



PUBLIC

2020-08-28

SAP Predictive Analytics Developer Guide

Content

- 1 Overview 6**
- 2 What's New in This Release. 9**
- 3 What You Can Do as an Administrator or Developer. 10**
- 4 What is Predictive Analytics for OEM?. 13**
 - 4.1 Main Concepts. 14
 - Model. 15
 - Transform. 18
 - Protocol. 20
 - Context. 21
 - Factory. 22
 - Store. 23
 - Space. 23
 - Case Iterator. 25
 - 4.2 Class Diagrams. 25
 - Control API. 26
 - Data Access API. 27
 - 4.3 Implementations. 27
 - Differences between Schemes. 28
 - 4.4 Distributed Computing. 29
 - Internationalization. 30
 - 4.5 Usage Scenarios. 30
 - Preparing the Data. 30
 - Training a Model. 30
 - Applying a Trained Model. 32
 - Using a Model for Simulation. 33
 - 4.6 Features Recapitulation. 33
- 5 Using the Automated Analytics API. 34**
 - 5.1 Import. 34
 - 5.2 Configuration. 34
 - Loading a Configuration File. 35
 - Performing Direct Calls. 35
 - Configuration Keys. 36
 - Licensing Your Program with the C++ API. 38
 - 5.3 Common Operation Workflow. 39

	Declaring the Java Variables.	39
	Getting the Class Factory.	41
	Loading a License File.	42
	Building a Model.	42
	Setting Datasets.	43
	Setting Parameters.	44
	Training a Model.	46
	Displaying the Results.	47
	Using a Model.	48
	Saving a Model.	49
	Loading a Model.	49
	Releasing the Current Model.	50
	Deleting a Model.	51
5.4	Message Management.	51
	Progress Report Messages.	52
	Message Translation.	52
5.5	Sample Scripts.	53
	Prerequisites.	53
	Java Sample Scripts.	53
	Python Sample Scripts.	54
5.6	Integration.	56
	In-Process Integration.	56
	Client/Server Integration.	58
5.7	Segmented Modeling in the Automated Analytics Engine.	59
	Filter Syntax.	62
6	Integrating Generated Code.	66
6.1	What's New in Integrating Generated Codes.	66
6.2	About Code Generation.	66
6.3	Available Implementations of Code Generation.	68
6.4	Generated Codes.	70
	AWK Code.	70
	C Code.	70
	CCL Code.	71
	C++ Code.	72
	HTML Code.	77
	Java Code.	77
	PMML Code (3.2 version).	80
	SAS Code.	84
	SQL Code.	84
	UDF Code.	86
	VB Code.	90

7	Model Parameter Reference	91
7.1	Model Generation Parameters	91
7.2	Infos	93
7.3	Protocols	96
	Variables	97
	Protocol Parameters	106
	Transforms	107
7.4	DataSets	195
	Parameters	195
7.5	Plan	207
7.6	External Executables	209
	ExternalExecutableAvailable	209
	External_Executable_Name	210
	Other Parameters	212
8	KxShell: SAP Predictive Analytics Command Line Interpreter	214
8.1	Overview	214
8.2	Regression and Classification	215
	Basic Script Using Text Files	215
	Basic Script Using an ODBC Source	216
	Using the Default Cutting Strategy: Random	217
	Changing the Cutting Strategy: Periodic	218
	Changing the Compression Parameter	219
	Excluding a Variable and Changing the Target	219
	Applying a Model Using Text Files (Scoring)	220
	Applying a Model in an ODBC Source (Scoring)	221
8.3	Segmentation	222
	Basic Script Using Text Files	222
	Basic Script Using an ODBC Source	223
	Using a Classification Model to Characterize a Cluster	223
	Applying a Model Using Text Files	225
	Applying a Model in an ODBC Source	226
8.4	KxCORBAShell	227
9	Integrating with the Data Access API	229
9.1	Architecture Elements	229
9.2	Integration of a User-Defined Data Storage	230
9.3	Minimum Implementation	231
	Space	231
	Store	232
	Case Iterator	233
	Refinement Steps	233

	Compilation Process.	234
9.4	Library Installation.	235
9.5	Declaration of a New User Class Name.	235
10	Appendix.	237
10.1	Data Type Mapping.	237
10.2	File Format Specifications.	240
10.3	Language ISO Codes.	244

1 Overview

The *SAP Predictive Analytics Developer Guide* describes how to integrate Predictive Analytics functions into your environments and products.

You can find the following information:

Information	Description
Your role when using Predictive Analytics for OEM	What You Can Do as an Administrator or Developer [page 10]
The latest features	What's New in This Release [page 9]
The main concepts and contents of the Automated Analytics API	Main Concepts [page 14]
A sample scenario that takes you through the complete process of how to create and use a model	Common Operation Workflow [page 39]
How to integrate Predictive Analytics for OEM for in-process or client/server use	Integration [page 56]
The set of input and output parameters that can be found in the parameter tree of each object	Model Parameter Reference [page 91]
How to write scripts for data-mining tasks using the proprietary KxShell tool	KxShell: SAP Predictive Analytics Command Line Interpreter [page 214]
How to access data using the Data Access API	Integrating with the Data Access API [page 229]

Audience

This guide is intended for integrators familiar with the use of Software Development Kits and Application Programming Interfaces.

Documentation Changes

Version	Changes
3.3	<p>This documentation has been fully edited and reorganized.</p> <p>This documentation now provides information about the Python scheme of the Automated Analytics API. Information about C++ Common Interface scheme has been also updated.</p> <p>The guide <i>Integrating Generated Code</i> has been merged to this documentation. See What's New in Integrating Generated Codes [page 66].</p> <p>A section about user roles has been added, see What You Can Do as an Administrator or Developer [page 10].</p>
3.2	<p>This documentation is provided in a brand new HTML format.</p>

API References

SAP Predictive Analytics API is available in different integration schemes (Java, C++, CORBA,...). While the objects and calls manipulated in the API have the same name from one scheme to another, there are only minor changes from one scheme to another. Find the following [Automated Analytics API reference](#) on the SAP Help Portal.





Reference	Description
Authentication Server	The API to be used when communicating with a Predictive Analytics Authentication server.
Java	The common interface for all integration schemes in Java
C++	The C++ wrapper used to develop for C++ in-process or CORBA out-of-process
CORBA	The CORBA API used to communicate with a Predictive Analytics CORBA server.
Python	The wrapper designed to make use of SAP Predictive Analytics in Python
KxShell	The KxShell script syntax and commands
Data Access	The C API to define a specific data access to SAP Predictive Analytics

Data Access API Sample Files

The library to be implemented by the integrator is specified through the `KxDataAccess.h` header file, which describes all the API calls to implement when creating a new UserStore. Sample files of a UserStore code are also available. Find [these files](#) on the SAP Help Portal.

Learn More

Go one step beyond in the exploration of the Python API by reviewing the following tutorials on the SAP community:

- [Train and save an Automated Analytics model](#) 
- [Load and debrief the model](#) 
- [Export the scoring equation](#) 
- [Make predictions](#) 

2 What's New in This Release

This guide presents the latest release of Predictive Analytics for OEM, which corresponds to the version 3.3 of the Automated Analytics API.

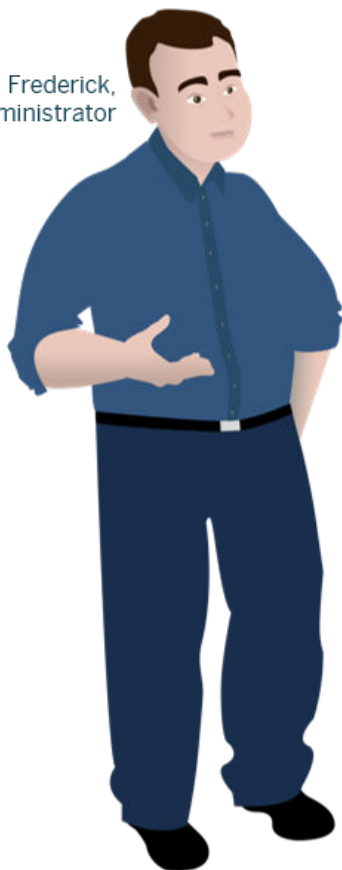
What's New:

- Support of a Python scheme for the Automated Analytics API. See [Implementations \[page 27\]](#).
- Python sample scripts are available in the installation directory. See [Sample Scripts \[page 53\]](#).

3 What You Can Do as an Administrator or Developer

The Administrator Role

Frederick,
Administrator



You're an administrator?

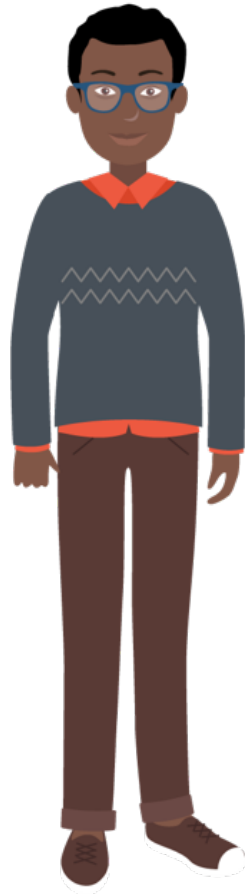
Click the action you want to perform.

 [Configure the Automated Analytics API](#)

- [Configuration \[page 34\]](#)

The Developer Role

Tom,
Developer



You're a Developer ?

Click the action you want to perform.



Create and run predictive models using a scripting language



Python



KxShell



Use the Automated Analytics API to embed SAP Predictive Analytics into your solution



In-process (Java, C++)



Client/server (CORBA)



Connect SAP Predictive Analytics to your data storage solutions using the Data Access API



Deploy predictive models using generated codes



C



C++



Java



SQL UDF

- [Using the Automated Analytics API \[page 34\]](#)
- [KxShell: SAP Predictive Analytics Command Line Interpreter \[page 214\]](#)
- [Integrating with the Data Access API \[page 229\]](#)
- [Python Sample Scripts \[page 54\]](#)
- [About Code Generation \[page 66\]](#)
- [In-Process Integration \[page 56\]](#)
- [Client/Server Integration \[page 58\]](#)
- [C Code \[page 70\]](#)
- [C++ Code Generator Framework Presentation \[page 72\]](#)
- [Java Code \[page 77\]](#)

- [SQL UDF \[page 86\]](#)

4 What is Predictive Analytics for OEM?

Learn about the whole architecture of Predictive Analytics for OEM.

Predictive Analytics for OEM provides advanced data analysis functions that can be embedded into third-party products and environments.

What is Predictive Analytics for OEM?

A component library to be integrated in final applications or software

This is not a standalone statistical environment. Its components can be integrated into full-blown environments such as SAS or SPSS.

An engineering product

It integrates some state of the art algorithms, with their associated engineering heuristics and scientific methodology describing the usage constraints. Selected algorithms must be able to give some a-priori estimations such as the estimated memory usage, and the estimated time to completion, and a way to assess the validity of their results.

A tool for non-statisticians

In particular, it must be able to check that it is used in a proper environment and to warn the users in understandable form for any violation of usage constraint. It allows non-statisticians to quickly apply the provided algorithms on proprietary data.

Who can use Predictive Analytics for OEM?

- Any application that could embed Automated Analytics
- Any application that could be packaged with SAP Predictive Analytics

Capabilities

Training, running, deploying and maintaining predictive models are its main capabilities. It relies on the Automated Analytics API that allows you to develop and perform the following types of predictive analysis:

- Clustering analysis
- Classification and regression analysis
- Time series analysis
- Social network analysis

What is not Predictive Analytics for OEM?

A scientific laboratory

Instead of giving all the possible tuning parameters to the end-user, the product can use some proven heuristics associated to the mathematical foundations to allow a fast design of operational solutions, and to reduce the number of tuning parameters.

A complete data analysis environment

SAP Predictive Analytics for OEM does not offer extended data pre-processing and data visualization facilities. There are a lot of tools on the market, and the most complex or domain specific pre-processing should be done within the data sources.

A vector for scientific spread

Most of the internal technology remains the sole property of SAP and is subject to patent.

4.1 Main Concepts

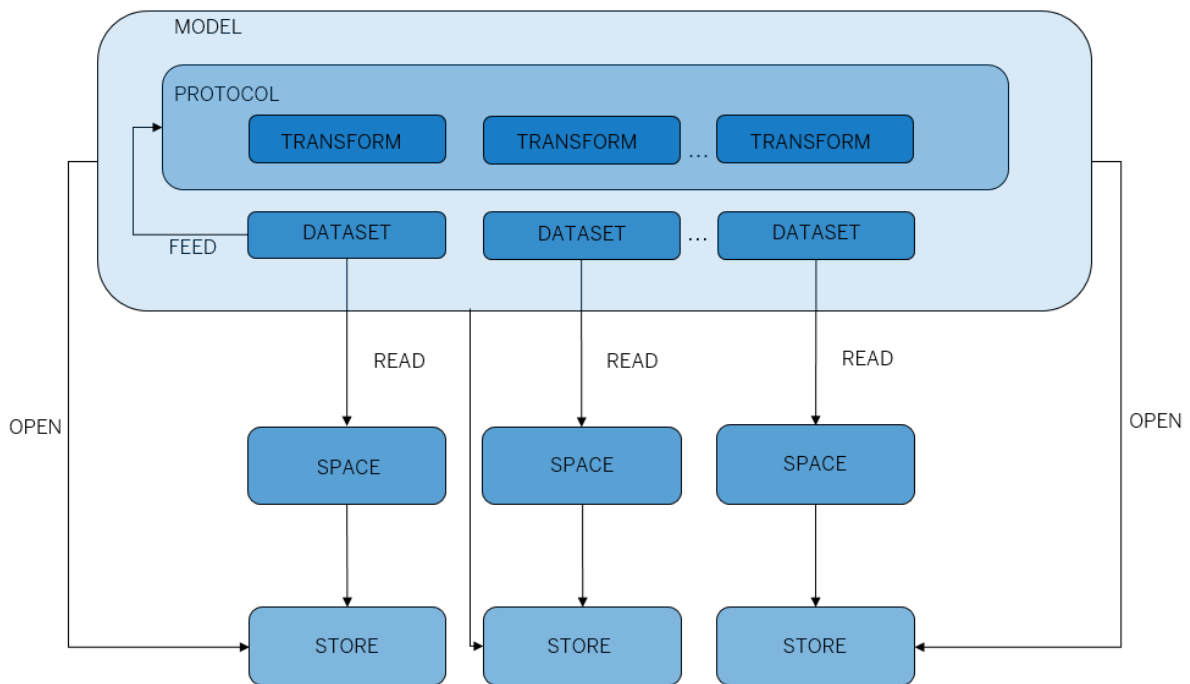
The concepts listed in this topic are the main concepts that you manipulate when working with the APIs. They are divided in two categories:

- The Control API, allowing the end user to create models:
 - Model
 - Transform
 - Protocol
- The Data Access API
 - Store
 - Space
 - Case Iterator (not in the following diagram, see [Architecture Elements \[page 229\]](#))

A list of datasets and a list of protocols define a model. Models and datasets can be saved and restored from stores.

You will also find descriptions of useful interfaces that are Context and Factory.

This image is interactive. Hover over areas for a short description.



- [Transform \[page 18\]](#)
- [Space \[page 23\]](#)
- [Space \[page 23\]](#)
- [Store \[page 23\]](#)
- [Protocol \[page 20\]](#)
- [Model \[page 15\]](#)

Related Information

[Context \[page 21\]](#)

[Factory \[page 22\]](#)

[Case Iterator \[page 25\]](#)

[Integrating with the Data Access API \[page 229\]](#)

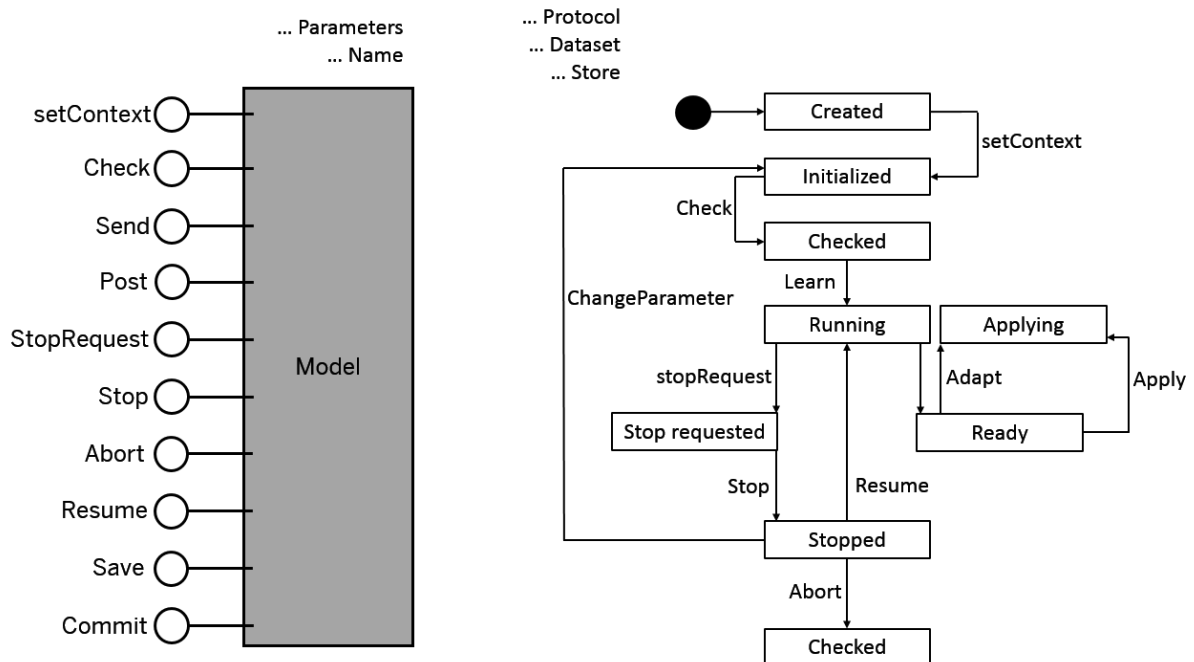
4.1.1 Model

Models are responsible for managing state transitions of the data analysis tasks and for connecting the components used in this data analysis, such as the data sources and the transforms. Models are the only entity that can be saved and restored between working sessions.

The module defines specialized components for the following analysis tasks, described in the next sections:

- Regression
- Classification

State Diagram



1. Initialize and set the context and some characteristics such as names, parameters, protocols, datasets, and stores. Parameter changes are validated through the `validateParameter` method. Every model can show a parameter view of its own datasets and protocols under the sections 'Datasets' and 'Protocols' of the parameter tree.
2. Check consistency between the components such as the datasets and the transforms. To do so you need to proceed as follows:
 1. Check that all datasets can be opened and that the transforms are initialized.
 2. Check or enforce that all datasets have the same variable descriptions.
 3. Ask the transforms to do the following:
 - To check the required datasets are present.
 - To check the compatibility of the variable types and eventually install some extra transforms.
 - To set the normalization type and reject policy.
 - To set the unknown reject policy.

i Note

SAP Predictive Analytics is able, at this point, to estimate the memory load and computing time. It also activates the portion of the state diagram which shows that the 'running' state can be interrupted, allowing a user interface to:

- Stop the process to change some characteristics or to compute some intermediate results.

- Resume/Abort the execution. The learn/adapt/apply phase can lead to several internal phases as decided by the transform.

3. Adapt/Learn/Apply the service to/of the dataset. The `send` method runs the process directly and the `post` method creates an internal thread to return immediately to the frontend.
4. Extract/Interpret the adaptation/learning/application results through the examination of the result parameters.

Related Information

[Regression \[page 17\]](#)

[Classification \[page 17\]](#)

4.1.1.1 Regression

This predictive model builds a mapping to associate a value to some input variables (presented as a Case). The Training data set must provide examples. An example is made of a case containing the input variables associated with one target (dependent) value. Cases can be made of variables of different types: continuous, nominal or ordinal.

The target variable must be coded as a number, which implies it is a continuous type. When the target variable is discrete (which means that the task is actually a classification forced into a regression), the model will code this target. This code is generally effected by associating '1' (for example) to each case having a specific discrete value and '-1' to the others.

There are two main types of transforms used in regression: some are purely numeric algorithms, which means that they only accept numbers as inputs. For these algorithms, nominal variables must be coded with numbers. The other type is symbolic and only accepts symbols as inputs. For these algorithms, continuous variables must be coded into ranges.

Depending on the transform, the model will apply some intermediate transforms in order to adapt the variables coding to the final algorithm or some intermediate transforms add extra variable based on the input variables in order to add more information on the model. For example, a date variable will be reencoding into multiple variable that describe the day of year, or the day of week.

Regression model results are made of some statistics on the difference between the target values and the generated values on the training set.

4.1.1.2 Classification

This predictive model builds a mapping to associate a class to some input variables. The training data set must provide examples. An example is made of a case containing the input variables associated with one target (dependent) value. Cases can be made of variables of different types: continuous, nominal or ordinal.

The target variable must be a discrete symbol representing the class. When the desired variable is continuous (which means that the task is actually a regression), the model will code the target. This code is generally

obtained by associating one symbol to the target values above the median and another symbol for the target values lower than the median.

Again, there are two main types of transforms used in classification: some are purely numeric algorithms, which means that they only accept numbers as inputs. For these algorithms, nominal variables must be coded with numbers. The other type is symbolic and only accepts symbols as inputs. For these algorithms, continuous variables must be coded into ranges.

Depending on the transform, the model will apply some intermediate transforms in order to adapt the variables coding to the final algorithm.

Classification model results are computed using misclassification costs, and lift curves on the training set. In the present release, classification can only be made between two classes.

4.1.1.3 Segmentation (Clustering)

Segmentation models groups case contained in a training data set. This clustering generally relies on a notion of distance between cases. Most of the times, the system tries to represent the data set through specific cases (that could be synthetic) that are called prototypes. It is used to understand some underlying coherent behaviors in the events.

4.1.1.4 Forecasting

Forecasting models are special cases of regression. Specialization is done through some specific pre-processing of time varying values. Such values are called signals and the extraction of the periodicity of these signals allows to restrain the search space used by the transforms, and gives better results than the pure regression techniques.

4.1.1.5 Data Representation

The purpose of data representation is almost every time linked to data compression or how to find a way (axes) to represent the cases with the minimum loss of information allowing performing a certain task. Most of the times, this data compression can only be done if the user knows how data will be used after compression: for example, an attribute that could be regarded as noise for one task could be very important for another one.

