not full.

```
d.insert(source, nodes, newIDs) : List
```

source (Diagram) : the diagram that owns the nodes to be inserted

nodes (List) : the nodes to be copied

newIDs (boolean) : True if new IDs should be generated for each node or False if the existing IDs should be re-used

Inserts copies of the nodes in the supplied list. It is assumed that all nodes in the supplied list are contained within the specified diagram. The new IDs flag indicates whether new IDs should be generated for each node, or whether the existing ID should be copied across. It is assumed that all nodes in a diagram have a unique ID so this flag must be set True if the source diagram is the same as this diagram. The method returns the list of newly inserted nodes where the order of the nodes is undefined (i.e. the ordering is not necessarily the same as the order of nodes in the input list).

Exceptions:

ObjectLockedException : if the diagram has been locked because of another operation

InvalidEditException : if the edit would be invalid

```
d.isEnabled(node) : boolean
```

node (Node) : the node

Returns True if the supplied node is enabled.

Exceptions:

OwnerException : if the node is not owned by this diagram

```
d.isOwner(node) : boolean
```

node (Node) : the node

Returns True if the node is owned by this diagram.

```
d.isValidLink(source, target) : boolean
```

`source (Node)` : the source node

`target (Node)` : the target node

Returns `True` if it would be valid to create a link between the specified source and target nodes. This checks that both objects belong to this diagram, that the source can supply a link and the target can receive a link, and that creating such a link will not cause a circularity in the diagram.

Exceptions:

`OwnerException` : if the source or target node are not owned by this diagram

```
d.iterator() : Iterator
```

Returns an iterator over the `Node` objects contained in this diagram. The behaviour of the iterator if the diagram is modified between calls of `next()` is undefined.

**Note:** The iterator returns "top-level" nodes and does not recursively descend into any composite nodes.

```
d.link(source, target)
```

`source (Node)` : the source node

`target (Node)` : the target node

Creates a new link between the source and the target.

Exceptions:

`OwnerException` : if any node parameters are not owned by this diagram

`ObjectLockedException` : if the diagram is currently locked

`InvalidEditException` : if the connection would be invalid

```
d.link(source, targets)
```

`source (Node)` : the source node

`targets (List)` : the list of target nodes

Creates new links between the source and each target node in the supplied list.

Exceptions:

`OwnerException` : if any node parameters are not owned by this diagram

`ObjectLockedException` : if the diagram is currently locked

`InvalidEditException` : if a connection would be invalid

`ClassCastException` : if targets does not contain instances of `Node`

```
d.linkBetween(inserted, source, target)
```

`inserted (Node)` : the node to be inserted

`source (Node)` : the source node

target (Node) : the target node

Connects a Node between two other instances and sets the position of the inserted node to be between those. Any direct link between the source and target is removed first. If any link would be invalid (the source is a terminal node, the target is a source node or the target cannot accept any more links), a ModelerException is thrown and no changes are made to the diagram.

Exceptions:

OwnerException : if any of the nodes are not owned by the diagram

ObjectLockedException : if the diagram has been locked because of another operation

InvalidEditException : if the edit is invalid e.g., the target already has the maximum number of input connections

```
d.linkPath(path)
```

path (List) : the set of Node instances

Creates a new path between Node instances. The first node is linked to the second, the second is linked to the third etc. If any link would be invalid (for example, the nodes are already linked, the source is a terminal node, the target is a source node or the target cannot accept any more links), a ModelerException is thrown and no changes are made to the diagram.

Exceptions:

OwnerException : if any objects in the path are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

InvalidEditException : if a link between two adjacent nodes in the path cannot be created.

```
d.linkUpdater(updater, updatable)
```

updater (Node) : the updater node

updatable (Node) : the updatable node

Creates a new update link between the updater and the updatable. It is expected that updater implements the Updater interface while the updatable implements the Updatable interface.

Exceptions:

OwnerException : if any node parameters are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

InvalidEditException : if the connection would be invalid

```
d.predecessorAt(node, index) : Node
```

node (Node) : the node

index (int) : which predecessor to return

Returns the specified immediate predecessor of the supplied node or None if the index is out of bounds.

Exceptions:

`OwnerException` : if the node is not owned by this diagram

```
d.predecessorCount(node) : int
```

`node (Node)` : the node

Returns the number of immediate predecessors of the supplied node.

Exceptions:

`OwnerException` : if the node is not owned by this diagram

```
d.predecessors(node) : List
```

`node (Node)` : the node

Returns the immediate predecessors of the supplied node.