4.1.2 Transform

Transforms are interfaces generating two types of information:

- Synthetic information that helps you understand a set of events. For example, some basic statistics such as the mean and standard deviation of variables. This information is stored into the transform itself as a result.

- Information saved as new items or numbers for each event or case into databases or files. For example, a score associated with a credit risk for a particular customer.

Transforms hold the actual statistical or machine learning algorithms and must go through a learning phase to be ready for use. This learning phase is done at the model level and used for the estimation of parameters and the computation of some results or descriptive information.

i Note

Although frontend users cannot initiate learning or adaptation phases, for specific uses it is possible to test a transform and store the results in a temporary space. This particular mechanism saves intermediate results. However, users do not need to know anything about the implementation details of the transform algorithms.

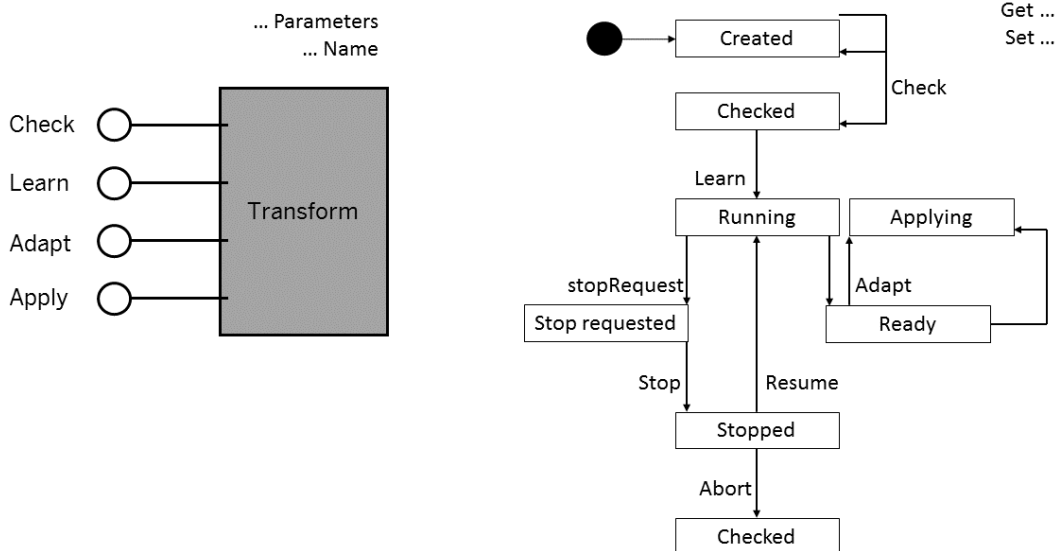
Transforms must be customized using parameters. However, all components are designed to keep the number of user-defined parameters as small as possible. The frontend is only allowed to edit the names and the parameters. The models actually control the transforms.

Internal parameters that are computed during 'Learn' or adapted during 'Adapt' can be accessed by the frontend through the parameters under the `Results` directory. Results are read-only.

When the frontend wants to stop the running computation, it asks for the `Model::stopRequest` method. This sends the appropriate event. It is up to the computation thread to process the stop request at some breakable points.

State Diagram

The following state diagram shows the possible transition paths from the 'Created' to the 'Ready' state.



4.1.2.1 Classification and Regression

The classification/regression engine builds models implementing a mapping between a set of descriptive variables (inputs of the model) and a target variable (output of the model). It belongs to the family of the so-called regression algorithms. This is a proprietary algorithm that is our own derivation of a principle described by V. Vapnik called "*Structural Risk Minimization*". The first quality of the models built with SAP Predictive Analytics classification/regression engine is their robustness, which makes them very useful for very noisy data sets. The second quality of these models is that they are based on polynomial functions, which are well known mathematical objects. Furthermore, it is very easy to interpret the order 1 polynomial coefficients into variable contributions. This allows end-users to make fruitful variable selection based on the robust information extracted from the data.

The learning phase is very fast: only tens of seconds for a problem on 50,000 cases described with 15 attributes.

4.1.2.2 Data Encoding

The data encoding engine is one of the corner stone of the application's features preprocessing utilities. Its purpose is to find robust encoding of discrete variables into ordinal numbers that can be injected into numerical algorithms such as Neural Networks, and the classification/regression engine.

The data encoding feature fits with the following constraints:

- Robust representations of discrete variables values,
- Reduced number of data access,

The target variable must be easily transformed into a number (number targets are thus accepted). When applied in a regression model, there is no target transformation. When applied in a classification model, the class less frequent is coded as a 1, and the other class as a 0 (or -1 depending on the succeeding transform).

The data encoding engine encode all discrete variables possible values with a number related to their rank in terms of target mean. Its only originality is to eliminate poorly represented values.

4.1.3 Protocol

Protocols are responsible for holding together a chain of interfaces containing the algorithms, named transforms, from the raw data to their encoding. The protocol passes through the final transforms in charge of information extraction and generation.

You can choose the variable role through protocols. The four roles are:

Role	Characteristics
skip	The transform chain doesn't use the variable.
input	The transform chain uses this variable input.

Role	Characteristics
<code>target</code>	The transform chain uses this variable target value.
<code>weight</code>	The variable indicated the relative weight case versus the others. This mechanism allows giving more stress to some specific cases in a database.

Strange Values

Information extraction can manage the problem of processing strange values, which are either missing or out of range. This problem is solved through a set of policies that are implemented by the protocols during the check phase.

Each protocol is characterized by a policy that can take one of the three following values:

- `Transform deals`
The protocol asks its transforms to discuss together in order to process strange values. This is the default mode.
- `skip`
The protocol forces all transforms to use a `skip` action, meaning that the cases with strange values will be discarded both during training or application.
- `Warning`
This level is associated with warnings that are generated for every strange value. If this level is higher than the context level, the user will not see any warnings.

i Note

You can change these three parameters between training and application phases.

The protocol says if it accepts strange values that could be generated by the last transform. To do so, it uses two Boolean parameters, which are defaulted as false. The default behavior is thus to process empty values or out of range values generated by the last transform.

4.1.4 Context

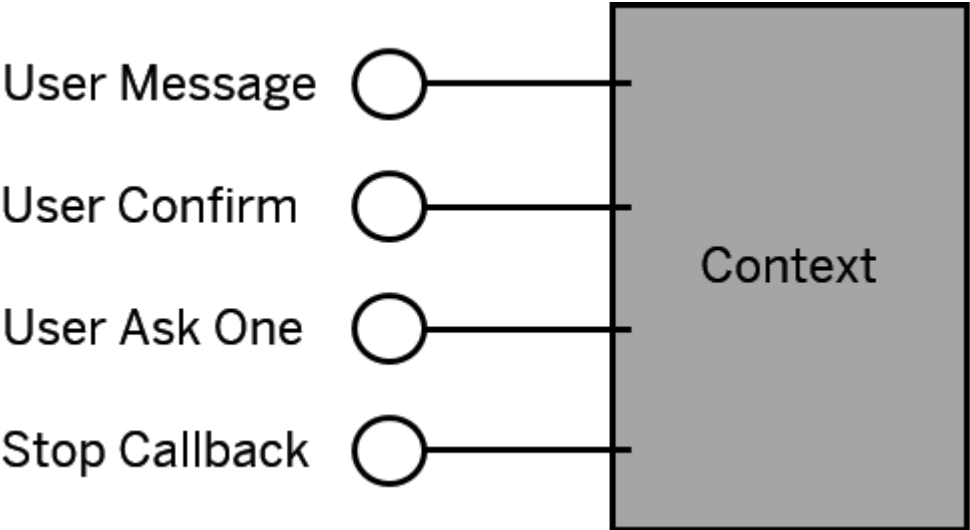
The Context interface provides a way for the user to integrate models within in-house environments or applications. It is basically a handle to four callback procedures that are called when events to be displayed to the user occur.

'`userMessage`' is the method called by any component that want to inform the front end of some event. Messages are build from a keyword and some arguments. A topic manager links this keyword with a message template, thus allowing very easy internationalization or customization of the messages. Messages are associated with a level. Only messages, with a level lower than the context level, are displayed to the user (level 0 means an error).

When a component decides it needs confirmation from the user, it calls the `userConfirm` method. This displays a prompt to the user and asks for confirmation. It returns a Boolean value to the caller component.

Sometimes, especially, when restoring a model from a store, a component will require a value to the user frontend (logins and passwords for example). It does so by using the `userAskOne` method.

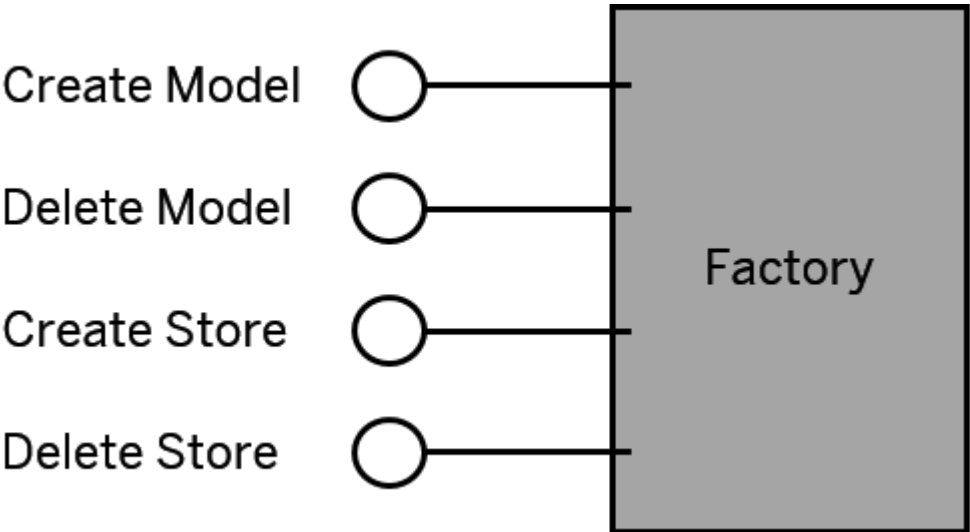
Finally, when a running component hits a request for stop, it asks for the `stopCallback` method that asks the user for the next operation which can be abort or resume.



4.1.5 Factory

The Factory implementation and cardinality depends on the interface framework (C++ or CORBA). Its role is to allow the creation of components by user frontends.

Besides the component creation role, the Factory collects information about the host machine such as duration of basic operations (add, multiply operators...) and available memory.



4.1.6 Store

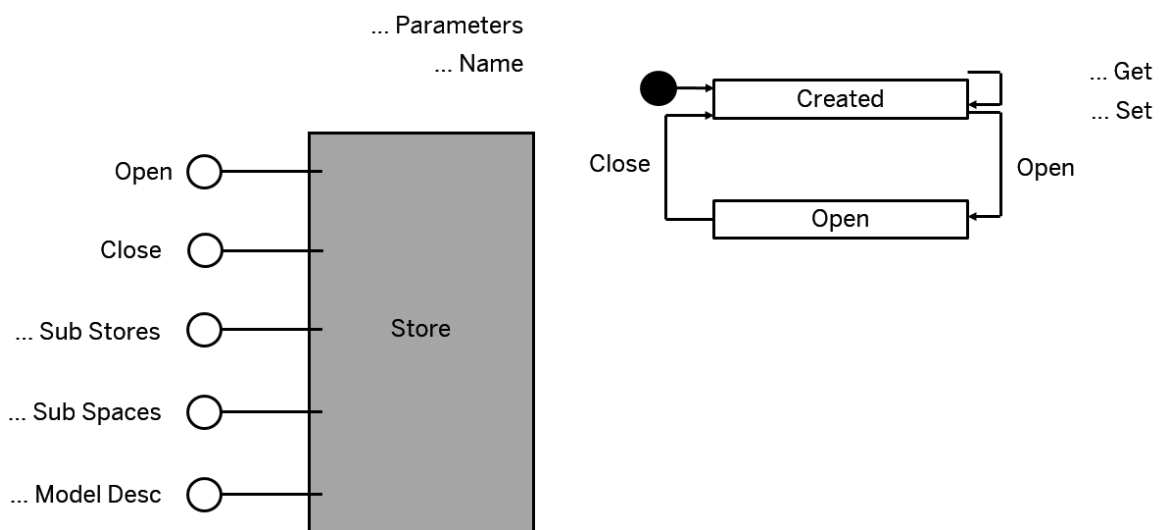
Stores federates several physical spaces and help frontend designers to show available stores, spaces and models. Stores can be viewed as a file directory, a database, or an Excel workbook.

Models can be saved into specific spaces contained in stores, allowing one to view the models saved into a store and to restore the model.

A model description contains:

- Its name
- Its class name
- Its version
- The date of its last save
- The space name of its last save

State Diagram



The two operations on store are `Open` and `Close`.

`Open` allows to access data spaces in a given location with a user and a password that can be empty strings.

When a store is open, it can be asked for its subspaces. There is one specific subspace called 'KxAdmin' which holds the descriptions of models saved in this store.

4.1.7 Space

Spaces are responsible for preparing the environment to fetch the data from their actual sources, and for the description of these variables composing a case. The actual fetch is made through the creation of case iterators.

i Note

A space is generally associated with a model through a role name. This association between a name and space is called a dataset.

A variable description contains several information items:

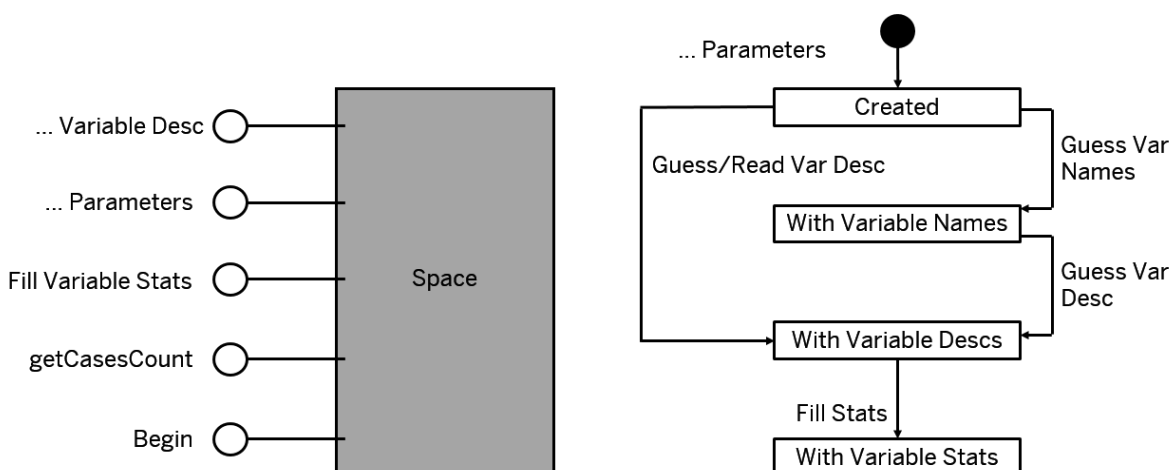
- Name
- Description
- Storage type (number, string, and date)
- Value type (discrete, continuous, ordinal, date)
- Level indicating if the variable is considered as a key that could be used to retrieve this sample in the data sources
- Level indicating if the variable corresponds to an order of the cases
- Group indicating if the variable must be considered with others

From the user frontend, the only calls that are available after a space creation (through a store or a model), are about parameter edition. The user can also ask to read a specific description file that holds the variable descriptions.

All the other operations are done internally within the components and are just shown here to present the interactions between the spaces and models or protocols. Most of the operations are either about the access or guess variable descriptions or statistics. They mainly consist of opening a new space within an open store and, via the `begin` method, creating case iterators that are used by all subsequent protocol stacks to read or write data within a space.

A space can only be opened in 'read' or 'write' mode and not both. If the user wants to perform both, it has to open two spaces, one in 'read' mode and the other in 'write' mode.

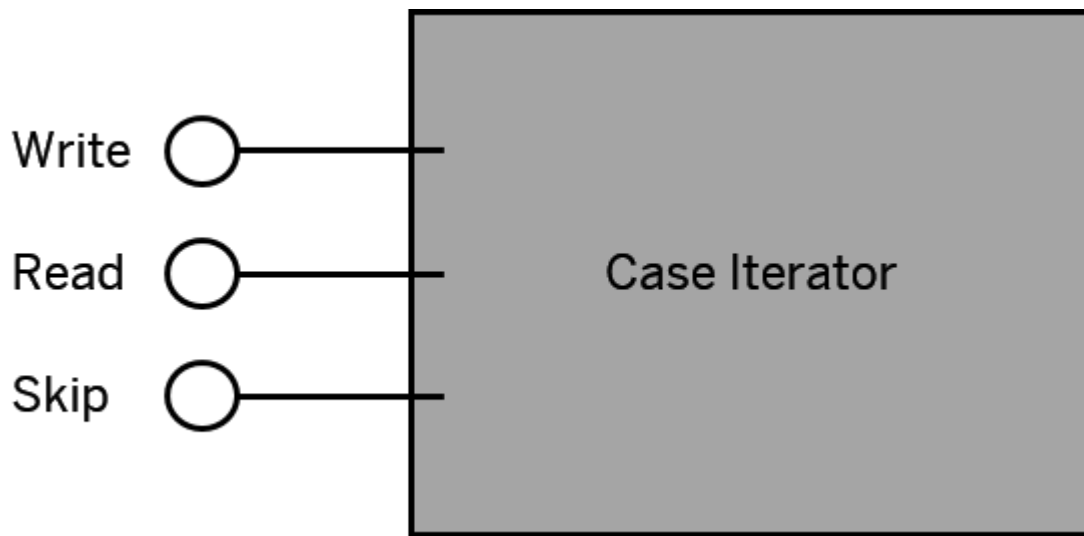
State Diagram



4.1.8 Case Iterator

Case iterators are responsible for reading cases from opened spaces and are classical design patterns used in all state of the art environments.

The main operations consist of actioning case iterators to fetch or write the next line, and to access cells within an iterator (get or set values). Cells will be accessed accordingly to their storage types defined in the description and cells can have empty values.



4.2 Class Diagrams

The Automated Analytics API can be viewed as a set of two APIs:

- The Control API allows you to create, estimate, and validate descriptive or predictive models.
- The Data Access API allows you to extract information from data sources of different formats such as text files, or relational databases tables. It has been specifically designed to minimize memory consumption and can be used on large databases and data warehouses.

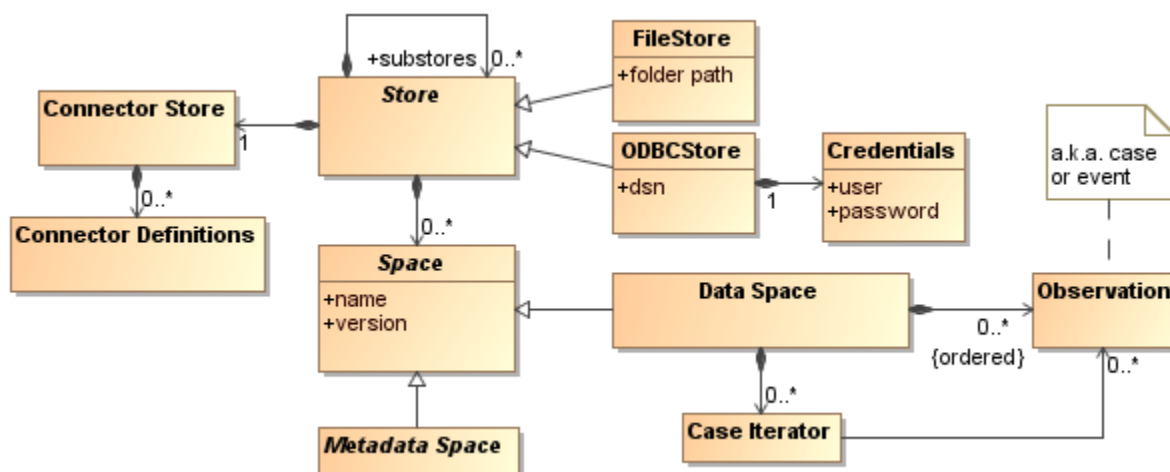
The purpose of the application components is to provide executable modules that embeds objects described in this overview. See the two following topics to view two class diagrams that illustrate the main concepts of the API.

Related Information

[Main Concepts \[page 14\]](#)

4.2.2 Data Access API

The following class diagram describes the inner structure of the Data Access API.



4.3 Implementations

You can use the Automated Analytics API through different schemes. All the API calls have the same parameters and the same semantic in all integration schemes.

Scheme	Description
C++	This scheme involves a simple dynamic library.
Java Native Interface (JNI)	This scheme uses a JNI layer (this option may not be available on Linux platforms). Here, two dynamic libraries are used (a Java Native wrapper library and the standard C++ library).
CORBA	The CORBA object model that you can use with a wide range of programming language such as C++, Java, or Python. In this case, the API appears as a standalone executable, which is accessed by the CORBA communication layer.
JNI and CORBA Common Interface	It is provided on top of JNI and CORBA Java implementations to facilitate the switch from one environment to another, and facilitate the integration process.
C++ and CORBA Common Interface	It is provided on top of the C++ native and CORBA C++ implementations. This scheme, provides more natural C++ API (error by exceptions, function has no output arguments, but returned values), and the same types are used in the in-process or client/server integration. Thus, it allows to switch easily from in-process to client/server integration (as only a few lines may differ).
Python	In Python language directly, through a SWIG layer. It relies on a Python module and on two dynamic libraries, that is, a SWIG layer and the SAP Predictive Analytics C++ library.

The following table resume the different schemes and the different integration paths.

Scheme	In/Out Process	Binaries	Headers	Platforms
C++	In-process	Dynamic library (KxCPP.dll)	Kxen_CPP.h	All (Windows, Linux)
C++ Common Interface	In-process	Dynamic library (KxCPP.dll)	KxCommonInterf.h, KxCPPInterf.h, KxCPPInterf.cpp, KxFrontEndInterf.h, Kxen_CPP.h	
C++ Common Interface CORBA	Out-of-process	Executable Authenticated Remote Server	KxCommonInterf.h, KxCORBAInterf.h, KxCORBAInterf.cpp, KxFrontEndInterf.h	
Java with JNI	In-process	Dynamic library (KxCPP.dll, KxenJni.dll)	KxJni.jar Optionally: KxCommonInterf.jar and KxJniInterf.jar	All
Java Common Interface JNI	In-process	Dynamic library (KxCPP.dll, KxenJni.dll)	KxJni.jar, KxCommonInterf.jar, KxJniInterf.jar	All
Java Common Interface CORBA	Out-of-process	Executable Authenticated Remote Server	KxCorba.jar, KxCommonInterf.jar, KxCorbaInterf.jar	All
CORBA	Out-of-process	Executable Authenticated Remote Server	IDL file Kxen_CORBA.idl and Kxen_AuthServer.idl	All
Python	In-process	Dynamic library (_aalib.pyd or _aalib.so, KxCPP3.dll)	aalib.py	All

4.3.1 Differences between Schemes

- The initial Class Factory creation is scheme-dependent.
- Some parameters types differ from one integration scheme to another, and the native type scheme is used wherever it is possible (for example, strings). However, we have try to keep type definition similar when

possible (for example, JNI parameters are defined in the same way than CORBA ones). See the table Type mapping for a description of the different types.

- The way output parameters are handled is scheme-dependent:
 - In C++, references to pointers (for example, for models), or literal types (for example, for integers) are used, and they are filled on success.
- In CORBA, JNI, C++ and Python, the main entry point in Predictive Analytics Components is the Predictive Analytics Class Factory, which allow the creation of all objects.
- Error management is scheme-dependent:
 - Using the common interface java framework, all error codes are returned through a generic `KxenRuntimeException`
 - In others framework no exception are thrown out from the kernel. Instead all functions return an error code to the caller that indicate if the function succeed or not.
 - Using the common C++ interface, error codes are also managed through exceptions.
 - Using the Python wrapper, all error codes are returned through Python exceptions derived from `aalib.AAException`.

4.4 Distributed Computing

Some of the design choices of the components were made because these components are designed and used in distributed computing environments such as CORBA. This section presents some of these choices.

One of the major things that could strike people familiar with distributed computing designs is that not all objects can be created remotely, even if most of the objects can be accessed through remote interfaces.

For example, frontends cannot create data spaces and transforms, they must go through a model to do so. This is because data spaces are class factories for case iterators and, for obvious performance reasons, these iterators must belong to the same memory space than the transforms.

So we choose to force transforms and the data spaces to be created in the same memory space. We could make one the class factory of the other but there was no philosophical reasons to choose one or the other, so we decide to choose a higher abstraction level to do so: and this was the birth of the model. Furthermore, models allow having a single entry point to save and restore full protocols, which is very handy.

The following table presents the accessibility of the objects in the distributed environment CORBA:

Remote Creation	Interface	Factory of
No (front-end)	Yes	
Yes	Yes	Parameter, Space, Store, Transform
Yes	Yes	Parameter
No	Yes	Parameter
No	Yes	Parameter

No

No

No

Yes

4.4.1 Internationalization

As it has been said the kernel uses complex mathematical functions that could encounter problems very deeply in their execution. Front-end applications must/can be informed of a lot of events that occur during fairly long processes. These events come from a large variety of problems. This is why a resource mechanism, that can be easily customized with a text editor, has been integrated within the components.

The core software uses an internal messaging service based on these resources. This system allows, in a distributed environment, several front-ends (with several languages) to communicate with a single modeling server.

4.5 Usage Scenarios

Four usage scenarios are described in the next sections:

- Preparing the data
- Training a model
- Applying a previously trained model
- Using a model for simulation

4.5.1 Preparing the Data

All training scenarios are based on using a training dataset in order to adapt the values of the transform parameters to a particular task. It is the responsibility of the external user of the software to prepare this dataset before entering this scenario.

Preparation here only means collect the data into a single file, table, view or SQL statement compatible with an ODBC driver. This dataset can come into a single dataset called the 'training' dataset, or several datasets, called 'estimation', 'validation', and eventually 'test' datasets. In this step the user declares the directory in which the files are stored or the ODBC source in which the table will be accessed.

4.5.2 Training a Model

Training a model requires the following tasks:

Creating a Model

Create a model (to perform a classification for example).

Accessing Metadata

Variable or column of every training dataset must be described in terms of storage detailing the string, number, date, datetime, angle and value whether it is nominal, ordinal, continuous or textual.

This variable can be taken as an identifier and used later by the system to synchronize the newly generated values at the proper place. The API provides a facility to guess this information directly from the data through SQL statement and a scan of the first 50 lines of the file by using the metadata, except if a `KxDesc_"dataset name"` file exists in the same directory.

Managing Variables

The user can exclude some variables from the study. The variables are not taken as inputs of the model to generate the target variable which the user chooses. The user can choose either a continuous variable, in this case the problem solved is a regression problem, or a Boolean variable, of which the problem is a two-classes classification problem.

Associating Characteristics and Transforms

This is done through a generic structure called `Parameter`. The design of this hierarchical structure is close to the Microsoft® Windows registry. It is a very versatile structure reachable by varying text based or graphical user front ends.

As an example, for a classification model only one parameter is needed, which is the order of the polynomial models created. For data encoding, the only important parameter is the profit strategy, used only for classification problems to associate a profit to each of the two possible classes and thus compare some models, based on their relative profit.

Training the Model

The model checks variable descriptions and their compatibility with the chosen transform to be sure that the training is valid. Internally, this creates a `CaseIterator` that scans through the data space to pass information to the transforms. Memory management for this iteration is handled by a run time class called `Stack`.

`CaseIterator` and `Stack` are volatile objects that cannot be seen from the external world. Missing and out of range values appearing in the cases that are processed by this stack are automatically taken care of.

The training process returns two main indicators: the Predictive Power (KI) and the Prediction Confidence (KR). The predictive power is an indicator between 0 and 100% that measures the amount of the target variable explained by the model using all the input variables. The prediction confidence is an indicator between 0 and 100% that measures the robustness of the final system. This robustness indicates how the user can trust the performance on the training dataset, to be a good predictor of the performance of the system on new data. As a general business rule, a model can be trusted if this indicator is above 95%.

Interpreting the Results

Each transform returns specific piece of knowledge on the training dataset:

- The data encoding engine:
 - Returns the intrinsic importance of each variable independently from the others, each variable being characterized by its own predictive power.
 - Compresses the categories of the variables that do not bring robust information into a single 'miscellaneous' category called 'KxOther'.
 - Ranks the categories of nominal variables in terms of importance, meaning the balance between the fact that the target average of cases of this category is different from the target average on the entire dataset, and the frequency of this category.

As an example, a category can have a large importance either because the average of the target in this category is very high compared to the others, or because it is slightly above the global average. This particular category is very frequent.

- The classification or regression engine returns the contribution of each variable. The notion of contribution indicates how much the model is using a variable relative to the other variables. This contribution is given to help decision process about which variables are important in order to build a robust model of the target. It comes in two forms: the intrinsic contribution as used by the model, and a special view of this contribution that allows us to take into account the correlation between variables, that can be used to perform variable selection.
- All SAP Predictive Analytics transforms generate profit curves for each input or generated variable. This profit curve can be used in order to compare the information and robustness contained in any variable versus a business question. Of course, using the profit curve of the variable generated by the model gives a very easy and direct interpretation of the quality and robustness of the model.

Saving the Model

It saves the entire protocol and the training dataset description.

4.5.3 Applying a Trained Model

In this scenario the user can either start from a model generated in the previous scenario or reload a previously trained model.

The user applies the model on new data. This requires that the user specify a new space in which input variables values are stored. It is very important that this space has the same structure than the one used for

training. No further consistency checks are made. This constraint also concerns the target variable that must be present in this data set (even if the actual values are yet unknown). This application also requires the user to specify an output space and a write strategy (append or write). The 'Write' strategy facilitates the replacement of previous results, and the 'Append' strategy allows the user to keep track of the previous results and add new ones.

4.5.4 Using a Model for Simulation

In this scenario, the user asks the model to be applied on a single case that is filled, variable by variable, from a user interface.

This simulation capability allows the user to test the results of the application of the model. When the problem is a regression problem, the user can request an error bar on top of the actual estimation. When the problem is a classification problem, the user can request a probability instead of a score.

4.6 Features Recapitulation

Here are some key elements to the functional specification of the application.

- Can process number, string
- Can process nominal, ordinal or continuous variables
- Can build robust regression or classification models
- Can extract data from text files or ODBC sources
- Automatic processing of strange values (missing and out of range)
- Automatic computation of profit curves
- Can save models in both text files and ODBC tables
- Provide a single and simple mechanism for parameter access of the components.
- Manage model configurations (versions)
- Simulation capability (one case at a time)

User front ends can specify internally created components through the generic parameter interface. This is available for Models and Stores on the one hand, and Spaces, Protocols, and Transforms on the other hand (dark gray in the previous schema).

To take advantage of the international messaging service offered by the application, all end user applications have to define a specific adapter called Context. The context is a call back handler that allows the components to inform the user front end of progress.

SAP Predictive Analytics provides data source classes to access text files, and ODBC compliant data sources for Windows and Linux environments (light blue in the previous schema). Extra data source types can be added, depending on the user demands, such as DBMSCopy, Excel and OLE DB (yellow in the previous schema).

5 Using the Automated Analytics API

Learn how to use the Automated Analytics API.

This section presents the functions to be called in the API for the main modeling functionalities. Java Samples [../EXE/Clients/Samples/java] described in this tutorial are also available.

5.1 Import

The following table presents the object definitions to import in your development environment for the different schemes:

Scheme	Import
C++	<pre>#include "Kxen_CPP.h"</pre>
C++ Common Interface	<pre>#include "KxFrontendInterf.h"</pre>
C++ Common Interface CORBA	<pre>#define KX_FRONTEND_CORBA #include "KxFrontendInterf.h"</pre>
Java JNI	<pre>import com.kxen.KxModelJni.*;</pre>
Java CORBA	<pre>import com.kxen.KxModel.*;</pre>
Java Common Interface	<pre>import com.kxen.CommonInterf.*;</pre>
C++ with CORBA	<pre>#include "Kxen_Client.h"</pre>
Python	<pre>import aalib</pre>

5.2 Configuration

You configure the SAP Predictive Analytics kernel to perform the following tasks:

- Specifying the location of your license file

- Localizing the application or adapting the user messages
- Adding some functionalities, like the possibility to define external data reader using the Data Access API
- Specifying some constants to use through the system

You can configure the SAP Predictive Analytics kernel by setting key/value pairs through one of the following API calls:

- `loadAdditionalConfig` to load an external configuration file or table
- `setConfiguration` to set the key/value pairs directly

The configuration is loaded at startup in both cases.

5.2.1 Loading a Configuration File

A configuration file is a text file that contains a set of key/value pairs. Each line contains a key, a tabulation, and the associated value. You must make sure to keep the tabulation while editing the file to avoid loading issues. You can also specify the configuration in an ODBC table if ODBC is available on your system.

Note

- Standalone `KxCORBA` and `KxShell` executables load their own configuration file at startup (`KxCORBA.cfg` and `KxShell.cfg`). The files are located in the same directory as the executables.
- The configuration files of the standard distribution load the license file provided by SAP Predictive Analytics that is also a configuration file.

Call the `loadAdditionalConfig` function to load the configuration.

Example

```
KxenStore lStore = mFactory.createStore("Kxen.FileStore");
lStore.openStore("c:/ConfigDir", "", "");
lStore.loadAdditionalConfig("MyConfig.cfg");
```

5.2.2 Performing Direct Calls

You can set the key/value pairs directly instead of loading the configuration from a text file.

Call the `setConfiguration` function for each key/value pair you want to set.

Example

```
// Change the default Admin table name
```

```
mFactory.setConfiguration("KxAdmin", "ModelAdmin");
// Set the license
mFactory.setConfiguration("KeyCode", "B7X60-0SY0800-1234567-0PP1");
```

5.2.3 Configuration Keys

The following table details the different configuration keys supported by SAP Predictive Analytics components.

Key	Description
KxAdmin	Sets the name of the administrative file or table to be used to store SAP Predictive Analytics model entries. The default value is <code>KxAdmin</code> . When components save a model in a store (a directory or an ODBC Source), it also creates or updates the administrative file (<code>KxAdmin.txt</code>) or table (<code>KxAdmin</code>). When SAP Predictive Analytics lists the available models in a store (a directory or an ODBC source), it only browses this administrative file. You can change this value if you want to change the name of this administrative file or table. Note that the change is global, and that it may prevent you from retrieving existing models.
MessageFile	Adds an additional message file for SAP Predictive Analytics. Message files are used to translate error codes and kernel message in a proper language. The country code associated to the message file is retrieved from the file name (which should be <code>XXXX_<country>.umsg</code>). Multiple <code>MessageFile</code> entries are supported; they are added to the engine.
FileStoreRoot	Adds new entry in the list of File "Root stores". File Root stores is the list of store retrieved by SAP Predictive Analytics, when one asks for the list of available store (<code>store.lsDirGet</code>) at the top position ("", the blank string).
TempDirectory	Sets the directory used for temporary internal storage. This directory might be used by certain components for temporary internal storage.

Key	Description
UserStore	<p>Declares a new possible Store class, using the Data Access API. This allows declaring that a Dynamic Library is available on the system to access external data. The value associated with this key should be a couple of string, separated by a ':' character:</p> <ul style="list-style-type: none"> • a symbolic name, that will be used in <code>createStore</code> calls. • the name of an accessible dynamic library. Only the radix part of the library filename needs to be specified. The suffix part of the library filename will be added on a platform dependent way (.dll on Windows, .so on Solaris, .sl on HP/UX).
Config	<p>Loads an additional configuration file or table. The name of the space is relative to the current configuration space (file or table). This configuration key is not available through "setConfiguration". Note that if the specified configuration file does not exist it is not considered as a fatal error, and it is silently ignored.</p>
CustomerId	<p>Sets the <code>CustomerId</code> license key. The string value associated with this key is the encrypted key provided by SAP Predictive Analytics through the license file. This entry is generally set by the license file (which is in turn set by a <code>Config</code> entry).</p>
EngineMode	<p>Sets the <code>EngineMode</code> license key. The string value associated with this key is the encrypted key provided by SAP Predictive Analytics through the license file. This entry is generally set by the license file (which is in turn set by a <code>Config</code> entry).</p>
Kxen.*	<p>Entries of this type, where * stands for a Components name (for example, <code>Kxen.RobustRegression</code>, or <code>Kxen.ConsistentCoder</code>) enable a license entry for a specified component. The string value associated with this key is the encrypted key provided by SAP Predictive Analytics through the license file. This entry is generally set by the license file (which is in turn set by a <code>Config</code> entry).</p>
TraceLevel	<p>Creates a log file. It takes as parameter the value corresponding to the message level you want to display in the log file generated (<code>kxlog.txt</code>).</p>

5.2.4 Licensing Your Program with the C++ API

If you are developing a program by using the C++ scheme of the Automated Analytics API, you first need to load a license file to be able to use the Predictive Analytics engine. SAP recommends you load the `KJWizard.cfg` configuration file to set up a global, complete, and clean configuration of the kernel for your future developments. The configuration file references the required license files.

i Note

Make sure that files are referenced with relative paths in the global configuration file as shown below.

1. Then in your program, call the `KxModel::IKxenStore_ptr::openStore` function:

```
openStore("C:/Program Files/SAP BusinessObjects Predictive Analytics/
Server/EXE/Clients/KJWizardJNI", "", "");
```

i Note

- Always specify the absolute path to the `KJWizard.cfg` file as name of the store.
- Always use slash ("/") instead of backslash ("\") as path separator.

2. Call the `KxModel::IKxenStore_ptr::loadAdditionalConfig` function to load the configuration file to the server from this store.

```
loadAdditionalConfig("KJWizard.cfg");
```

i Note

Make sure that the file name is passed as argument, not its absolute path. You can also specify this additional file relatively to the store name.

Example

```
Key=Value
FileStoreRoot=UserDir
FileStoreRoot=../../Samples
FileStoreRoot=DefaultRoots
LogConf=logconf.txt
MessageDirectory=../../Resources/Messages
KTCStore.StoreClass=Kxen.FileStore
KTCStore.StoreOpen=../../Resources/KTCData
#uncomment to define your own ktc store
#UserDefinedKTCStore.StoreClass=Kxen.FileStore
#UserDefinedKTCStore.StoreOpen=KTC_Test_Data\Test_Rules
KxDesc=KxDesc
KxAdmin=KxAdmin
# Comment to activate the Explain feature
DataAccessExplanation.*.Activated=false
SKDXml=../../Resources/BusinessObjects_KCDefinitions_dfo.xml
Config=$UserDir/.SAP_AA_License.cfg
Config=../../License.cfg
Config=../../KxStatTr.cfg
Config=../../DataCacheManager.cfg
```

```
Config=../../../../FastWrite.cfg
Config=../../../../SparkConnector/Spark.cfg
```

i Note

If the configuration file cannot be found, the error 2147024894 E_SYSTEM__NOENT occurs when calling `loadAdditionalConfig`. Check the path.

Once the license is loaded, you may get the error `KXEN_E_ABSTRACTTRANSFORM_BADLICENSE` if the key code defined in `License.cfg` is not correct or if `BusinessObjects_KCDefinitions_dfo.xml` is not reachable.

5.3 Common Operation Workflow

In this section you can find code snippets of the `KXTutorial.java` sample script that describe the workflow of commonly used operations.

Samples are provided in Java language using the Java common interface layer, which allows you to switch easily from CORBA (client/server mode) to JNI (standalone application).

1. [Declaring the Java Variables \[page 39\]](#)
2. [Getting the Class Factory \[page 41\]](#)
3. [Loading a License File \[page 42\]](#)
4. [Building a Model \[page 42\]](#)
5. [Setting Datasets \[page 43\]](#)
6. [Setting Parameters \[page 44\]](#)
7. [Training a Model \[page 46\]](#)
8. [Displaying the Results \[page 47\]](#)
9. [Using a Model \[page 48\]](#)
10. [Saving a Model \[page 49\]](#)
11. [Loading a Model \[page 49\]](#)
12. [Releasing the Current Model \[page 50\]](#)
13. [Deleting a Model \[page 51\]](#)

5.3.1 Declaring the Java Variables

Declare the following variables to be used in the sample codes:

- Variables to initialize the Automated Analytics engine
- Default parameters for the input data access
- Default values for saving the model
- Default values for applying the model
- A series of other default values

Example

```
// ----- VARIABLES OF THE CLASS -----
// ----- Variables to initialize the Automated Analytics engine -----
// -----
/** Place containing the Config file */
private static final String CONFIG_DIR = ".";
/** Name of the config file */
private static final String CONFIG_FILE = "KxConfig.cfg";
// ----- Default parameters for the input data access -----
/** Default value for the main transformation of the model */
private static String sModelTransformation = "Kxen.RobustRegression";
/** Default value for the kind of store used for reading data */
private static String sStoreClass = "Kxen.FileStore";
/** Default value for the Name of the store to open, in this case it is the path
of the current directory. */
private static String sStoreName = ".";
/** Default value for the Name of the data file to open as a space */
private static String sDataFile = "Census01.csv";
/** Default value for the description file to use for the data file. */
private static String sDescriptionFile = "Desc_Census01.csv";
// ----- Default values for saving model -----
/** Default value for the Name of the store */
private static String sModelStoreClass = "Kxen.FileStore";
/** Default value for the Store name. It is a path of the upper directory. */
private static String sModelStoreName = ".";
/** Default value for the Space Name. Name of a file to append or create in the
disk. */
private static String sModelSpaceName = "MyModel.mdl";
/** Default value for the Name of the model to be saved */
private static String sModelName = "MyModel";
/** Default value for the Name of the store */
private static String sModelComment = "Model generated by KxTutorial.java";
// ----- Default values for applying the model -----
/** Default value for the Name of the space to open in the main store to use the
model. */
private static String sUsingModelInput = "ApplyFile.txt";
/** Default value for the Name of the space to put the results of the model on
the data set to use. */
private static String sUsingModelOutput = "Output.txt";
// ----- Other default values -----
/** Model object. */
private KxenModel mModel = null;
/** Object Factory. */
private KxenClassFactory mFactory = null;
/** Transform object. */
private KxenTransform mTransform = null;
/** Transform unique name. */
private String mTransformName = "";
/** Context class used to display messages */
private KxContext mContext;
/** Language variable used to specify to the context class which messages to
print. */
private String mLanguage = "us";
/** Level limit of the messages to print by context object */
private int mMsgLevel = 6;
/** Choose a method for the communication with the Automated Analytics server */
private boolean mUseSend = false;
```

Task overview: [Common Operation Workflow \[page 39\]](#)

Next task: [Getting the Class Factory \[page 41\]](#)

5.3.2 Getting the Class Factory

First, the client application must initialize the SSL Layer. Then call the `getFactory` function to get the Class Factory.

1. Activate the SSL Layer:
 - a. Make sure the file `KxCorbaSSL_1_5.jar` is in the classpath of the application. It can be found in the `CorbaJars` folder of the installation directory.
 - b. Set this file as default socket factory by doing one of the following:

Option	Description
Running the following command	<code>-Dcom.sun.CORBA.connection.ORBSocketFactoryClass=com.kxen.CorbaSSL.Sun1_5.MyORBSocketFactoryClass</code>
Calling the <code>System.setProperty()</code> method	<code>System.setProperty("com.sun.CORBA.connection.ORBSocketFactoryClass", "com.kxen.CorbaSSL.Sun1_5.MyORBSocketFactoryClass");</code>

2. Call the `getFactory` function.

JNI:

```
mFactory = KxenClassFactory.getJNIFactory(CONFIG_DIR, CONFIG_FILE);
```

CORBA:

```
String[] lArguments = new String[6];
lArguments[0] = "-ORBInitialHost";
lArguments[1] = "localhost";
lArguments[2] = "-ORBInitialPort";
lArguments[3] = "12345";
lArguments[4] = "-ServiceName";
lArguments[5] = "FactoryEntries3";
mFactory = KxenClassFactory.getCORBAFactory(lArguments);
```

Authenticated Server:

```
String[] lArguments = new String[6];
lArguments[0] = "-ORBInitialHost";
lArguments[1] = "localhost";
lArguments[2] = "-ORBInitialPort";
lArguments[3] = "12345";
lArguments[4] = "-ServiceName";
lArguments[5] = "KxAuthServer3";
KxenAuthenticatedServer lServer =
KxenAuthenticatedServer.getAuthenticatedServer(lArguments);
KxenConnection lConnection = lServer.connect("login", "password");
mFactory = lConnection.getFactory();
```

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Declaring the Java Variables \[page 39\]](#)

Next task: [Loading a License File \[page 42\]](#)

5.3.3 Loading a License File

You are required to load a license file in order to build a model. There are three ways to load a license file:

- Loading the license file like a configuration file
- Including a reference to the license file in a configuration file
- Using direct API calls

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Getting the Class Factory \[page 41\]](#)

Next task: [Building a Model \[page 42\]](#)

5.3.4 Building a Model

A model is the holder of the different Transforms and holds the whole process chain from raw data to the final outputs.