Exceptions:

`OwnerException` : if the node is not owned by this diagram

```
d.replace(originalNode, replacementNode, discardOriginal)
```

`originalNode (Node)` : the node to be replaced

`replacementNode (Node)` : the new node

`discardOriginal (boolean)` : if `True`, the id of the original node is assigned to the new node and the original node is automatically deleted from the diagram. If `False`, the original node is retained and the replacement node id is unchanged

Replaces the specified node from this diagram. The both nodes must be owned by this diagram.

Exceptions:

`OwnerException` : if any node is not owned by this diagram

`ObjectLockedException` : if the diagram is currently locked

`InvalidEditException` : is the link operation is invalid

```
d.setEnabled(node, enabled)
```

`node (Node)` : the node

`enabled (boolean)` : whether the node should be enabled

Sets the enabled state of the supplied node.

Exceptions:

`OwnerException` : if the node is not owned by this diagram

```
d.size() : int
```

Returns the number of `Node` objects contained in this diagram.

```
d.successorAt(node, index) : Node
```

`node (Node) :` the successor

`index (int) :` which successor to return

Returns the specified immediate successor of the supplied node or `None` if the index is out of bounds.

Exceptions:

`OwnerException :` if the node is not owned by this diagram

```
d.successorCount(node) : int
```

`node (Node) :` the node

Returns the number of immediate successors of the supplied node.

Exceptions:

`OwnerException :` if the node is not owned by this diagram

```
d.successors(node) : List
```

`node (Node) :` the node

Returns the immediate successors of the supplied node.

Exceptions:

`OwnerException :` if the node is not owned by this diagram

```
d.unlink(source, target)
```

`source (Node) :` the source node

`target (Node) :` the target node

Removes any direct link between the source and the target.

Exceptions:

`OwnerException :` if any node parameters are not owned by this diagram

`ObjectLockedException :` if the diagram is currently locked

```
d.unlink(source, targets)
```

`source (Node) :` the source node

`targets (List) :` the list of target nodes

Removes any direct links between the source and each object in the targets list.

Exceptions:

`OwnerException :` if any node parameters are not owned by this diagram

`ObjectLockedException :` if the diagram is currently locked

`ClassCastException :` if targets does not contain instances of `Node`

d.unlinkPath(path)

path (List) : the list of Node instances

Removes any path that exists between Node instances. If no link exists between two adjacent nodes in the path, these are silently ignored i.e., no exception will be thrown.

Exceptions:

ObjectLockedException : if the diagram is currently locked

OwnerException : if any objects in the path are not owned by this diagram

d.unlinkUpdater(updater, updatable)

updater (Node) : the updater node

updatable (Node) : the target node

Removes any update link between the updater and the updatable. It is expected that updater implements the Updater interface while the updatable implements the Updatable interface.

Exceptions:

OwnerException : if any node parameters are not owned by this diagram

ObjectLockedException : if the diagram is currently locked

## NodeFilter Objects

This is used to define arbitrary criteria for filtering nodes e.g. during searches.

n.accept(node) : boolean

node (Node) : the node

Returns True if the node should be included by the filter.

## Stream Objects

Subclass of Diagram, PropertiedObject, ParameterProvider.

This is the top-level container used to assemble Node objects into a connected "flow". It also provides the environment for setting which may be used to modify node behaviour.

s.close()

Closes the current stream. If the stream is already closed, this method does nothing. No further operations can be applied to a closed stream.

s.getContentProvider() : ContentProvider

Returns the ContentProvider for this stream. The content provider manages additional content on behalf of applications.

s.getGlobalValues() : GlobalValues

Returns the global values computed for this stream. Global values are constructed and updated by executing a Set Globals node.

s.getID() : string

Returns the temporary session ID of this object. A new ID is allocated each time a new stream is created or opened and the ID is not persisted when the stream is saved. This means that if the same persisted object is re-opened multiple times, each object will have a different ID.

```
s.getServerConnectionDescriptor() : ServerConnectionDescriptor
```

Returns the ServerConnectionDescriptor used to connect this stream to a server or None if the stream has not yet been connected or if the owner session was not created using SessionFactory.

```
s.isClosed() : boolean
```

Returns True if the stream has been closed, False otherwise.

```
s.isConnected() : boolean
```

Returns True if this has a server connection.

```
s.isExportable(format) : boolean
format (FileFormat) :
```

Returns True if this stream can be exported using the supplied FileFormat or False otherwise.

```
s.runAll(results) : ExecutionHandle
```

results (Collection) : an empty collection that will contain any built objects once execution has completed

Executes the stream synchronously and waits for it to complete. Returns an ExecutionHandle which can be used to access the exit status and any result from the task.