Request the model to the Class Factory:

```
mModel = mFactory.createModel("Kxen.SimpleModel");
```

i Note

The string `Kxen.SimpleModel` is a keyword to one possible model definition. Currently, only this type of model is supported.

Once a model is created, it does not include any Transform. You can add a SAP Predictive Analytics Modeler - Regression/Classification (K2R) engine in it, by doing one of the following:

```
int lTransformPlace = mModel.pushTransformInProtocol("Default",  
sModelTransformation);  
mTransform = mModel.getTransformInProtocol("Default", lTransformPlace);  
mTransformName = mTransform.getName();
```

Such a simple model is able to perform general regression or classification task. However, to perform a regression or a classification model, the training dataset must be encoded. SAP Predictive Analytics provides a component (Consistent Coder) to encode your training dataset. This component could be explicitly or implicitly added in your learning process.

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Loading a License File \[page 42\]](#)

Next task: [Setting Datasets \[page 43\]](#)

5.3.5 Setting Datasets

You define datasets to be used with the current model. A dataset is a physical data support, like a text file or an ODBC table, associated to the model through a role, which defines how the model is going to use the data.

To define a dataset, you must set the following:

- A store, for example the directory or the ODBC source where the space can be found
- A space name, for example the file name, the table name, or a SQL statement
- A role, for example "Training", "Estimation" or "ApplyOut"

In the following procedure, you create a training dataset:

1. Open the directory:

```
/** Default value for the kind of store used for reading data */
private static String sStoreClass = "Kxen.FileStore";

/** Default value for the Name of the store to open, in this case it is the
path of the current directory.
*/private static String sStoreName = ".";

int lStoreIndex = mModel.openNewStore(sStoreClass, sStoreName, "", "");
```

i Note

There is no user nor password used here because the directory where data are stored is a filestore, hence the double quotes.

2. Declare the training dataset in the store:

```
/** Default value for the Name of the data file to open as a space */
private static String sDataFile = "Census01.csv";

mModel.newDataSet("Training", sDataFile, lStoreIndex);
```

3. Read or guess the data descriptions by calling one of the following functions:

- Load them:

```
/** Default value for the description file to use for the data file.
private static String sDescriptionFile = "Desc_Census01.csv";

mModel.readSpaceDescription("Training", sDescriptionFile,
lStoreIndex);
```

- Analyze them through the system:

```
mModel.guessSpaceDescription("Training");
```

The variables are created in the protocol so that they can be accessed for parameter modifications. By default, the current algorithm will select the last compliant variable as target.

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Building a Model \[page 42\]](#)

Next task: [Setting Parameters \[page 44\]](#)

5.3.6 Setting Parameters

Parameters are trees of key/values pairs. Before running the model, you can tune the following parameters:

- High-level parameters of the model
The model has some high-level parameters such as the one used to split a single training dataset into three Training/Validation/Testing datasets.
 - Variable descriptions
Each variable has some descriptive parameters (type, storage, keys...) that can be adjusted, and also has a role with respect to the current model.
 - Transform parameters
Each transform defines its own set of parameters that generally control the algorithm used in the transform, for example the regression polynomial degree used in K2R.
1. Call the `getParameter` function on the object with an empty string ("") to retrieve its parameter tree.
 2. Set the correct values to the corresponding node in the tree.
 3. Validate or commit the changes made to the parameter tree.

Example

Set the `CutTrainingPolicy` of the model to the value `random` and the role of the variable `class` to `target` through the following script:

```
KxenParameter lParameterP = mModel.getParameter("");  
lParameterP.release();  
mModel.changeParameter("Parameters/CutTrainingPolicy", "random");  
mModel.changeParameter("Protocols/Default/Variables/class", "target");  
mModel.validateParameter();
```

i Note

`getParameter` and `validateParameter` are expensive function calls, as they convert an object internal state into a tree of parameters or revert the parameter tree into the internal state. You can group the `changeParameter` calls for each object, in order to limit the number of such calls. See the documentation Components Parameters for an in-depth description of such parameters.

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Setting Datasets \[page 43\]](#)

Next task: [Training a Model \[page 46\]](#)

5.3.6.1 Variable Roles

A role is defined for each variable. The following table describes the available roles and their corresponding code:

Role	Code
Excluded variable	skip
Target variable	target
Weight variable	weight
Explanatory variable	input

i Note

- `skip` for all variables which have a `KeyLevel` different from 0 (key of your dataset)
- `target` for the last compliant variable if the current algorithm requires a target

Example

For example, to exclude the age variable, call the `changeParameter` function and set the variable `age` to `skip` as follows:

```
mModel.changeParameter("Protocols/Default/Variables/age", "skip");
```

5.3.6.2 Getting Variable Values

To retrieve a list of variables with values, proceed as follows:

1. Call the `getParameter` function:

```
mModel.getParameter("")  
mModel.getParameter("Protocols/Default/Variables")  
gKxenModel.ParaModel.getSubEntries "Value", lstrTemp1, lstrTemp2
```

2. Parse the two strings `lstrTemp1` and `lstrTemp2` to get arrays:

```
lVarName = Split(lstrTemp1, Chr(10))  
lValue = Split(lstrTemp2, Chr(10))
```

i Note

`Value` can be replaced to obtain all roles, storages and so on.

5.3.7 Training a Model

- The model is defined.
- The transform protocol is established.
- The input data is set.
- The parameters are tuned.

You can now train the model.

Send a "Learn" message by doing one of the following:

Option	Description
Synchronous call	<pre>mModel.sendMode(KxenModeTag.Kxen_learn, 0);</pre> <p>The <code>sendMode</code> function is a synchronous call, which means it waits until the model training is completed.</p> <p>Note The 0 refers to a store that could be used for a temporary storage, if needed by the model.</p>
Asynchronous call	<pre>mModel.postMode(com.kxen.CommonInterf.KxenModeTag.Kxen_learn, 0); while (mModel.isRunning()) { try { // To avoid too much communication Thread.sleep(400); } catch (InterruptedException e) { System.err.println("Waiting method failed"); } }</pre> <p>The <code>postMode</code> function is an asynchronous call, which means it waits while the model state is running.</p> <p>Note The loop above corresponds to the one implemented by the <code>KxShell</code> command <code>waitRunning</code>. In this case, the learning process will be fired in a separate thread. This can be used to keep a Graphical User Interface active while learning the model. For example, in Visual Basic environment a call to <code>DoEvents</code> will keep the GUI reactive. Also, retrieving information and error messages can be done here. For more information, see Message Management [page 51].</p>

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Setting Parameters \[page 44\]](#)

Next: [Displaying the Results \[page 47\]](#)

5.3.8 Displaying the Results

Once the model has been generated you will need to retrieve its results.

Parent topic: [Common Operation Workflow \[page 39\]](#)

Previous task: [Training a Model \[page 46\]](#)

Next task: [Using a Model \[page 48\]](#)

5.3.8.1 Getting Variable Statistics

Use the function `getSubValue` to get a value (`iElement`) from the parameter tree:

1. Define a dataset (`iDataset`), a target variable (`iTarget`) and a variable (`iVariable`).
2. Proceed as follows:

```
private double getStatisticsValue(String iVariable, String iDataset, String
iTarget, String iElement) {
    double lDoubleValue = 0;
    String lPathKr = "Protocols/Default/Variables/" + iVariable + "/Statistics/"
+ iDataset + "/Targets/" + iTarget + "/" + iElement;
    try {
        KxenParameter lParam = mModel.getParameter("");
        String lString = lParam.getSubValue(lPathKr);
        lDoubleValue = Double.parseDouble(lString);
    } catch (NumberFormatException e) {
        lDoubleValue = -1;
    }
    return lDoubleValue;
}
```

It returns a double value. If any error occurs on converting to double, it returns -1.

Example

Call the `getKiKr` function to retrieve the statistic values for Prediction Confidence "KR" and Predictive Power "KI":

```
private void getKiKr() {
    String lDatasetKi = "Validation";
    String lDatasetKr = "Estimation";
    String lVariable = "rr_class";
    String lClassName = "c_lclass";
    System.out.println("KR Value=" + getStatisticsValue(lVariable, lDatasetKr,
lClassName, "Kr"));
    System.out.println("KI Value=" + getStatisticsValue(lVariable, lDatasetKi,
lClassName, "Ki"));
}
```

5.3.8.2 Getting Variable Contributions

Retrieve the contribution of each variable for the current model:

1. Load variable names and role from the model using the `getParameter` function.
2. Get roles as an array of name/role using the `getSubEntries` and `getValue` functions.

Example

```
private void getVariableContributions() {
    String lParameterName = "Protocols/Default/Transforms/" + mTransformName + "/"
    Results/class/Coefficients";
    KxenParameter lParam = mModel.getParameter(lParameterName);
    DataTable lRoles = lParam.getSubEntries("Contrib");
    System.out.println("-- Contributions of each variable --");
    for (
        int lIdx = 0;
        lIdx < lRoles.getRowCount(); lIdx++) {
        System.out.print(lRoles.getRowName(lIdx));
        System.out.print(" = ");
        System.out.println(lRoles.getValue(lIdx, 0));
    }
}
```

5.3.9 Using a Model

All the output results are stored in the resulting parameter tree. You can retrieve output values computed by the model through the parameter objects, by using the `getParameter` and `getNameValue` functions. In the Java Wizard interface, all the displayed graphs are simple plots of values found in the model parameter tree after the training phase. You can also apply the model.

Apply the transformation built during the training phase on new data and produce some expected output by proceeding as follows:

- a. Set a new input dataset, with a role `ApplyIn`.
- b. Set a new output data, created by the model, with a role `ApplyOut`.
- c. Send an `Apply` message to the model with the `sendMode` function.

The data files in the following code are located in the same store than the training data. It illustrates the latest procedure:

```
// Open the Store containing the apply data
int lStoreIndex = mModel.openNewStore(sStoreClass, sStoreName, "", "");
// Set input and output datasets for Apply
mModel.newDataSet("ApplyIn", sUsingModelInput, lStoreIndex);
mModel.newDataSet("ApplyOut", sUsingModelOutput, lStoreIndex);
try {
    mModel.sendMode(KxenModeTag.Kxen_apply, 0);
} catch (KxenAbortException e) {
    System.err.println("Apply stopped");
}
```


Task overview: [Common Operation Workflow \[page 39\]](#)

Previous: [Displaying the Results \[page 47\]](#)

Next task: [Saving a Model \[page 49\]](#)

5.3.10 Saving a Model

Once you create and validate a model on the basis of KI/KR (Prediction Confidence/Predictive Power) indicators, you can save it.

1. Create a new data store to hold the model:

```
int lStoreIndex = mModel.openNewStore(sModelStoreClass, sModelStoreName, "",
    "");
```

2. Set the model name:

```
mModel.setName(sModelName);
```

3. Save the model:

```
mModel.saveModel(lStoreIndex, sModelSpaceName, sModelComment);
```

Here, for simplicity, use the same store to save the model in. A model can be indifferently saved in a text file or in an ODBC table.

Note

A `commitModel` call is also available to be able to save a new version of the same model in the same Space (file or table).

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Using a Model \[page 48\]](#)

Next task: [Loading a Model \[page 49\]](#)

5.3.11 Loading a Model

1. Create a directory reference:

```
KxenStore lStore = mFactory.createStore(sStoreClass);
```

2. Open the correct location:

```
lStore.openStore(sModelStoreName, "", "");
```

3. Load the model using the last saved version:

```
mModel = lStore.restoreLastModelD(sModelName);
```

Alternatively, instead of restoring the latest version, you can also load a specific version of a model:

```
int lModelVersion = 1;  
mModel = lStore.restoreModelD(sModelName, lModelVersion);
```

i Note

When the model is loaded, create some memory space to avoid any performance drop. To do so you need to delete the store:

```
lStore.release();
```

At that stage, the model is fully restored and ready for use. It can be applied to a new dataset or some parameters can be retrieved to display some curves, indicators and so on. For more information, see [Displaying the Results](#)

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Saving a Model \[page 49\]](#)

Next task: [Releasing the Current Model \[page 50\]](#)

5.3.12 Releasing the Current Model

When you release a model from the memory, you free all the corresponding resources on the server (for example in CORBA framework).

Use the `release` function:

```
mModel.release();
```

i Note

It also releases recursively all the objects created through the model such as Transforms, Space, Parameters and so on.

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Loading a Model \[page 49\]](#)

Next task: [Deleting a Model \[page 51\]](#)

5.3.13 Deleting a Model

To remove a model from the disk storage, call the `eraseModelD` function at the store level.

Use the `eraseModelD` function:

```
KxenStore lStore = mFactory.createStore(sStoreClass);
int lModelVersion = 1;
lStore.eraseModelD(sModelName, lModelVersion);
```

Task overview: [Common Operation Workflow \[page 39\]](#)

Previous task: [Releasing the Current Model \[page 50\]](#)

5.4 Message Management

SAP Predictive Analytics for OEM may send messages to your integration environment.

Messages can be:

- Error messages when an error occurs
- Information messages that provides progress information on the current process

The type of message is indicated by a message level as described in the following table:

Level	Type of Message	Usage
0	error	error pop-up
1	info	info pop-up
2	old progress messages	not used
3	phase message	put in a log
4	detailed information	put in a log
5	warning	put in the log
6	progress report message	in a progress bar
8	programming error	written in a log file for debugging purpose

You can implement one of the following mechanisms to manage messages:

- **Push**
You implement a class that inherits from `IKxenContext` and pass such a "context" object to the Automated Analytics API objects used. To do this, you call the `setContext` function that is available from models and

stores. All messages from these objects are forwarded to the context object in the integration environment.

- **Pull**

After each call or during long asynchronous calls done using `postMode`, you implement a loop on the `getMessage` function to retrieve the next available message. The `getMessage` function also returns the message level.

You generally need to filter and dispatch the messages according to the level of interaction needed.

5.4.1 Progress Report Messages

Progress report messages are sent to the integration environment as regular string messages.

A progress message is formatted as follows:

```
KXEN_W_PROGRESS <CompletedSteps> <TotalSteps> ...message...
```

Where:

- `<CompletedSteps>` is the current number of steps completed in the current phase.
- `<TotalSteps>` is the total number of steps to be performed in the current phase.
- The rest of the string is a text describing the current phase.
- Blank characters are separators.

Here is a sample processing of this string:

Example

This example illustrates the parsing of the message string before print:

```
StringTokenizer lTk = new StringTokenizer(iMessage);
String lMsgCode = lTk.nextToken();
int lCompleted = 0;
int lTotal = 0;
try {
    lCompleted = Integer.parseInt(lTk.nextToken());
    lTotal = Integer.parseInt(lTk.nextToken());
} catch(java.lang.Exception e) {}
String lPhase = lTk.nextToken("");
System.out.println("Phase: " + lPhase + " - " + lCompleted + "/" + lTotal);
```

5.4.2 Message Translation

The message translation is done by SAP Predictive Analytics for OEM and depends on the language requested by the application.

You set the requested language as follows:

- **Push**
In the `setContext` call, when you give the context object to the Automated Analytics API
- **Pull**
In the `getMessage` call, when you get back each message.

There is a number of message files for each language. Their name is built on the following convention:

```
Kx<Module>_<languageCode>.umsg
```

For example, the distribution provides the following language files: `KxTool_us.umsg`, `KxTool_fr.umsg`, and `KXObject_us.umsg`.

These files must be loaded in the system at the start up of the application. This is already done for server processes, such as the SAP Predictive Analytics CORBA server. For in-process integration (JNI, C++), you load them through a configuration file that lists the available message files for translation. You load the configuration file with the `loadAdditionalConfig` function.

5.5 Sample Scripts

SAP Predictive Analytics provides Java and Python sample scripts to test the Automated Analytics API.

5.5.1 Prerequisites

- You must install a valid license in the following location of your system:
 - In the `C:\Program Files\SAP Predictive Analytics\` folder on Microsoft Windows
 - In the folder where you have decompressed the `KXENAF` archive file on Linux.
- You must set up J2SDK 1.6 on your system and declare the `JAVA_HOME` environment variable.
- You must set the `PATH` environment variable to the following value:
 - `%JAVA_HOME%/bin` on Microsoft Windows
 - `$JAVA_HOME/bin` on Linux

5.5.2 Java Sample Scripts

Use the following sample scripts to run the sample scenario described in this guide.

File	Description
<code>KxTutorial.java</code>	The complete script described in Common Operation Workflow [page 39] .

File	Description
KxContext.java	An object used to retrieve SAP Predictive Analytics messages. See Message Management [page 51] .
KxTutorialAdvanced.java	An advanced scenario that scores each line of a data file (simulation).
Util.java	Common functions used in both scenarios.

5.5.2.1 Running the Java Sample Scripts on Microsoft Windows

The following batch files are located in the `C:\Program Files\SAP Predictive Analytics\Predictive Analytics\...\EXE\Clients\Samples\java\script`, where they must be executed.

1. Run the `prepare.bat` batch file to retrieve the files needed to run the sample scenario, such as configuration files and datasets.
These files are located in the `C:\Program Files\SAP Predictive Analytics\Predictive Analytics\...\EXE\Clients\Samples\java\script` directory.
2. Run the `compile.bat` batch file to compile the Java sample scripts.
These files are located in the `C:\Program Files\SAP Predictive Analytics\Predictive Analytics\...\EXE\Clients\Samples\java\src\` directory.
3. Finally, run the `run.bat` batch file to execute the scenario.

5.5.2.2 Running the Java Sample Scripts on Linux

The batch files are located in the `KXROOT/SamplesSrc/java/script` directory, where they must be executed.

1. Run the `prepare.sh` batch file to retrieve the files needed to run the sample scenario, such as configuration files and datasets.
These files are located in the `KXROOT/SamplesSrc/java/script` directory.
2. Run the `compile.sh` batch file to compile the Java sample scripts.
These files are located in the `KXROOT/SamplesSrc/java/src` directory.
3. Finally, run the `run.sh` batch file to execute the scenario.

5.5.3 Python Sample Scripts

Run the following sample scripts to test the Python scheme.

File	Description
<code>classification.py</code>	A script that demonstrates the classification model scenario with the Python scheme of the Automated Analytics API.
<code>clustering.py</code>	A script that demonstrates the clustering model scenario with the Python scheme of the Automated Analytics API.
<code>recommendation.py</code>	A script that demonstrates the recommendation model scenario with the Python scheme of the Automated Analytics API.
<code>timeseries.py</code>	A script that demonstrates the time series model scenario with the Python scheme of the Automated Analytics API.

5.5.3.1 Running the Python Sample Scripts on Microsoft Windows

Make sure you have installed Python 3.5 on your system. Possible distributions are CPython, Anaconda, or WinPython.

The installation of Python 3.5 automatically declares the `PYTHONHOME` environment variable and updates the `PATH` environment variable with the `%PYTHONHOME%` value.

1. Run the `setvars.bat` batch file to configure the `PATH` and `PYTHONPATH` environment variables.
The file is located in the `C:\Program Files\SAP Predictive Analytics\OEM\EXE\Clients\Python35` directory.
 - `PATH` is updated with the directory of the Automated Analytics C++ dynamic libraries.
 - `PYTHONPATH` is updated with the directory of the `aalib.py` file.
2. Open the sample script for editing and modify the `AA_DIRECTORY` value with the Automated Analytics installation directory, for example `C:\Program Files\SAP Predictive Analytics\OEM\ORC:\Program Files\SAP Predictive Analytics\Desktop\Automated`.
The sample scripts are located in the `C:\Program Files\SAP Predictive Analytics\OEM\EXE\Clients\Samples\Python` directory.
3. Run the sample script with the Python executable `python.exe` either on a command line or within a Python notebook, Jupyter for example.

5.5.3.2 Running the Python Sample Scripts on Linux

Make sure you have installed Python 3.5 on your system. Possible distributions are CPython or Anaconda.

The installation of Python 3.5 automatically declares the `PYTHONHOME` environment variable and updates the `PATH` environment variable with the `$PYTHONHOME` value.

1. Source the `setvars.sh` batch file to configure the `PATH` and `LD_LIBRARY_PATH` environment variables.
The file is located in the `Python35` directory of the Automated Analytics installation.

- `PATH` is updated with the directory of the Automated Analytics C++ dynamic libraries.
 - `LD_LIBRARY_PATH` is updated with the directory of the `aalib.py` file.
2. Open the sample script for editing and modify the `AA_DIRECTORY` value with the Automated Analytics installation directory, for example `/opt/AutomatedAnalyticsOem_X86-64-redhat-Linux-2.6.18-8.el5smp_v3.3`.
The sample scripts are located in the `Samples/Src/Python` directory of the Automated Analytics installation.
 3. Run the sample script with the Python executable `python` either on a command line or within a Python notebook, Jupyter for example.

5.6 Integration

The way to deploy and distribute SAP Predictive Analytics for OEM within your application depends on the kind of integration. The components and files that must be embedded in your software are reviewed in the next sections according to the different integration schemes.

5.6.1 In-Process Integration

In an "in-process" integration, the embedding software includes SAP Predictive Analytics for OEM within the same process and memory space.

C++ Integration

You install the following components and files to use the C++ native implementation of the SAP Predictive Analytics kernel.

Required?	Components and Files
Yes, this is the minimum configuration for using the basic functionality.	<ul style="list-style-type: none"> • The SAP Predictive Analytics C++ library • Some resource files and message translations for the supported languages. The location of the messages should be specified to the kernel using a configuration key or file. • For Linux, ODBC libraries, as provided by SAP Predictive Analytics

Required?

No, it depends on your modeling or data access needs.

Components and Files

- SAP Predictive Analytics Advanced Access provides access to external data files such as SAS data files or SPSS files. It is implemented as an external data access plug-in to SAP Predictive Analytics.
- A Japanese library to process Japanese language with Text Coding feature. A set of external libraries must be used (specific stemming process).
- Teradata FastLoad to improve the read/write performance of Teradata connections. It is implemented as an external data access plug-in to SAP Predictive Analytics. Teradata FastLoad must be installed on the target machine.

i Note

- These components are activated by some additional configuration keys. You can reuse existing distributed SAP Predictive Analytics configuration files.
- Some of these components may not be available for all platforms supported by the SAP Predictive Analytics kernel.

The following table gives the name of the files associated with each resource presented above according to the platform. Files between parentheses may not be available on all platforms.

Components and Files	Microsoft Windows	Linux
SAP Predictive Analytics C++ library	KxCPP3.dll	libKxCPP3.so
Resources files	Resources directory	Resources directory
ODBC Driver Manager	Included in the operating system	KxenConnectors/*
KAA	KxStatTr.dll	libKxStatTr.so
	statrn64.dll	libst.so
	iconv.dll	(libst.so.11)
	libxml2.dll	(libstodbc.so.11) (libiconv_st.so*)
Japanese library for KTC	KTCJpLib.dll	libKTCJpLib.so
	libmecab.dll	
FastLoad Access	KxFastLoadBridge.dll	libKxFastLoadBridge.so

i Note

- On Microsoft Windows, the DLL files can be found for example in the EXE\Clients\CPP directory.
- On Linux, the libraries are located in SAP Predictive Analytics `libs` directory. The libraries needed at runtime must also be included in the shared library search path (environment variable `LD_LIBRARY_PATH`).

Java Integration

Java integration is done through Java Native Interface (JNI). To deploy SAP Predictive Analytics with a Java application using JNI, you install the components and files needed for the C++ integration and the following JNI resource files:

- `KxCommonInterf.jar`
- `KxJniInterf.jar`
- `KxJni.jar`
- `KxUtils.jar`
- `KxenJni3.dll` on Microsoft Windows, `libKxenJni3.so` on Linux.

Python Implementation

Python integration is done through a SWIG wrapper. To deploy SAP Predictive Analytics with a Python script, you install the components and files needed for the C++ integration and the following Python integration layer:

- `aalib.py`
- `_aalib.pyd` on Microsoft Windows, `_aalib.so` on Linux

5.6.2 Client/Server Integration

In the following case of a client/server integration, your client application communicates with an SAP Predictive Analytics Authenticated Server. It is assumed that the server standard installer is used to configure the server.

Java Integration

The `CommonInterf` layer can be derived for CORBA communication. You must deploy the following JAR files with the application:

- `KxCommonInterf.jar`
- `KxAuthentInterf.jar`
- `KxAuthent.jar`
- `KxUtils.jar`

Integration with Other Languages

SAP Predictive Analytics does not provide specific wrappers for other languages. You must use a CORBA client implementation layer and integrate the CORBA description of the SAP Predictive Analytics Server used with these layers.

For example, using Python and a CORBA implementation (for example omniORBPy), you need to import `KxAuthServer.idl` into the development environment.

5.7 Segmented Modeling in the Automated Analytics Engine

Segmented modeling means adding a filter to a data space so only rows matching the filter will feed the Automated Analytics modeling engine.

A filter only supports:

- Any imbrications of AND & OR operators with any number of operands.
- Comparisons between a variable and a given constant.

For example:

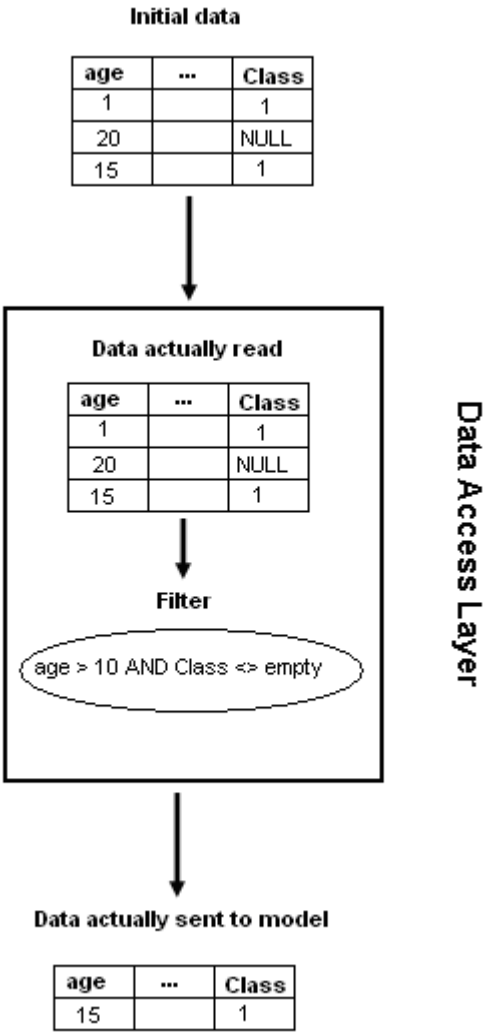
```
Age>10
Age>10 AND Class <> <empty>
Age<10 OR ( Age>10 AND Class <> empty)
```

Default Filtering

Default filtering (or abstract filtering) is used by the engine when the data storage has no native filtering capability (File, SAS File, etc.):

- Each row is read by the Data Access layer of the engine.
- If the row matches the filter, it is sent to modeling layer, otherwise it is skipped.

For the filter age>10 AND Class<>empty, the following process is applied:



Optimized Filtering

Optimized filtering is used by the engine when the data storage has a native filtering capability (DBMS):

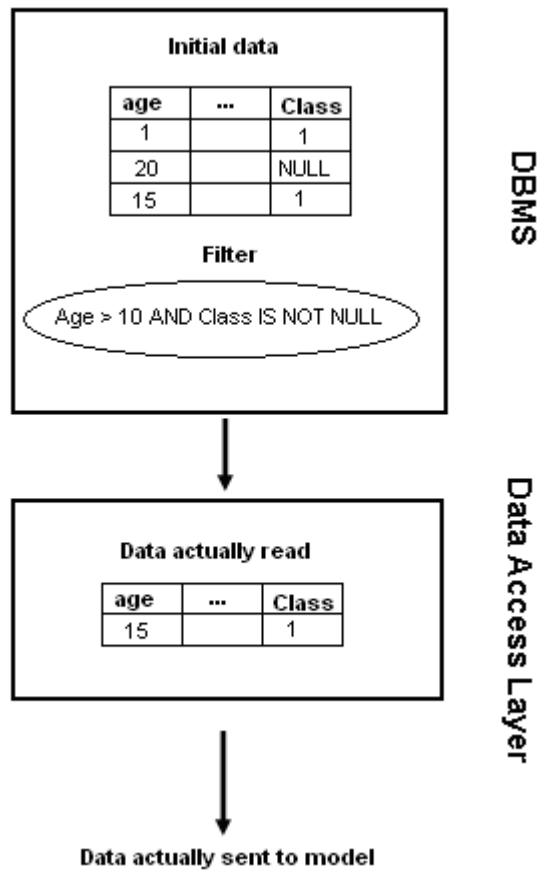
- The filter is translated into an SQL expression and added to a WHERE clause.
- Only rows matching the filter are sent by DBMS so all rows are sent to the modeling layer.

For example, the previous examples are translated into SQL as:

```
((Age>10) AND (Class IS NOT NULL))
```

So, the actual SQL used by engine to read data is:

```
SELECT * FROM <table> WHERE ((Age>10) AND (Class IS NOT NULL))
```



Filter Specifications

Logical Operators

The operators `AND`, `OR` are available. These operators can have any number of operands (>2) not only 2. The classic optimization is used which means that the evaluation stops:

- At the first operand evaluated to 'false' when operator is `AND`
- At the first operand evaluated to 'true' when operator is `OR`

Comparison Operators

The operators `<`, `<=`, `>` and `>=` are available. The usage of empty values as value to test is checked and forbidden.

The operators `=` and `<>` are available. The usage of the empty values is allowed and translated as `IS NULL` and `IS NOT NULL`.

Remarks

The filter evaluation is done after the mapping process meaning the variable names are the logical variable names and not the physical fields.

The Data Cache is compatible with Filters. And only the filtered values are stored in the cache.

The `KxIndex` variable cannot be used in a filter (it may work in a file but will trigger an error in an ODBC). This is not checked at this time.

The values of `KxIndex` in a filtered space are generated after the filtering: the standard sequence 1,2,3... is visible (and not 1,45,74... depending on the `KxIndex` values coming from the non filtered space)

5.7.1 Filter Syntax

Syntax of a filter

Here is a pseudo BNF grammar of the filter:

```
<FilterCondition>:= <LogicalOperator>{<SimpleFilter>} |  
<LogicalOperator><FilterCondition>  
<SimpleFilter>:= <Operator><Variable><Value> |  
<Operator>:=Equal|NotEqual|Greater|GreaterEqual|Less|LessEqual  
<logicalOperator>:=And|Or
```

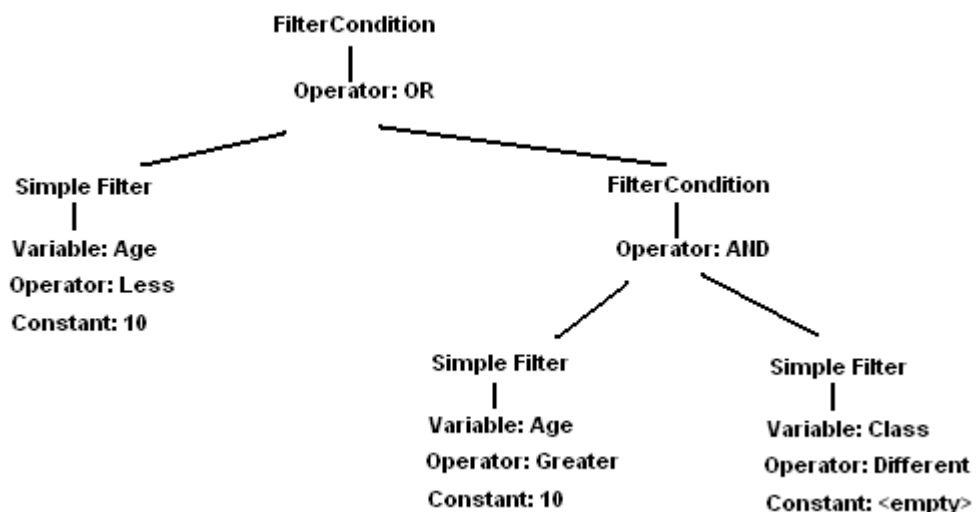
Globally, a filter is:

- A list of simple filters connected by logical operators (AND, OR)
- A list of filters connected by logical operators (AND, OR)

And a simple filter is just a variable, an operator and a constant.

A final filter is a list of simple filters connected by operators (AND, OR).

For example, the filter `Age<10 OR (Age>10 AND Class <> empty)` can be seen as the tree:



Describing a filter with SAP Predictive Analytics API

A filter definition is stored in the parameter tree of a space under `Parameters/FilterCondition`.

In the previous BNF, each bold word is a new parameter name or subtree name.

For example:

```
Age>10 AND Class <> <empty>
```

Is expressed in the parameter tree:

```
Parameters/FilterCondition/Operator          "And"
      /SimpleFilter1/Operator                "Greater"
      /SimpleFilter1/Variable                "Age"
      /SimpleFilter1/Value                    "10"
      /SimpleFilter2/Operator                "NotEqual"
      /SimpleFilter2/Variable                "Class"
      /SimpleFilter2/Value                    ""
```

And can be built by this KxShell script:

```
st.newSpace "adult01_500.csv" sp
sp.getParameter ""
sp.bindParameter "Parameters/FilterCondition" AndFilter
AndFilter.insert "SimpleFilter1" SimpleFilter
delete SimpleFilter
AndFilter.insert "SimpleFilter2" SimpleFilter
delete SimpleFilter
delete AndFilter
sp.validateParameter
sp.getParameter ""
sp.changeParameter "Parameters/FilterCondition/Operator" "And"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Operator" "Greater"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Variable" "age"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Value" "10"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter2/Operator" "NotEqual"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter2/Variable" "Class"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter2/Value" ""
sp.validateParameter
```

The more complex filter:

```
Age<10 OR ( Age>10 AND Class <> empty)
```

Is expressed as the following parameter tree:

```
Parameters/FilterCondition/Operator          "Or"
      /SimpleFilter1/Operator                "Less"
      /SimpleFilter1/Variable                "Age"
      /SimpleFilter1/Value                    "10"
      /FilterCondition2/Operator            "AND"
      /SimpleFilter1/Operator                "Greater"
      /SimpleFilter1/Variable                "Age"
      /SimpleFilter1/Value                    "10"
      /SimpleFilter2/Operator                "NotEqual"
      /SimpleFilter2/Variable                "Class"
      /SimpleFilter2/Value                    ""
```

This parameter tree can be built by the following script:

```
st.newSpace "adult01_500.csv" sp
sp.getParameter ""
sp.bindParameter "Parameters/FilterCondition" OrFilter
OrAndFilter.insert "SimpleFilter1" SimpleFilter
delete SimpleFilter
OrFilter.insert "FilterCondition2" FilterCondition2
FilterCondition2.insert "SimpleFilter1" SimpleFilterLevel2
delete SimpleFilterLevel2
FilterCondition2.insert "SimpleFilter2" SimpleFilterLevel2
delete SimpleFilterLevel2
delete FilterCondition2
delete OrFilter
sp.validateParameter
sp.getParameter ""
sp.changeParameter "Parameters/FilterCondition/Operator" "Or"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Operator" "Less"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Variable" "age"
sp.changeParameter "Parameters/FilterCondition/SimpleFilter1/Value" "10"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/Operator" "And"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter1/
Operator" "Greater"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter1/ /
Variable" "Age"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter1/
Value" "10"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter2/
Operator" "NotEqual"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter2/ /
Variable" "Class"
sp.changeParameter "Parameters/FilterCondition/FilterCondition2/SimpleFilter2/
Value" ""
sp.validateParameter
```

Operators have different names when stored in the parameter tree:

Operator	Parameter value
<	Less
<=	LessEqual
>	Greater
>=	Greater
=	Equal
<>	NotEqual

These names are case sensitive.

Special Case

An elementary filter like `age>10` must be expressed as AND operator with only one operand.

Example

```
Age>10
```

Is expressed as the simple parameter tree:

```
Parameters/FilterCondition/Operator           "And"  
      /SimpleFilter1/Operator                 "Greater"  
      /SimpleFilter1/Variable                 "Age"  
      /SimpleFilter1/Value                    "10"
```

6 Integrating Generated Code

6.1 What's New in Integrating Generated Codes

Links to information about the new features and documentation changes for Integrating Generated Codes.

SAP Predictive Analytics 3.3

What's New	Link to More Information
Codes for SAP HANA, Hive, Spark, and Vora are now supported.	<ul style="list-style-type: none">• About Code Generation [page 66]• Other SQL Codes [page 85]
MySQL, WX2 and Neoview are no longer supported.	<ul style="list-style-type: none">• About Code Generation [page 66]• Other SQL Codes [page 85]•
Sybase has been added to the list of available UDFs.	SQL UDF [page 86]
The table listing the differences of results has been updated.	Available Implementations of Code Generation [page 68]

6.2 About Code Generation

Code generation is a component that exports regression and segmentation models in different programming languages. The generated code enables you to apply models from outside of the application. It reproduces operations made by the application when encoding data and creating either classification, regression, clustering or recommendation models. All code types are not available depending on the model definition.

The code generation uses the following API call:

```
long generateCode(in string iType, in string iDirectory, in string iFileName);  
long generateCode2(in string iType, in string iDirectory, in string iFileName,  
in String iTargetName, in String iSpaceName, in String iKeyName);
```

The following table details how to use the variables used in the API call.

Variable	Description
iType	Key code of the generated language. For the list of available key codes, see section <i>Available Keycodes</i> .

Variable	Description
iDirectory	Directory where the code is generated
iFileName	Name of the output file
iTargetName	Selects the model target to generate the code
iSpaceName	Replaces the reference dataset used in the generated code
iKeyName	Replaces the reference key used in the generated code.

Available Key Codes

Key Code

AWK
C
CCL - on SAP HANA Smart Data Streaming databases
CPP
JAVA
JSON
PMML3.2
SAS

i Note

For SAS code, use the call `generateCode2` directly to set the application dataset and the key of the application dataset.

Codes with UDF Available

Key Code	UDF Key Code	Dedicated Database
HANA	HANAUDF	SAP HANA
ORACLE	OracleUDF	Oracle
SQLDB2	DB2UDF	IBM DB2
SQLNettezza	N/A	Nettezza
SQLServer	SQLServerUDF	SQLServer
SQLTeradata	TERAUDF	Teradata
SybaseIQ	SybaseIQUDF	Sybase IQ

i Note

For SQL code, use the call `generateCode2` directly to set the application dataset and the key of the application dataset.

Codes for use in HADOOP

Key Code	Dedicated Database
HIVE	Hive
SPARK	Spark
VORA	SAP HANA Vora

Full Documentation

Complete documentation can be found on the SAP Help Portal at <http://help.sap.com/pa>.

6.3 Available Implementations of Code Generation

Scores obtained by using generated codes should be the same as those obtained with the application. However, slight differences may exist, mainly due to precision issues in computation.

⚠ Caution

- Only C++ and Java codes can work with composite variables.
- All generated codes, except for PMM3.2 and AWK, can work with dateparts. A datepart is an automatic information extraction from an input date or datetime variable. Note that the code generator doesn't support date or datetime variables that are not split into dateparts. It means that if the final equation contains a date or datetime variable that is not split into dateparts, the application cannot generate an export of your model.

Possible Differences of Results Depending on the Code

Key Code	Classification/ Regression with a polynomial degree 1	Classification/ Regression with a polynomial degree greater than 1	Segmentation/ Clustering	Segmentation/Clustering with SQL Expression
AWK	!	!	!	!
C	++	++	++	++
CCL	++	!!	!!	++
CPP	++	++	++	++
DB2V9	++	!!	!!	++
DB2UDF	++	!!	!!	++
HANA	++	!!	!!	++
HANA UDF	++	!!	!!	++
Hive	++	!!	!!	++

Key Code	Classification/ Regression with a polynomial degree 1	Classification/ Regression with a polynomial degree greater than 1	Segmentation/ Clustering	Segmentation/Clustering with SQL Expression
JAVA	++	++	++	++
ORACLE	++	!!	!!	++
Oracle UDF	++	!!	!!	++
PMML 3.2	++	!!	++	++
PostgreSQL	++	!!	!!	++
SAS	++	++	++	++
Spark	++	!!	!!	++
SQLDB2	++	!!	!!	++
SQLNetezza	++	!!	!!	++
SQLServer	++	!!	!!	++
SQLServerUDF	++	!!	!!	++
SQLTeradata	++	!!	!!	++
SQLVertica	++	!!	!!	++
SybaseIQ	++	!!	!!	++
SybaseIQUDF	++	!!	!!	++
TERAUDF (UDF export in C)	++	++	!!	++
Vora	++	!!	!!	++

Caption

Symbol	Meaning
++	The syntax is correct and results are the same as the ones obtained with Automated Analytics engines ¹ .
!	The syntax is correct but the results are different ² .
!!	The code is not implemented.

i Note

¹ Results may be slightly different due to precision issues. Since each variable introduces a delta, the more variables the model contains, the more the results can differ.

² Database types without right trim (`RTrim`) consider as distinct two categories with names only differing by an ending whitespace.

6.4 Generated Codes

6.4.1 AWK Code

The generated AWK code allows applying a model on flat files with a simple script. There is: no need to compile, just run it through a single command line as follows:`awk -F"," -f myawkmodel.awk myinputdata > output.csv`

6.4.2 C Code

The generated C code is a standard ANSI code.

The entry point is a function with the following signature:

```
void    mymodel _apply( char* iValues[],
                        FILE* iOutFile,
                        unsigned long iLineIdx);
```

The following table details each parameter used by this function:

The parameter...	is...
<code>iValues</code>	an array of string (char*) containing the input values for the current record. These values must be in the exact same order as the training dataset (target and skipped values included, even empty)
<code>iOutFile</code>	an open file where the generated values are written
<code>iLineIdx</code>	the index of the current record
<code>mymodel</code>	the name of the current model

This function writes the model output in the argument file as follows:

```
iLineIdx, target_1, output_1, ..., target_n, output_n
```

where `n` is the number of targets. For Classification/Regression, the output is the classification score when dealing with binary target and the predicted value when dealing with continuous target, and for Clustering, the output is the cluster index.

i Note

When generating probability in classification mode for Classification/Regression, the output appears as shown below:

```
iLineIdx, target_1, output_1, proba_1, ..., target_n, output_n, proba_n
```

The sample program (`main.c`), provided with the product, allows reading and code generating a flat file. To be able to compile with the generated file, the sample program includes some `#define` flags.

Example

On Windows platform, using Microsoft C compiler, the following command should compile sources:

```
cl /o model.exe
-DKX_GENERATEDCFILENAME="\myModel.c\"
-DKX_MODELFUNCTIONNAME=mymodel_apply
-DKX_FIELDSEPARATOR=","
-DKX_NB_FIELDS=300 main.c
```

This command generates the `model.exe` file. To run this executable file to score a flat dataset, use the following syntax:

```
model.exe -in myInputData -out output.txt
```

where:

- `myinputdata` is the name of the dataset to score
- `output.txt` is the generated file containing scores

i Note

`myinputdata` must have its columns in the exact same order as the training dataset - target and skipped variables included.

6.4.3 CCL Code

Continuous Computation Language (CCL) is an event processing language of SAP HANA Smart Data Streaming (SDS). CCL is based on Structured Query Language (SQL), and adapted for stream processing. Generating CCL code allows embedding prediction code associated to a model in a Smart Data Streaming project.

The code generation module of Automated Analytics generates one or more output stream definitions that compute scores and other prediction values. Those stream definitions can then be included in a streaming project. Your Advanced Analytics model must be trained using a dataset containing a subset of your streaming events. This dataset must have variable names matching exactly the name of the input stream that will be used for scoring.

The following table details the parameters needed to generate CCL code.

Parameter Name	Description
<code>Key</code>	key field of the input stream/window
<code>Dataset</code>	name of the input stream/window to score

In some situations where additional information beyond the score is requested, the CCL code generator may produce more than one stream. In that case, cascading streams are produced to build the required prediction

information. The name of the final output stream is deduced from the name of the generated file by removing the extension, and additional output stream names are built by appending a meaningful suffix, such as `_bar` or `_proba`, for example.