Exceptions:

OwnerException : if the stream was not created by this session

ObjectLockedException : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running another task, cannot execute the task or if execution completes in a state other than SUCCESS

```
s.runScript(results) : ExecutionHandle
```

results (Collection) : an empty collection that will contain any built objects once execution has completed

Executes the stream script synchronously and waits for it to complete. The stream script is always run regardless of whether script execution is set as the default behaviour. Returns an ExecutionHandle which can be used to access the exit status and any result from the task.

Exceptions:

OwnerException : if the stream was not created by this session

ObjectLockedException : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running another task, cannot execute the task or if execution completes in a state other than SUCCESS

```
s.runSelected(nodes, results) : ExecutionHandle
```

nodes (Node[]) : the array of Node objects to be executed

results (Collection) : an empty collection that will contain any built objects once execution has completed

Executes the supplied array of nodes synchronously and waits for them to complete. There must be at least one node in the array. Returns an ExecutionHandle which can be used to access the exit status and any result from the task.

Exceptions:

OwnerException : if the nodes are not all owned by this stream

ObjectLockedException : if the stream is locked

ServerConnectionException : if the stream is not connected to a server

SessionException : if the session is already running a stream, another task, cannot execute the task or if execution completes in a state other than SUCCESS

IllegalArgumentException : if the array is empty

## PublishedImage Objects

The result of publishing a DataWriter.

p.getImageContent() : byte[]

Returns the contents of the image file.

p.getMetadataContent() : byte[]

Returns the contents of the metadata file, or None if there is no metadata file.

p.getOutputDataModel() : DataModel

Returns a data model which describes the fields exported by the image. This is the server-side data model and differs from the export data model of the published node in that fields typically have known storage but default measure.

p.getParameterContent() : byte[]

Returns the contents of the parameter file.

## ReportBuilder Objects

Subclass of DocumentBuilder.

This encapsulates the functionality of a node that builds a report.

r.getReportOutput() : ReportOutput

Returns the most recent ReportOutput produced by this node, or None if this ReportBuilder has either not be executed before or failed to produce an object during the most recent execution.

## ReportOutput Objects

Subclass of DocumentOutput.

This encapsulates the concrete results of executing DocumentBuilder nodes that produce report-type outputs such as quality reports.

## RowSetBuilder Objects

Subclass of DocumentBuilder.

This encapsulates the functionality of a node that builds an output based on a RowSet.

```
r.getRowSetOutput() : RowSetOutput
```

Returns the most recent RowSetOutput produced by this node, or None if this RowSetBuilder has either not be executed before or failed to produce an object during the most recent execution.

## RowSetOutput Objects

Subclass of DocumentOutput.

This encapsulates the concrete results of executing DocumentBuilder nodes that produce objects that are based on underlying RowSet such as tables and matrices.

```
r.getRowSet() : RowSet
```

Returns the RowSet underlying this output object.

## TerminalNode Objects

Subclass of Node.

This encapsulates the functionality of any node that terminates a particular sequence of nodes.

## Updatable Objects

Subclass of Node.

This represents Node objects that can be updated by Updating objects. Updating can happen either by modifying the contents of an updatable object, or by replacing it in the stream. An updatable object can only be associated with one updater at a time.

```
u.getUpdater() : Node
```

Returns the Node that is updating this one or None if there is no updater.

```
u.getUpdaterID() : string
```

Returns the id of the node that is updating this one or the empty string if there is no updater.

```
u.isUpdatingEnabled() : boolean
```

Returns True if this node can be updated by an updater. This value can be set or returned regardless of whether an updater has been specified.

```
u.setUpdatingEnabled(canUpdate)
canUpdate (boolean) :
```

Sets whether this node can be updated by any associated updater.

## Updater Objects

Subclass of Node.

This represents `Node` objects that update `Updatable` objects. Updating can happen either by modifying the contents of an updatable object, or by replacing it in the stream. An updater can have multiple updatable objects associated with it.

```
u.getUpdatableCount() : int
```

Returns the number of `Updatable` nodes that this is updating.

```
u.getUpdatables() : Iterator
```

Returns an iterator for the `Updatable` nodes that this is updating.

# Notices

This information was developed for products and services offered worldwide.

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 (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 Software Group
ATTN: Licensing
200 W. Madison St.
Chicago, IL; 60606
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 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and 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.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

# Index

## T



## U



## V

Printed in USA