Only the key information is added by default to the output stream along with the prediction information. If you want to keep all input fields and not only the key, you can set the global parameter named `CodeGenerator.CCL.AddAllInputStreamInResult` in the Automated Analytics configuration file.

Example

Add the following line to the `KJWizard.cfg` file or the `KxCORBA.cfg` file, depending on the component you have installed.

```
CodeGenerator.CCL.AddAllInputStreamInResult=true
```

6.4.4 C++ Code

6.4.4.1 C++ Code Generator Framework Presentation

The framework is close to the Java code generator framework as both frameworks use vectors and maps to manage models.

The C++ code generator framework is based on several interfaces and classes:

- `KxCppRTModel`: interface implemented by each generated model
- `KxCppRTModelManager`: manager holding a map of models and allowing to reach back models by their name.
- `KxCppRTCCase`: representation of one input record or output record, it is composed of `KxCppRTValue` elements
- `KxCppRTValue`: representation of one variable value

The C++ code generator produces a single file, the model class definition and the implementation.

Each file contains only a single class defining the apply function of a model. In addition, the name of the model is equal to the class name (this name is created by the application). Each generated model is registered in the model manager (this process is automatic, see `static` at the end of this example).

Here is a sample skeleton of such a generated file:

Example

```
Sample Code
```

```
...
```



```

    // definition of all categories
    ...
class cKxMyModel : public virtual KxCppRTModel
{
public :
    cKxMyModel();
    cKxMyModel(KxCppRTModelManager& iModelManager);
    // Others functions definition...
    ...
private:
    ... // Internal functions definition
    double Kxen_RobustRegression_10e4c501__0_KxVar1(KxCppRTCCase iInput) const;
    ...
};
cKxMyModel::cKxMyModel(KxCppRTModelManager& iModelManager)
{
    // register the model into the model manager
    iModelManager.registerModel("cKxMyModel", this);
    ...
}
void
cKxMyModel::apply(const KxCppRTCCase& iInput, KxCppRTCCase &oOutput) const
{
    Kxen_RobustRegression_10e4c501__0_KxVar1 (iInput, oOutput);
}
static CPPKxModel gKxMyModel(KxCppRtModelManager::instance());

```

6.4.4.2 C++ Framework Detail

This section details the behavior and usage of each class in the C++ code generator framework.

6.4.4.2.1 KxCppRTModel

Each generated model implements a generic interface called `KxCppRTModel`.

The `KxCppRTModel` interface has the following methods:

- `String getModelName()`: this method returns the name of the model as a string
- `apply(KxCppRTCCase, KxCppRTCCase)`: this method applies the model on a data row called a case. It needs input object providing input variable values and returns an object containing results value. Both input and output values are using an object to represent a set of values that will be described further.
- `StringVector getInputVariables()`: this method specifies the input variables the generated model needs. It returns a vector of needed input variable names.
- `StringVector getOutputVariables()`: this method specifies the output variables generated by the model. It returns a vector of output variable names.

```

class KxCppRTModel
{
public:
    virtual ~KxCppRTModel() {};
    virtual const KxSTL::string& getModelName() const = 0;
    virtual const KxSTL::vector<KxSTL::string>& getModelInputVariables() const =
0;

```

```

    virtual const KxSTL::vector<KxSTL::string>& getModelOutputVariables() const
    = 0;
    virtual void apply(const KxCppRTCCase& iInput, KxCppRTCCase& iOutput) const =
    0;
};

```

6.4.4.2.2 KxCppRTCCase

KxCppRTCCase interface allows feeding models with values. It provides services allowing the model to access values by rank (generated codes use the variable rank internally), and allows the external environment to set values using input variables names.

KxCppRTCCase interface is implemented by the integrator to connect its physical data values to the model class instances.

- `void setValue(String iVaribleName, KxCppRTValue value)`: this method is used by the calling program to fill the input case with proper values associated with variable names. This method is also used by the model to fill the proper output value.
- `KxCppRTValue getValue(Int)`: this method is used by the model to get the value associated by the input case provided by the integrator.
- `KxCppRTValue getValueFromName (String iVaribleName)`: this method can be used by the external program to get back the generated value from the model in the output case.

```

class KxCppRTCCase
{
    virtual ~KxCppRTCCase() {};
    virtual void setValue(KxSTL::string const& iName, KxCppRTValue const&
iValue) = 0;
    virtual const KxCppRTValue& getValue(int i) const = 0;
    virtual const KxCppRTValue& getValueFromName(KxSTL::string const& iName)
const = 0;
};

```

An example of a very simple implementation is provided (see `SampleMappedCase.cpp`)

6.4.4.2.3 KxCppRTModelManager

The framework uses a model manager providing model registering facilities. It associates each model with a name.

```

struct sPrivateData;
class KxCppRTModelManager
{
public:
    ~KxCppRTModelManager ();
    static KxCppRTModelManager& instance();
    void registerModel(KxSTL::string const& iModelName, KxCppRTModel* iModelPtr) {
mModelFactory[iModelName] = iModelPtr;
}
    static const KxCppRTModel& getKxModel (KxSTL::string const& iModelName) {
return instance().getModel(iModelName);
}
}

```

```

KxSTL::vector<KxSTL::string> getListModel() {
...
}
private:
    KxCppRTModelManager () {}
    const KxCppRTModel& getModel (KxSTL::string iModelName) {...}
    struct sPrivateData *mData;
};
struct sPrivateData {
    KxSTL::map< KxSTL::string, KxCppRTModel* > mModelFactory;
};

```

6.4.4.2.4 KxCppRTValue

The CPP Runtime uses KxCppRTValue as data.

```

struct sValueData;
class KxCppRTValue
{
public:
    KxCppRTValue(KxCppRTValue const& iOther);
    KxCppRTValue();
    KxCppRTValue(KxSTL::string const& iValue);
    KxCppRTValue(const char* iValue);
    ~KxCppRTValue();
    KxSTL::string const& getValue() const;
    KxCppRTValue& operator=(KxCppRTValue const& iOther);
private:
    struct sValueData* mValueData;
};
struct sValueData {
    KxSTL::string mValue;
    sValueData() {}
    sValueData(KxSTL::string const& iValue) : mValue(iValue) {}
    sValueData(const char* iValue) : mValue(iValue) {}
    sValueData(sValueData const& iOther) : mValue(iOther.mValue) {}
};

```

6.4.4.3 Using the Generated Models

The provided main sample program will create an instance of the generated model and feed it with the proper cases.

Sample Code

```

#include "StringUtilities.h"
#include "KxCppRTModelManager.h"
#include "SampleMappedCase.cpp"
int main( int argc, char ** argv )
{
    FILE* lInFile = NULL;
    FILE* lOutFile = stdout;
    ...
    lInFile = fopen(..., "r");
    lModelName = ...;

```

```

lOutFile    = fopen(..., "w");
...
// return model called CPPModel
const KxCppRTModel& lModel = KxCppRTModelManager::getKxModel(lmodelName);
// return the variable names used
KxSTL::vector<KxSTL::string> lInputNames = lModel.getModelInputVariables();
SampleMappedCase lInCase    = SampleMappedCase(lInputNames);
...
SampleMappedCase lOutCase =
    SampleMappedCase(lModel.getModelOutputVariables());
while (...)
{
...
    lInCase.setValue(...,
                    ...);
    // apply
    lModel.apply(lInCase, lOutCase);
...
}
fclose(lOutFile);
fclose(lInFile);
return 0;
}

```

Maintaining Generated Codes

In this sample implementation, generated codes are located in a stand-alone dynamic library (DLL). To add a new model runtime in this DLL, update `dll_x86-win32.mak` makefile. Replace the value of `MODEL_OBJECTS` target by the list of generated models.

6.4.5 HTML Code

The generated HTML code is an HTML page that can be viewed in any Javascript compliant Web browser. The user has to fill each variable value and click the target link at the bottom to access the model output.

Model	Transform
class_Census01	Kxen.RobustRegression

Variable	Value
age	37
capital-gain	608
capital-loss	89
education	KxOther
education-num	9
fnlwgt	190991
hours-per-week	39
marital-status	KxOther
native-country	KxOther
occupation	KxOther
race	KxOther
relationship	KxOther
sex	Male
workclass	KxOther

Target	Value
class	

6.4.6 Java Code

Java Code compilation needs `KxJRT.jar` in the classpath as shown below:

```
javac -classpath "path-to-KxJRT.jar" model.java
```

where `model.java` is the generated java code. This generates a file named `model.class` containing java bytecode.

i Note

KxJRT.jar file is provided in the Samples directory.

Then, to use the model, the `KxJRT.IKxJModelInputWithNames` interface must be implemented. This object is passed as an argument to the `IKxJModel.apply()` method and defines how to retrieve the input data.

This interface has the following methods that must be implemented:

- `java.lang.String[] getVariables()`
returns the variable names used in the data source the model is to be applied on.
- `boolean isEmpty(int iVarIdx, java.lang.String iMissingString)`
returns whether the current value of the variable with index `iVarIdx` in the array returned by `getVariables` is the empty value or not.

Other methods (`floatValue()`, `intValue()`, ..., `dayOfWeek()`) that convert a value into a correct data type are also available. These methods are described in the `IKxJModelInput` interface.

Example

Consider an object `DataProvider` able to provide variable values, the following code could be:

```
...
import KxJRT.*;
...
class KxJModelInput extends DataProvider implements IKxJModelInputWithNames {
...
    // IKxJModelInputWithNames interface

    String[] getVariables() {
        // use DataProvider to get the list of available variables
        ...
    }

    boolean isEmpty( int iVarIdx, java.lang.String iMissingString ) {
        boolean lIsMissing;
        // use DataProvider to test if current
        // value for variable iVarIdx is missing.
        // for example, perform a "ISNULL" when using a SQL source.
        String lStringValue;
        // use DataProvider to get, if possible,
        // the string representation of the current value.
        // when dealing with SQL source, it is not
        // available for continuous value.
        if( lIsMissing || lStringValue.equals( iMissingString ) ) {
            return true;
        }
        else return false;
    }

    float floatValue( int iVarIdx ) {
        // if current value is continuous, then return its value, else return a
        // default value say 0 .
        ...
    }
    ... And so on ...
}
```

The previous class reads a flat file and converts string values into the desired type. To have a concrete implementation of this class, please refer to `KxJRT.KxFileReaderWithNames` for an example code.

The final skeleton code should contain the following lines to execute an exported model in JAVA code:

```
...
import KxJRT.*;
...
    // String containing the model class name. For example,
    // if model.java was generated, then lModelName is "model" .
    java.lang.String lModelName;
    ...
    // ask the factory to instantiate the class.
    IKxJModel mKxJModel = KxJModelFactory .getKxJModel( lModelName );
    ...
    // instanciate IKxJInputModelWithNames
    KxJModelInput lInput = new KxJModelInput( ... );
    ...
    // instanciate KxJRT.KxJModelInputMapper (see KxJRT.Mapper class )
    KxJModelInputMapper    mMapper = new KxJModelInputMapper ( lInput,
mKxJModel );
    ...
    // instanciate a data source (it may be already done via lInput)
    DataProvider mDataProvider = new DataProvider( ... );
    ...
    // main loop that reads each data row and applies the model on it
    while( mDataProvider.hasMoreRows() ) {
        Object[] lResults = mKxJModel. apply( mMapper );
        // store or print the results somewhere.
        // here print results on standard output
        for( int i=0; i<lResults.length; i++ ) {
            System.out.print( lResults[i].toString() );
            if( i+1<lResults.length ) System.out.print(",");
        }
        System.out.println();
    }
    ...
```

The above code should be compiled with `KxJRT.jar` in it, and executed with both `KxJRT.jar` and the directory containing `model.java` in it.

i Note

See `KxJRT.KxJApplyOnFile` for sample code.

`KxJRT.jar` contains a sample program to read and score flat files.

To apply a Java model on a flat file, use the syntax as below:

```
"Usage: [-nonames] [-separator <sep>] [-out <file>] -model <model> -in <file>
```

The parameter...

`-nonames`

indicates...

do not look for variables names on first line

i Note

Setting `nonames` implies the input file has the same structure as dataset used for training.

The parameter...	indicates...
-separator	set field separator (default is a comma (,))
-out	set output file (default is standard output)
-model	set Java model
-in	set input file to score

Example

To apply the model `SampleModel.java` on `SampleDataset.csv` (that is a comma separated values file) and store results in `results.csv`: `javac -classpath KxJRT.jar SampleModel.java java -jar KxJRT.jar -model mymodel -in SampleDataset.csv -out results.csv"`.

The help is obtained by typing this command: `java -jar KxJRT.jar -usage`.

6.4.7 PMML Code (3.2 version)

Predictive Model Markup Language (PMML) is an XML markup language used to describe statistical and data mining models. It is edited by the Data Mining Group (<http://www.dmg.org/>). PMML is supported by products listed on this Web page: <http://www.dmg.org/products.html>.

6.4.7.1 Code Generation Using PMML and DB2 IM Scoring

This section explains how to generate scores in a database with PMML and DB2 IM Scoring V7.1 from a model.

Some prerequisites are needed to use PMML with the application:

- A database (DB2 or Oracle)
 - The IBM product DB2 IM Scoring V7.1 that should be installed on this database
1. Generate the PMML code corresponding to the model.
 2. Insert the PMML model in the database with an SQL insert statement.
 3. Apply the model on a table with an SQL query.

6.4.7.1.1 DB2 IM Scoring for DB2 and Oracle

DB2 IM Scoring is available for two databases: DB2 and Oracle.

6.4.7.1.1 Installing DB2

The following DB2 versions are required to work with IM Scoring:

- Fixpack 5 if you are working with DB2 UDB V6
- Fixpack 1 if you are working with DB2 UDB V7

During the DB2 installation, use the same logon and password for the DB2 instance user as for your Windows account. Otherwise, the scripts will be unable to automatically stop and start your DB2 instance when required.

→ Tip

To launch a DB2 command, use `C:\>db2cmd` DOS command.

To launch `script.sql` SQL script in the DB2 environment (in a `DB2 CLP` DOS window), use `C:\>db2 -stf script.sql` DB2 command.

6.4.7.1.2 Installing Oracle

The Oracle8i 8.1.7 database version is required to work with DB2 IM Scoring.

→ Tip

To launch an Oracle command, use `C:\>sqlplus user/password@connectionstring` DOS command.

To launch `script.sql` SQL script in the Oracle environment, use `SQL> @script.sql` Oracle command.

6.4.7.1.2 DB2 IM Scoring V7.1 for DB2 and Oracle

⚠ Caution

Before installing DB2 IM Scoring, create a specific user (into DB2 or Oracle) to store all IM Scoring tables.

It is also recommended to install the IM Scoring V7.1 Fixpack 1.

Download Program Temporary Fixes (PTFs) from IBM Web site: <http://www-3.ibm.com/software/data/iminer/scoring/downloads.html> .

6.4.7.1.2.1 Installing IMScoring DB2 UDF on an Existing Database

1. Launch the DB2 command in IM Scoring bin directory: `idmEnableDB.bat DatabaseName fenced`
2. Set the database parameters to increase memory management:
 - a. Launch the following DOS script:

```
rem ---increase size of UDF_MEM_SZ
db2 update dbm cfg using udf_mem_sz 60000
db2set DB2NTMEMSIZE=APLD:60000000
rem --- increase size of APPLHEAPSZ
db2 update db cfg for YourDatabaseName using APPLHEAPSZ 8192
db2stop
db2start
rem --- start the DB2 JDBC Applet Server for port 6789
db2jstrt 6789
```

- b. Be sure that your DB2 instance has been restarted by checking the DB2 instance `UDF_MEM_SIZE` property. This parameter must be set to 60000.

6.4.7.1.2.2 Installing IMScoring Oracle Packages on the Default Database

The Default Database is identified by `ORACLE_SID` parameter.

1. Use `idm_setup.bat` DOS script with the SYS Oracle user (default password: `change_on_install`).
2. In `samples\Oracle IM Scoring` directory, use `idm_create_demotab.sql` Oracle SQL script to create tables containing PMML models.

6.4.7.2 Inserting PMML in the Database

Once the PMML file has been generated, you have to load it into the database.

This insert is made with an SQL query and with an IM Scoring function.

In IM Scoring, a regression PMML model is inserted into `REGRESSIONMODELS` table with `DM_impRegFileE` function.

`REGRESSIONMODELS` table contains two columns:

- `MODELNAME` (VARCHAR(240)): name of the model
- `MODEL` (IDMMX.DM_REGRESSIONMODEL): PMML model in DB2 format. IDMMX is the DB2 schema installed and used by DB2 IM Scoring.

DB2 IM Scoring create its own data types and functions (User Defined Function) on this schema.

`DM_impRegFileE` function from the IDMMX schema is used to transform a PMML file into `DM_REGRESSIONMODEL` DB2 IM Scoring data type.


```
'sex', t.SEX),
'capital-gain', t.CAPITALGAIN),
'capital-loss', t.CAPITALLOSS),
'hours-per-week', t.HOURLPERWEEK),
'native-country', t.NATIVECOUNTRY))
)
from IDMMX.ReggressionModels r, TABLE_TO_SCORE t
where r.modelname='KXENmodel';
```

6.4.8 SAS Code

SAS code is a script that can be interpreted by the SAS system.

The following first two lines must be added:

```
data in_dataset ;
set out_dataset;
```

The following table details the parameters used:

The parameter...	Is...
in_dataset	the SAS dataset to score
out_dataset	the generated dataset with scores

i Note

The parameter `&Key` must be replaced by the ID of the dataset to have an ID with a score.

6.4.9 SQL Code

The SQL code is database-dependent and some context-dependent variables have to be set in the SQL query before applying:

The variable...	Has to be replaced by...
<code><\$KEY></code>	the key of the apply dataset
<code><\$DATASET></code>	the table or view to score

i Note

In an automatic process, the search-and-replace job can be done with a PERL or AWK script. A special SQL for MySQL database has been released because of the symbol surrounding variable names (" instead of standard ").

6.4.9.1 SQL ANSI

The generated SQL ANSI code is compliant and should work on most databases which are not yet supported by the application.

6.4.9.2 SQL Generation Options

SAP Automated Analytics allows users to set the following parameters for code generation:

`separator [SQL]`: allows customizing the SQL separator between two SQL queries. By default, this parameter is set to `GO`.

6.4.9.3 Other SQL Codes

Key Code	UDF Key Code	Dedicated database
HANA	N/A	SAP HANA
Hive	N/A	?
ORACLE	OracleUDF	Oracle databases
Spark	N/A	?
SQLServer	SQLServerUDF	SQLServer databases
SQLDB2	DB2UDF	IBM DB2 databases
SQLTeradata	TERAUDF	Teradata databases
SQLNettezza	N/A	Nettezza databases
SQLVertica	N/A	Vertica databases
SybaseIQ	SybaseIQUDF	Sybase IQ databases
SYBASEASE	N/A	Sybase ASE databases
VORA	N/A	?

6.4.10 UDF Code

6.4.10.1 SQL UDF

The generated SQL code creates a SQL User Defined Function (UDF) computing a score from model parameters.

The syntax for creating a UDF and describing its parameters depends on the DBMS. The SQL UDF code generators for the following DBMS are available:

- DB2
- Oracle
- SAP HANA
- SQLServer 2000
- Sybase

6.4.10.1.1 Installing the SQL UDF

The file generated by the code generator contains all necessary SQL instructions to install the UDF.

You must use the standard SQL front-end of the DBMS to execute the generated file.

The following table details the standard SQL front-end for each DBMS:

DBMS	SQL Front-End
SAP HANA	SAP HANA Studio SAP HANA web-based development workbench SAP HANA Web IDE
SQLServer 2000	Query Analyser
Oracle	SQLPlus
DB2	Command Center

⚠ Caution

Since the generated file can contain the instruction to drop the UDF before re-creating it, it is normal for the SQL front-end to signal an error the first time when the UDF does not exist yet.

Except the UDF header, the generated SQL code does not use special features of the DBMS or ANSI mode setup.

The current user must have the actual rights to drop/create a UDF. Check with your DBA.

6.4.10.1.2 Using the SQL UDF

When installed, the UDF extends SQL exactly as a standard SQL function.

A typical use is (supposing UDF is named `ClassPredictedByKXAF`):

```
SELECT
ClassPredictedByKXAF (age, workclass, fnlwgt, education, educationnum, maritalstatus, occu-
pation, relationship, race, sex, capitalgain, capitalloss, hoursperweek, nativecountry)
FROM Adult
```

As a convenience, all generated UDF code files include a comment with a typical usage of the current UDF.

6.4.10.1.3 SQL UDF Generation Options

The application allows users to set the following parameters for code generation:

`SmartHeaderDeclaration [UDF]`: this parameter allows excluding from the generated code all the non-contributive variables (variables with a contribution of 0). The default value of the parameter is set to `true` for the application to generate an UDF declaration with only useful variables. In some cases, it can significantly reduce the size of the generated code. Changing the value of this parameter has no effect on the final results.

6.4.10.1.4 DBMS Dependent Options on Generated SQL UDF

Each DBMS has different options for fine tuning the UDF.

Here is the set of options the application uses:

- **ORACLE**
 - `PARALLEL_ENABLE`: self explained.
 - `DETERMINISTIC`: SQL code does not use or change external values. Each UDF call with the same actual parameters gives the same result. It allows Oracle to cache previous calls result.
- **DB2**
 - `DETERMINISTIC`: SQL code does not use or change external values. Each UDF call with the same actual parameters gives the same result. It allows Oracle to cache previous calls result.
 - `NO EXTERNAL ACTION`: no external DBMS resource (file, lock, and so on) is used or changed
 - `CONTAINS SQL`: indicates that the code does not read or modify other SQL data.
- **SybaseIQ**
 - `DETERMINISTIC`: SQL code does not use or change external values. Each UDF call with the same actual parameters gives the same result. It allows Oracle to cache previous calls result.
- **Other UDFs**
 - No specific option is used.

6.4.10.2 Teradata C UDF

The application generates specific Teradata User Defined Function. The scoring code is written in C (with Teradata coding conventions) and the UDF itself is described in SQL.

⚠ Caution

Teradata C UDF is only available on Teradata V2R5.1.

6.4.10.2.1 Installing the Teradata UDF

Considering a model with n targets, the code generator generates $n+2$ files:

- The Teradata C code: the file name is given by the user with the extension `.teraudf`. This file contains all the C code needed to build the UDF for all targets.
- The SQL creation and description of the UDF: the file name is the same as the one given to the C code file with a `.bteq` extension. This file contains all SQL wrappers needed to describe the UDF from a SQL perspective.
- One C file per target (that is n files) which name is `<Name><TargetName>.c`, where `<Name>` is the name given by the user. These files are only created for technical reasons linked to the limitations of the Teradata compilation environment.

Example

If the user has chosen the name `MyUDF` and a model with two targets, class and sex, four files will be generated:

- `MyUDF.teraudf`: the C code
- `MyUDF.bteq`: the SQL installation and description code of two UDF named `MyUDFclass` and `MyUDFsex`
- `MyUDFclass.c`
- `MyUDFsex.c`

You must use *Teradata BTEQ tool* to execute the `.bteq` file. In the *BTEQ tool*, enter:

- `.login <node>/<login>`
- `.run file=<UDF Name>.bteq`

This will automatically do the whole task:

- transfer the C code
- compile the C code
- create the UDFs

⚠ Caution

Since the generated file contains the instruction to drop the UDF before re-creating it, it is normal bteq signals an error the first time when the UDF does not exist yet.

The current user must have the actual rights to drop/create a UDF. Check with your DBA.

6.4.10.2.2 Using the Teradata UDF

When installed, the UDF extends SQL exactly as a standard SQL function.

Supposing UDF is named `ClassPredictedByKXAF`, a typical use is:

```
SELECT
ClassKXAFPredicted (age, workclass, fnlwgt, education, educationnum, maritalstatus, occupa
tion, relationship, race, sex, capitalgain, capitalloss, hoursperweek, nativecountry) FROM
Adult
```

As a convenience, the `.bteq` file includes a comment with a typical use of the UDF.

6.4.10.2.3 Obtaining help on the Teradata UDF

The code generator generates a Teradata comment associated to the UDF.

This comment contains useful information, for example the application version and the generation date.

To see this comment in any Teradata request tool, type: `Comment on function <UDF Name>`

You can also display a full description of the UDF and its parameters by typing: `Show function <UDF Name>` in any Teradata request tool.

6.4.10.3 Frequent Problems when Using UDF

Since parameters types (`INTEGER`, `FLOAT`, and so on) are explicitly described in the UDF, you must call the function with correct types parameters. If actual parameters types do not match, the DBMS displays this error: `<UDF name> not found`. To solve this issue, use standard SQL CAST operators.

For the same reason, passing an explicit `NULL` value as an actual parameter also needs a CAST operator. Because `NULL` is a special SQL value with no type, the parameter type will never match.

Example

If the first parameter of the UDF is described as a `FLOAT`, `TheUDFName (NULL, ...)` call must be replaced by `TheUDFName (CAST (NULL AS FLOAT), ...)`

6.4.10.4 Performances of Automated Analytics SQL and UDF

This section details results of a performance test on model scoring.

The test consisted in applying a model on an existing dataset (update mode) in three different ways:

- Using the Automated Analytics engine and an ODBC connection
- Using a SQL generated code
- Using a generated UDF

DBMS	Automated Analytics /UDF ratio	Automated Analytics /SQL ratio	UDF/SQL ratio
DB2	4.7	5.5	1.2
Oracle	4.3	5.0	1.2
SQLServer 2000	4.7	5.6	1.2
Teradata v2R5.1	Cannot be tested	Cannot be tested	0.5

6.4.11 VB Code

The VB code is generated in standard VB 7 code.

The entry point is a function with the following signature:

```
Public Function ApplyModel( iInputs() As Variant ) As Variant()
```

The following table details parameters used in the function above:

The parameter...	Is...
<code>iInputs</code>	an array of variants containing the input values for the current record

These variants must be in the exact order specified by the following function generated at the beginning of the script:

```
Public Function GetModelInputs() As String()
```

It returns an array of input names.

`ApplyModel` function returns an array of outputs specified by the following function:

```
Public Function GetModelOutputs() As String()
```

7 Model Parameter Reference

This section presents the model parameters as they are displayed in the parameter tree.

☰ Syntax

Path: <Model_Name>

The root of the path is the name of the model.

In SAP Predictive Analytics, a model is more than an algorithm. It contains all the information needed to process the data in the original format they are stored in the source database up to the modeling results. This is why a model is described by the data sets it has been trained on and by the protocol (which is a chain of transforms) it uses. All this information is saved in a tabular form when a model is saved.

The models are adaptive models: they must be generated, or trained (the terms "estimated" or "adapted" are also used), on a training data set before they can be used, or applied, on new data sets. As all objects defined in SAP Predictive Analytics architecture, a model is described by parameters, which are described in the following sections.

7.1 Model Generation Parameters

☰ Syntax

Path: Parameters

This folder contains the parameters related to the model generation.

Parameter	Description	Values
CutTrainingPolicy	Indicates the way a training dataset is cut into three subsets when needed (estimation, validation and test). The impact of each of these strategies can be finely tuned through the parameters of the training dataset.	<ul style="list-style-type: none"> random with no test (default except for Time Series) sequential with no test (default for Time Series) random periodic sequential periodic with test at end random with test at end periodic with no test
CodingStrategy	Version of the model strategy used to build the model	By default, it is the number of the SAP Predictive Analytics version used to generate the model.
State	<p>Indicates the state of the model.</p> <p>As all SAP Predictive Analytics objects, a model is described by a state diagram. States are used internally to know what are the possible actions that can be asked on these objects. This is an internal parameter and the user cannot change the state of a model. Most of the time, when a model has been trained, it will be saved in the "ready" state. When a model is saved before training, it can be saved with a "checked" state and used as a template.</p>	<ul style="list-style-type: none"> created (default) ready checked running

☰ Syntax

Path: Parameters/AutoSave

This folder contains the necessary information to automatically save the model at the end of the learning phase.

Parameter	Description	Values
AutosaveEnabled	Boolean value that indicates whether the model must be automatically saved at the end of the learning phase	<ul style="list-style-type: none"> False (default): the model will not be saved True: the model will be automatically saved
AutosaveStoreIdx	Integer that represents the return value of the openStore function used when saving the model to indicate the store location	<ul style="list-style-type: none"> 0 (default) Any unsigned integer

Parameter	Description	Values
AutosaveModelComment	Comment associated to the model	User-defined character string
AutosaveModelSpace	Space where the model is saved	Name of a file or of a database table

≡ Syntax

Path: Parameters/CodeGeneration

This folder contains the parameters used to generate the code corresponding to the model.

Parameter	Description	Values
NbLineForScoreCard	Number of colors used to display the result lines in the score card	<ul style="list-style-type: none"> 2 (default) any integer
UTF8	Allows you to generate a report in UTF-8 format	<ul style="list-style-type: none"> false (default) true
Separator	Allows you to customize the SQL separator between two SQL queries	<ul style="list-style-type: none"> GO (default) Any user-defined string
SmartHeaderDeclaration	Allows you to exclude from the generated code all the non-contributive variables (variables with a contribution of 0). In some cases, this can significantly reduce the size of the generated code. Changing the value of this parameter has no effect on the final results.	<ul style="list-style-type: none"> true (default): all non-contributive variables are excluded from the generated code false: all variables are included in the generated code

7.2 Infos

≡ Syntax

Path: Infos

Additional information is stored in this folder after the model has been generated, that is, when the Parameters/State value is ready. All these parameters are read-only.

Parameter	Description	Values
Author	Name of the user who has generated the model	By default it is the name of the user logged on the machine
LearnTime	Duration of the model generation (in seconds)	Positive integer

Parameter	Description	Values
ApplyTime	Duration of the latest application of the model (in seconds) This duration is updated each time a model receives one of the following commands using the <code>sendMode</code> call: <ul style="list-style-type: none"> • <code>Kxen_apply</code> • <code>Kxen_filterOutlier</code> • <code>Kxen_test</code> 	<ul style="list-style-type: none"> • 0 (default): this value is used when the model has never been applied • Positive integer
BuildDate	GMT time at the end of the generation process, in the format <code>YYYY-MM-DD HH:MM:SS</code>	<ul style="list-style-type: none"> • blank (default) • Any date
KxenVersion	Version of SAP Predictive Analytics used to generate the model	Any SAP Predictive Analytics version number (for example 2.1.0)
ModelVersion	Version of the model built by SAP Predictive Analytics When saving a model you have two options: overwriting the existing model with the current one, or creating a new version of the model, in which case the previous model will still be accessible.	<ul style="list-style-type: none"> • 1 (default) • Positive integer
Model32Bits	Boolean value that indicates if the model has been generated on a 32-bit architecture	<ul style="list-style-type: none"> • true • false
BuildData	Name of the training dataset	Name of the file or database table that was used to train the model
ModelName	Logical name of the model either automatically generated (default) or provided by the user	<ul style="list-style-type: none"> • <code><target_name>_<dataset_name></code> (default, except for Social and Association Rules) • <code><dataset_name>_<graph_type></code> (default for Social) • <code><item_column_name>_<dataset_name></code> (default for Association Rules) • User-defined character string
FilterConditionString	Definition of the filter applied on the training dataset, if there is one	<ul style="list-style-type: none"> • blank (default) • Logical expression

Parameter	Description	Values
LastApplyStatus	The status of the last application (or test) task on the model It is empty if the model has never been applied.	<ul style="list-style-type: none"> • Success: the task has been successfully completed with no warning. • Failure: errors have prevented the task from completing successfully. • SuccessWithWarnings: the task has been successfully executed but some warnings have been encountered (for example conversion problems). • Aborted: the user has canceled the process.

⌘ Syntax

Path: Infos/ClassName

This parameter allows you to assign a class to the model. This will make it easier to sort your models and find them (for example when using Model Manager). You can use the project name or the type of campaign (churn, up-sale, ...).

Parameter	Description	Values
Default	If the value of this parameter is not set by the user, the value set in the <code>Default</code> subparameter is used. The <code>Default</code> parameter is automatically filled by the system and depends on the type of model.	<p>Any user-defined string</p> <p>System values:</p> <ul style="list-style-type: none"> • <code>Kxen.Classification</code> (classification model - nominal target) • <code>Kxen.Regression</code> (regression model - continuous target) • <code>Kxen.Segmentation</code> (segmentation model - SQL Mode) • <code>Kxen.Clustering</code> (clustering model - no SQL Mode) • <code>Kxen.TimeSeries</code> (time series model) • <code>Kxen.AssociationRules</code> (association rules model) • <code>Kxen.Social</code> (social networks analysis model) • <code>Kxen.Recommendation</code> (recommendations model) • <code>Kxen.SimpleModel</code> (multi-target models, any other model)

7.3 Protocols

⌘ Syntax

Path: `Protocols`

A protocol is a stack of transforms applied on the data as it is stored in the databases or files.

Each transform in this stack generates information of higher and higher abstraction level, and more and more related to business questions. A good way of picturing a protocol is to remember communication protocol stacks that are in charge of transporting information from one point to the other using more and more complex structures. A protocol is referred to in the model through its name ('Default' for SAP Predictive Analytics).

A protocol contains information about the chain of transforms processing the data, and about all variables used or produced by these transforms. Furthermore, the high level strategies driving the transforms behaviors are also defined in the protocol. '`Kxen.SimpleModel`' uses only one protocol (whose default name is 'Default'), but, as an example, multi-class models will use several protocol (each protocol dedicated to a two-classes problem).

7.3.1 Variables

☰ Syntax

Path: Protocols/Default/Variables

This folder contains all information on variables used in the modeling process. This does not only mean the variables that are stored into the spaces (files or DBMS tables), but also variables that are created by data preparation transforms within protocols.

Variables are defined by their name, some high level descriptions, and their role with respect to the protocol. They contain information about statistics on each useful dataset. Variables can be accessed either through the protocol objects or from the datasets objects.

7.3.1.1 Variable Parameters

☰ Syntax

Path: Protocols/Default/Variables/<Variable_Name>

The parameter tree devoted to the variable can be very large, because it contains information about the statistics collected on this variable. Parameters are under several groups for clarity of the presentation.

Basic Description

Parameter	Access	Value Type	Description
Value	Read-only when the model is in ready state	<ul style="list-style-type: none">• Nominal• Ordinal• Continuous• Textual• Composite	
Storage	Read-only when the model is in ready state	<ul style="list-style-type: none">• Integer• Composite• Angle• String• Date• Datetime	
KeyLevel	Read-only when the model is in ready state	Integer	A number different than 0 indicates that this variable can be considered as a part of the identifier of each line.

Parameter	Access	Value Type	Description
OrderLevel	Read-only when the model is in ready state	integer	A number different from 0 indicates that this variable can be used to sort the cases in a natural order.
MissingString	Read-only when the model is in ready state	String	When a value equal to the string specified here is found in the input space, the variable is considered as missing. This allows coping when specific codes are used to represent missing values (such as 9999 for a four digit unknown year for example).
Group	Read-only when the model is in ready state	String	Used to identify the group of the variable. This notion can be used to optimize internal computation. Information is not searched in crossing variables that belongs to the same group. For example, when a color information is already encoded into three disjunctive columns in a dataset (one column for green, the second for red, the third for blue), if an event is green, it is useless to search for information in object that could be both green and blue.
Description	Read-write	String	Comment

Advanced Description

The second group collects information about some elements that can be refined by the advanced user.

Parameter	Access	Value Type	Description
MinForRange	Read-write		Only used for ordinal or continuous variables. The user can increase this value before the apply process, but in this case, the expectation on the level of performance of the global system cannot be guaranteed. It also has a sub parameter named <code>Default</code> [<i>always read-only</i>], which is filled with default values by the statistics computation (minimum value found on the Estimation data set). The default value is used if no value is specified (or "").

Parameter	Access	Value Type	Description
MaxForRange	Read-write		<p>Only used for ordinal or continuous variables.</p> <p>The user can decrease this value before the apply process, but in this case, the expectation on the level of performance of the global system cannot be guaranteed. It also has a sub parameter named <code>Default</code> [<i>always read-only</i>], which is filled with default values by the statistics computation (maximum value found on the Estimation data set). The default value is used if no value is specified (or "").</p>
ConstantForMissing	Read-write		<p>Indicates the preferred value to be used to replace missing values from the data.</p> <p>It also has a subparameter <code>Default</code> [<i>always read-only</i>] which is filled with default value corresponding to the most frequent category of the Estimation data set with nominal or ordinal variables, and with the average computed on the Estimation data set for continuous variables. The default value is used if no value is specified (or "").</p>
IsVirtualKey	Read-only	Boolean	<p>Is <code>true</code> if the variable is an index automatically generated by SAP Predictive Analytics (typically named <code>KxIndex</code>). This index is generated only when there is no declared key in the current data set.</p>
UserPriority	Read-only when the model is in ready state	Number	<p>This parameter can be used by the components in internal computations.</p> <p>If a component needs to make variable selection (either in to discard some variables in internal processes or to present variables to the user in a specific order), the priority can be used in conjunction with other internal measures: for variables with the same internal importance, the variable with the lower priority will be selected. The user can use this variable in order to enforce usage of variable that are "cheap" to collect versus others (in that case, a high number in the priority parameter must be set for expensive variables). The default priority is 5.</p>
UserEnableCompress	Read-only when the model is in ready state	Boolean	<p>When set to <code>false</code> allows the user to deactivate the target based optimal grouping performed by K2C on this single variable.</p>

Parameter	Access	Value Type	Description
UseNaturalEncoding	Read-only when the model is in ready state	Boolean	The Natural Encoding Mode has been added for the Variables. In this mode, only the original version of the variable is used by SAP Predictive Analytics. Encoded versions are disabled and exclusion criteria are relaxed for original versions.
UserBandCount	Read-only when the model is in ready state	Number	Only for continuous variables. Allows the user to force a number of bands (segments, bins) to collect statistics on this variable.
UserEnableKxOther	Read-only when the model is in ready state	Boolean	Only for nominal variables. When set to <code>false</code> allows the user to deactivate the compression into <code>KxOther</code> for very infrequent categories.
			<div style="border: 1px solid #ccc; background-color: #f9f9f9; padding: 5px;"> <p>⚠ Caution This will generally lead to non stable data representation and coding.</p> </div>
NominalGroups, OrdinalBands or ContinuousBands		Subdirectories	Can be inserted by the user to force a data structure.
UserModulus	Read-only when the model is in ready state		Allows the user to enforce the bands of the continuous variables to be modulus of the given value. For example this allows the user to enforce the fact that bands are always multiple of 1000 when dealing with monetary values.
SpaceName	Read-write	String	Secondary name of the variable that can be used when putting back results within an output data set. The user can change the actual column in which the values are stored back by changing the <code>SpaceName</code> of the variable. This allows making the distinction between the name of the variable presented to the user and the name of the technical column to write in. If not explicitly changed, the technical column name to write back is the name of the variable.
InSpaceName	Read-write	String	Another name of the variable that can be used when getting results from an input data set. The user can change the actual column in which the values are read by changing the <code>InSpaceName</code> of the variable. If not explicitly changed, the technical column name to read from is the name of the variable.

Statistics

The final group is the entry point to get information about variable statistics on the different datasets.

Parameter	Access	Value Type	Description
Monotonicity	Read-write	Boolean	<p>Indicates that the variable is monotonic with respect to the line number.</p> <p>This can be bad news if the cutting strategy is not consistent with it. When a useful variable is monotonic, the system will enforce in next releases the cutting strategy to be sequential.</p>
StatForCode	Read-only	String	<p>Name of the data set used to encode the data.</p> <p>Most of the time, the statistics used to encode the data is "Estimation" for the input variables and "Validation" for the generated variables.</p>
EstimatorOf	Read-only		<p>Used internally to remember that a variable has been created as an estimator of another one belonging to the protocol.</p>
TargetKey	Read-write		<p>Only for nominal target variable.</p> <p>It contains the variable category specified by the user as the target category. By default, the target category is the least frequent one.</p>
Translations	Read-write	Folder	<p>A folder is created for each variable, it contains one entry per language, and for each language, an entry per variable category. This allows the user interfaces to translate the identifiers of the category with their proper string representation (that could depend on the language of the client interface). If no translations have been defined, the folder is empty. If one or more languages have been defined, they appear as sub-parameters.</p> <ul style="list-style-type: none"> • <code>all</code>: contains the default translations. If the translations for the language used by the interface do not exist, the translations stored in <code>all</code> are used. If there are no translations in <code>all</code>, the value found in the data set is used. • <code><ISO_language code></code>: contains the translations for the language corresponding to the give ISO language code (see the list of ISO codes).

Parameter	Access	Value Type	Description
SpaceOrigin	Read-only	Boolean	<p>true if the variable has been found in the input space (training for example), and false otherwise.</p> <p>For example, for all variables generated by SAP Predictive Analytics Explorer - Event Logging, SAP Predictive Analytics Explorer - Sequence Coding, SAP Predictive Analytics Explorer - Text Coding and score or segment variables, SpaceOrigin is set to false.</p>
Statistics	Read-write	Folder	Folder in which the statistics on all data sets are stored.
Role	Read-only when the model is in ready state	<ul style="list-style-type: none"> • skip • input • weight • target 	<div style="border: 1px solid #ccc; background-color: #f9f9f9; padding: 5px;"> <p>⚠ Caution</p> <p>This parameter does not exist for internal variables.</p> </div>

7.3.1.2 Variable Statistics

≡ Syntax

Path: `Protocols/Default/Variables/<Variable_Name>/Statistics/<Dataset_Name>`

Statistics are collected for each dataset. Each dataset is referenced through its name. In this folder, you find as many directories as there are valid datasets defined for the model. None of these computed elements can be changed by the user.

Parameter	Description
NbKnown	Number of observations where this variable is known on the dataset
NbUnknown	Number of observations where this variable is not known on the dataset
NbOutOfRange	Number of observations where the variable is out of range (outside the data dictionary builds for nominal and ordinal variables, and outside the [min, max] range for ordinal continuous variables)
NbTotal	Total number of observations. It is equal to the sum of NbKnown, NbUnknown and NbOutOfRange.
IsMerged	Used to reload some models that were saved with a lack of precision.
ProbaDeviation	Probability that there is a significant deviation in the distribution of categories (or bands in the case of continuous variables) between this data set and the data set jointed by the parameter StatForCode. This probability is computed through a CHI-square test.

Parameter	Description
Min	Only for number or date variables Minimum value on the dataset.
Max	Only for number or date variables Maximum value on the dataset
Mean	Only for number or date variables Mean value on the dataset
Variance	Only for number or date variables Variance on the dataset
StandardDeviation	Only for number or date variables Standard deviation value on the dataset
Targets	The new architecture of components allows for multiple targets. This folder is prepared to hold information specific to each target, this comprises the groups of the original categories and the computed values such as KI and KR. See Targets [page 103] .
Categories	Sub tree collecting statistics of each category of nominal or ordinal variables. It also correspond to some binning of continuous variables when this is required by the following transforms. See Categories [page 104] .
UniformCurvePoints	Only for continuous variables It corresponds to the points allowing to draw the cumulative distribution curve. Each point is given through [X, Y] coordinates. In regular situations, it should be S-shaped.

7.3.1.2.1 Targets

≡ Syntax

```
Path: Protocols/Default/Variables/<Variable_Name>/Statistics/<Dataset_Name>/
Targets/<Target_Name>
```

This folder holds information that is specific to each possible target. You find a subfolder for each target variable.

Parameter	Description
Ki	KI (Predictive Power) of the variable with respect to this target variable
Kr	KR (Prediction Confidence) of the variable with respect to this target variable
Ks	Kolmogorof-Smirnoff of the variable with respect to this target variable
AUC	Area Under the ROC Curve of the variable with respect to this target variable

Parameter	Description
GINI	GINI index of the variable with respect to this target variable
GroupProbaDeviation	Probability that there is a significant deviation in the distribution of the categories groups. created with respect to the target, between this data set and the data set jointed by the parameter StatForCode
Groups	<p>Statistics of the groups that have been created for the target variable.</p> <p>The grouping is different for each target variable, as the grouping strategy depends on the target variable itself. Groups for continuous variables are made. The information under the "Groups" folder is equivalent to the information on the original categories of the variables. For each, you will find a sub tree under the group label. The sub tree is exactly equivalent to the one found under CategoryName (see below).</p>
OrderForTarget	<p>Default (or "natural") order of categories that gives the best KI (Predictive Power) for the variable with respect to the target. This parameter is especially useful for continuous and ordinal variables to express non-linear relationship with the target, if the value is "profit" or "basic profit". Possible values:</p> <ul style="list-style-type: none"> • Increase • Decrease • Profit • Basic Profit
TargetCurvePoints	<p>Only for continuous variables.</p> <p>This parameter corresponds to the points allowing to draw the curve used to encode a continuous variable with respect to the target. Each point is given through [X, Y] coordinates.</p>
BarCurvePoints	<p>Only when the variable is an estimator of another variable (for example, rr_<target>).</p> <p>It corresponds to the points that allow drawing the piecewise linear interpolation curve giving the error bar in relation to the score. This values can be used to plot an expected error bar around the predicted value.</p>
ProbabilityCurvePoints	<p>Only when the variable is an estimator of a nominal variable (for example rr_<target> when target is nominal).</p> <p>It corresponds to the points that allow drawing the piecewise linear interpolation curve giving the probability in relation to the score.</p>

7.3.1.2.2 Categories

⌘ Syntax

Path: Protocols/Default/Variables/<Variable_Name>/Statistics/<Dataset_Name>/Categories/<Category_Name>

Parameter	Description
Code	Code of the category is randomly assigned in the order of category appearance in the dataset
Count	Number of cases where this category has been encountered on the dataset.
Frequency	Category frequency (ratio of the cases with this category on the number of cases where this variable is known). This is found only in the parameter tree when presented to the user (not in the saved version of the models).
ProbaDeviation	Probability that the category is significantly different from its distribution in StatForCode
SegmentMean	Mean of the variable for this segment when this variable is continuous.
TargetMean	Mean of the target for cases belonging to this category when the target is continuous.
SmoothedTargetMean	TargetMean corrected to obtain a smoothed cycle. This component will only be usable for a model generated by SAP Predictive Analytics Modeler – Time Series.
TargetVariance	Variance of the target for cases belonging to this category when the target is continuous.
UserProfit	Profit that can be associated with this category when the variable is the target variable. This user profit is then used when asking the model for profit curve with user profits.
Targets	Subtree collecting information about how the (several) targets are distributed for cases with this category in the data set. Of course, this is only available when the variable is not a Target Variable Name
NormalProfit	Coding number associated with this category and used by SAP Predictive Analytics Modeler - Regression/Classification in order to translate the category name into a number when appropriate.
TargetRank	Rank of this category with respect to this target
GroupCode	Technical detail reserved for internal use. This parameter links towards the code of the associated group, if any.
ProbaDeviation	Probability that the category is significantly different from its distribution in StatForCode, taken the distribution over the groups into account for the target under consideration.
TargetCategories	Subtree collecting information on all the categories of the target: <Target_Category_Name> <ul style="list-style-type: none"> Count Number of times the category of the target has been encountered on the data set for cases with this category.

7.3.1.3 Data Structure

Concept hierarchies allow the mining of knowledge at multiple levels of abstraction. A concept hierarchy defines a sequence of mappings from a low level concepts to a higher level, and more generally, concepts related to a single dimension (or variable, or column). In general, concept hierarchies describe several levels of abstraction. In SAP Predictive Analytics, we have focused on representing the first level of concept hierarchies

that we call: Data structures. A data structure can be seen as a user defined specific initial binning/banding before the eventual compression. Data structures come from different viewpoints:

- A data structure provided by system users generally reflects a grouping used to store information in large scale data bases or OLAP systems.
- A data structure provided by domain experts generally reflects background knowledge (a dimension location aggregates cities in states, then aggregated in countries).
- A data structure provided by knowledge engineers generally reflects a grouping strategy to improve the robustness of internal representations (a category very infrequent is aggregated with a larger one).
- A data structure can be automatically generated by SAP Predictive Analytics components or an external tool (some tools generate ranges for continuous variables).

A data structure can be used by the components to build higher levels of abstraction. A data structure represents the first level of aggregation of concept hierarchies. Data structure elements depend upon the variable type (nominal, ordinal, or continuous).

- Nominal Variables

The data structure is described by groups of categories. Each group is designed with a name for the user, and the list of possible values belonging to this group. In version 2.1 of the components, this list must be given in extension (all values must be listed), but the possibility to use regular expressions will be given in the future. The entry point in the parameter tree is called `NominalGroups`.

- Ordinal Variables

The data structure is described by ranges of values (called bands). A band is defined by a name for the user, a minimum value, and a maximum value. These two values are assumed to belong to the range. These values can be either number or strings (for which the alphabetical order will be used as the sort function). The entry point in the parameter tree is called `OrdinalBands`.

- Continuous Variables

The data structure is described by ranges of values (called bands). A band is defined by a name for the user, a minimum value, a flag indicating if the minimum value belongs to the range or not (open or closed boundary), a maximum value, and a flag indicating if the maximum value belongs to the range or not (open or closed boundary). The data structure for continuous variables is checked after the user has described it from a parameter tree. The system automatically checks if the bands given by the user overlap (in which case it outputs an error message), or if there is a 'hole' between successive bands (in which case the system completes with the needed segment). The entry point in the parameter tree is called `ContinuousBands`.

7.3.2 Protocol Parameters

☰ Syntax

Path: `Protocols/Default/Parameters`

The following table lists the parameters that can be accessed under the protocol `Parameters` section:

Parameters	Access	Description	Possible Values
<code>StrangeValueLevel</code>	Read-write	Used to send warning messages to the client context when strange values are encountered by one transform in the protocol	<ul style="list-style-type: none"> 10 (default value) Positive integer
<code>VariableCountInSpace</code>	Read-only	Number of variables in the source data sets	<ul style="list-style-type: none"> 0 (default value) Positive integer
<code>WeightIndex</code>	Read-only	Index of the weight variable in the source data sets. A <code>WeightIndex</code> of <code>-1</code> means that there is no weight column defined in the source data set	<ul style="list-style-type: none"> -1 (default value) Integer
<code>StrangeValuePolicy</code>	Read-write read-only when the model is in ready state	The strange value policy is used when the system checks the compatibility of the transforms and the available data. This "check" phase takes place before model generation (or training). After the model has been generated, this value is no longer used, because it is converted into a set of actions to take place for each variable contained into the protocols.	<ul style="list-style-type: none"> Dialog (default value) skip
<code>CopyTarget</code>		Forces the copy of the target into the apply output during an apply process.	<ul style="list-style-type: none"> true (default value) false
<code>SoftMode</code>		Enables a mode where all the involved transforms produce a default model when no model is produced by the standard learning process. In this mode, when no model is found, SAP Predictive Analytics Modeler - Regression/Classification produces a constant model and SAP Predictive Analytics Modeler – Time Series produces a Lag1 Model. No effect on other transforms. This mode is disabled by default.	<ul style="list-style-type: none"> false (default value) true

7.3.3 Transforms

☰ Syntax

Path: `Protocols/Default/Transforms/`

Transforms are holding the actual statistical or machine learning algorithms. All transforms must go through a learning phase to be ready for use. This learning phase is used for the estimation of parameters, and the computation of some results or descriptive information. The transforms held by the protocols are the data processing unit.

Whilst most transforms parameters are specific each the kind of transform used (SAP Predictive Analytics Modeler - Data Encoding, SAP Predictive Analytics Modeler - Regression/Classification,... see below), some are common to all transform types.

7.3.3.1 Transform Information

≡ Syntax

Path: `Protocols/Default/Transforms/<Transform_Name>/Infos`

Additional information is stored in this folder after the model has been generated, that is, when the `Parameters/State` value is `ready`. All these parameters are read-only.

Parameter	Access	Description
<code>LearnTime</code>	Read-only	The time (in seconds) that this transform took to process the learn request.

≡ Syntax

Path: `Protocols/Default/Transforms/<Transform_Name>/Parameters`

The following parameters can be found in the `Parameters` folder of each Transform:

Parameter	Access	Description	Values
<code>State</code>	Read-only	The state of the transforms. As all objects, a model is described by a state diagram. States are used internally to know what are the possible actions that can be asked on these objects. This is an internal parameter and the user cannot change the state of a model. Most of the time, when a model has been trained, it will be saved in the "ready" state. When a model is saved before training, it can be saved with a "checked" state and used as a template.	<ul style="list-style-type: none">Created(default value)ReadyCheckedRunning
<code>VariablePrefix</code>	Read-only when transform is in ready state	It indicates to the user the used prefix on the target.	<ul style="list-style-type: none"><code>rr</code> (default)User-defined string

≡ Syntax

Path: `Protocols/Default/Transforms/<Transform_Name>`

This parameter can be found at the root of each transform:

Parameter	Description
<code>Extensions</code>	Folders under which an integrator can add specific information. Any information stored in these folders will be saved in the model.

7.3.3.2 Kxen.RobustRegression

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression`

SAP Predictive Analytics Modeler - Regression/Classification builds models implementing a mapping between a set of descriptive attributes (model inputs) and a target attribute (model output). It belongs to the regression algorithms family building predictive models.

SAP Predictive Analytics Modeler - Regression/Classification uses a proprietary algorithm, a derivation of a principle described by V. Vapnik as "*Structural Risk Minimization*". It builds predictive models from a training data set containing a business question. The returned models are expressed as polynomial expression of the input numbers. The only element specified by the user is the polynomial degree.

It also allows to specify weighting factor for each training case in order to adapt the cost function to the user requirements. The output model can be analyzed in terms of attributes contribution weighing the relative importance of the inputs and is characterized by two indicators:

- the Predictive Power (KI)
- and the Prediction Confidence (KR).

The consistent (robust) models built with SAP Predictive Analytics Modeler - Regression/Classification are very efficient with very noisy data sets. Furthermore, based on polynomial functions, the model learning phase is very fast: only 50 seconds for a problem on 50,000 cases described with 13 attributes. This speed is obtained because SAP Predictive Analytics Modeler processes data with a single pass on the estimation set and only a few passes on the validation set. The behavior of SAP Predictive Analytics is almost linear with the number of lines of the training set, but it is combinatorial with respect to the degree of the polynomial and the input variables number.

7.3.3.2.1 Thresholds

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Thresholds/`

This folder contains one subfolder for each discrete target variable and defines for each the threshold used for the classification decision.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Thresholds/<Target_Name>`

This folder contains the threshold of the target variable.

☰ Syntax

Path `Protocols/Default/Transforms/Kxen.RobustRegression/Thresholds/<Target_Name>/Threshold`

This folder indicates the score used to separate positive observations from negative observations (number between 0 and 1). If no value is set by the user, the system value provided in the `Default` subparameter is used.

7.3.3.2.2 SelectionProcess

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/SelectionProcess`

This folder contains the results of the variable selection process.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/SelectionProcess/Iterations`

This folder contains all the iterations made during the selection process.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/SelectionProcess/Iterations/<Iteration_Number>`

Number of the iteration of the variable selection process. This folder contains the following parameters:

Parameter	Description
Ki	Predictive Power (KI) obtained on the validation dataset with the current iteration. The predictive power is the quality indicator of the models generated using Chile. This indicator corresponds to the proportion of information contained in the target variable that the explanatory variables are able to explain.
Kr	Prediction Confidence (KR) obtained on the estimation dataset with the current iteration. The prediction confidence is the robustness indicator of the models generated using Chile. It indicates the capacity of the model to achieve the same performance when it is applied to a new data set exhibiting the same characteristics as the training data set.
KiE	Predictive Power (KI) obtained on the estimation dataset with the current iteration.
L1	L1 obtained with the current iteration L1 is the residual mean (the mean of the absolute value of the difference between the predicted value and the actual value); also known as City Block distance or Manhattan distance.

Parameter	Description
L2	L2 obtained with the current iteration. L2 is the square root of the mean of square residuals.
LInf	LInf obtained with the current iteration. LInf is the maximum residual absolute value.
NbVariablesKept	Number of variables selected for this iteration.
Chosen	This parameter indicates whether the current iteration is the one selected by the variable selection process. <ul style="list-style-type: none"> • <code>true</code>: the current iteration is the selected one • <code>false</code>

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/SelectionProcess/Iterations/LastUsedIterations`

This folder contains the list of the input dataset variables with the number of the last iteration in which each was used.

7.3.3.2.3 Transform Parameters

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters`

This section describes the parameters of the classification/regression engine.

Parameter	Description	Values
Order	The degree of the polynomial model. This parameter is set by the user before the learning phase and cannot be changed later on for a given model.	<ul style="list-style-type: none"> • 1 (default) • A positive integer
K2RMode	A specific interpretation mode for scores. If the value is set to either <code>Rule Mode</code> or <code>Risk Mode</code> , the corresponding parameter <code>RuleMode</code> or <code>RiskMode</code> is set to <code>true</code> .	<ul style="list-style-type: none"> • <code>Standard Mode</code> (default) • <code>Rule Mode</code> • <code>Risk Mode</code>
Strategy	Allows the user to specify the type of strategy used to build the regression model	<ul style="list-style-type: none"> • <code>WithoutPostProcessing</code> (default) • <code>WithOriginalTargetEncoding</code> • <code>WithUniformTargetEncoding</code>

Parameter	Description	Values
EncodingStrategy	Controls the way the inputs are encoded. By default the encoding in a piece-wise linear. When in risk mode, SAP Predictive Analytics uses a step wise linear encoding.	<ul style="list-style-type: none"> PieceWise Encoding (default) StepWise Encoding
MaximumKeptCorrelations	Allows the user to set the maximum number of displayed correlation. This parameter accepts only an unsigned integer.	<ul style="list-style-type: none"> 1024 (default) Integer
LowerBound	Allows the user to set the threshold to define if a correlation has to be displayed or not.	<ul style="list-style-type: none"> 0.5 (default) real number
ContinuousEncode	Allows the user to de-activate the encoding of the continuous variables. This is given to compare results between version 1 and version 2 and allows a more precise control for advanced users.	<ul style="list-style-type: none"> true (default) false
ExtraMode	<p>A special flag that drive the kind of outputs that the classification/regression engine will generate.</p> <p>Depending on its value, the outputs generation will be done either in the expert mode or the assisted mode. The former allows selectively choosing the outputs and the latter will generate consistent and system-defined outputs.</p> <p>Note that when switching from the expert mode to the assisted one, the user choices are discarded and replaced by those implied by the specified extra mode value.</p>	<p>Assisted Modes / Generated outputs</p> <ul style="list-style-type: none"> No Extra <i>default</i> / key + predicted value + score Min Extra / No Extra + probabilities + error bars IndividualContributions / No Extra + variable individual contributions Decision / No Extra + decision Quantiles / No Extra + approximated quantiles Expert Mode Advanced Apply Settings (set in ApplySettings)
PutTargetNameInIndividualContrib	<p>Used to guarantee the backward compatibility with version prior to 2.1.1. It will prevent generating the name of the target in the individual contributions column names</p> <p>Available only for mono-target models</p>	<ul style="list-style-type: none"> true (default) false
ScoreUserBoundCount	Defines the number of bins for score variables	<ul style="list-style-type: none"> 20 (default) Integer
DecisionTreeMode	Activates the ability to present SAP Predictive Analytics Modeler - Regression/Classification as a decision tree.	<ul style="list-style-type: none"> true false (default)

7.3.3.2.3.1 IDBScoreDevConfig

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/IDBScoreDevConfig`

This folder contains the parameter allowing you to generate SQL code that computes score deviations for the model. This code will be executed at the end of the in-database application process.

Parameter	Description	Values
Apply	This parameter allows you to activate or deactivate the IDBScoreDevConfig feature.	<ul style="list-style-type: none">• <code>true</code> (default): activated• <code>false</code>: not activated

7.3.3.2.3.2 GainChartConfig

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/GainChartConfig`

This folder contains the parameters allowing to compute the gain chart. The gain chart allows you to rank your data in order of descending scores and split it into exact quantiles (decile, vingtile, percentile).

The gain chart can be computed when training a model or when applying it. Two different folders contain the gain chart parameters depending on the task to perform on the model:

- `Learn`: when training a model
- `Apply`: when applying a model on new data

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/GainChartConfig/Learn`

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/GainChartConfig/Apply`

Parameter	Description	Values
NbQuantiles	This parameter allows you to set the number of quantiles you want to compute for the gain chart.	<ul style="list-style-type: none">• 10 (default)• Positive integer

Parameter	Description	Values
ValueVariables	This folder allows you to set the list of the variables for which aggregated values must be computed for each quantiles. Each variable is a sub-parameter of this folder and is defined by its name.	

7.3.3.2.3.3 VariableExclusionSettings

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/VariableExclusionSettings`

Contains the settings for variable exclusion during modeling.

Parameter	Description	Values
ExcludeSmallKR	This parameter allows you to indicate whether variables with a low predictive confidence (KR) must be excluded from the modeling.	<ul style="list-style-type: none"> • <code>system</code> (default): the value (<code>true/false</code>) is automatically selected by SAP Predictive Analytics • <code>true</code>: the variables will be excluded • <code>false</code>: the variable will not be excluded
ExcludeSmallKIAddKR	This parameter allows you to indicate if the variables for which the sum of the predictive power and the prediction confidence (KI+KR) is too low must be excluded from the modeling.	<ul style="list-style-type: none"> • <code>system</code> (default): the value (<code>true/false</code>) is automatically selected by SAP Predictive Analytics • <code>true</code>: the variables will be excluded • <code>false</code>: the variable will not be excluded

7.3.3.2.3.4 VariableSelection

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/VariableSelection`

Description	Values
When set to true, the variable selection is enabled. This folder contains the parameters used by the variable selection process.	<ul style="list-style-type: none"> • true • false (default)

Parameter	Description	Values
DumpIntermediateSteps	It allows the user to specify that all intermediate iterations are saved in the parameter tree or not.	<ul style="list-style-type: none"> • true • false (default)
SelectionMode	This folder contains the settings of the automatic variable selection process.	
StopCriteria	Contains the settings of the stop criteria to use for the variable selection process.	

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/VariableSelection/SelectionMode`

Parameter	Description	Values
Mode	Allows the user to specify the type of automatic process variable selection.	<ul style="list-style-type: none"> • ContributionBased: for each iteration, the variables that contains the less quantity of information are skipped. Used with the parameter PercentageContribution. • VariableBased: for each iteration, a specified number of variables is skipped. Used with the parameter NbVariableRemovedByStep.
NbVariableRemovedByStep	Allows the user to specify, in the case of the automatic variable selection is in VariableBased, the number of skipped variables by iteration process.	<ul style="list-style-type: none"> • 1 (default) • integer
PercentageContribution	Allows the user to specify the percentage amount of information to keep	<ul style="list-style-type: none"> • 0.95 (default) • Real number

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/VariableSelection/StopCriteria`

Parameter	Description	Values
QualityCriteria	Allows the user to set the type of quality criteria to be used for the automatic variables selection	<ul style="list-style-type: none"> • None: no quality criteria is set • KiKr:(default) the quality criteria is based on the sum of the predictive power (KI) and the prediction confidence (KR). • Ki: the quality criteria is based on the sum of the predictive power (KI) • Kr: the quality criteria is based on the sum of the prediction confidence (KR).
MaxNbIterations	Allows the user to stop the automatic variable selection process when the number of iterations exceeds this value.	<ul style="list-style-type: none"> • Integer • -1 (default): no limit
MinNbOfFinalVariables	Allows the user to fix the final number of kept variable in the final model.	<ul style="list-style-type: none"> • 1 (default) • Integer
MaxNbOfFinalVariables	Allows the user to fix the maximum number of variable to keep in the final model	<ul style="list-style-type: none"> • -1 (default) : all variables • positive integer
QualityBar	Allows the user to specify the quality loss by iteration.	<ul style="list-style-type: none"> • 0.01 (default) • Real number
ExactNumberOfVariables	Allows the user to force the final number of variable to be equal to MinNbOfFinalVariables	<ul style="list-style-type: none"> • true • false (default)
SelectBestIteration	Allows you to select which model of the variable selection process will be used. Usually the best model is the one before last, however the quality of the last model can be sufficient for your needs and you may want to use it instead.	<ul style="list-style-type: none"> • true (default): the best model will be selected • false: the last model will be selected
FastVariableUpperBoundSelection	Allows you to define the strategy to use when you have set the parameter MaxNbOfFinalVariables. Two strategies are available: <ul style="list-style-type: none"> • Removing the variables with the lowest contribution to reach directly the expected number of variables (true). • Creating as many iterations as needed to reach the expected number of variables (false). Note that using this strategy can be resource-consuming if the initial number of variables is high and the expected number is low. 	<ul style="list-style-type: none"> • true (default) • false

7.3.3.2.3.5 RiskMode

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/RiskMode`

Description	Values
<p>Allows the user to activate the risk mode for a classification model. It allows advanced users to ask a classification model to translate its internal equation obtained with no constraints into a specified range of scores associated with a specific initial score. When this mode is activated, the different types of encoding that are used internally for continuous and ordinal variables are merged in a single representation, allowing a simpler view of the model internal equations. To use this mode, you need to choose a range of scores associated with probabilities.</p> <p>Available only for classification models, that is models with a nominal target</p>	<ul style="list-style-type: none">• <code>true</code>• <code>false</code> (default)

Parameters

Parameter	Description	Values
PDO	<p>PDO stands for Points to Double the Odds.</p> <p>This parameter allows the user to specify the low score associated with the low probability.</p>	<ul style="list-style-type: none">• 15 (default)• Real number
RiskScore	<p>Allows the user to specify a low probability that will be associated with a low score.</p>	<ul style="list-style-type: none">• 615 (default)• Real number
GBO	<p>Allows the user to specify a high probability that will be associated with a high score.</p>	<ul style="list-style-type: none">• 9 (default)• Real number

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/RiskMode/RiskFitting`

Description	Values
<p>This folder contains the parameters allowing the user to control the way risk score fitting is performed, that is, how SAP Predictive Analytics fits its own scores to the risk scores.</p>	<ul style="list-style-type: none">• <code>Frequency_Based</code> (default)• <code>PDO_Based</code>

Parameters

Parameter	Description	Values
NbPDO	When RiskFitting is set to PDO_Based, indicates the number of PDOs around the median score. It is used to compute the fitting area: [Median Score - NbPDO*PDO ; Median Score + NbPDO*PDO].	<ul style="list-style-type: none"> • 2 (default) • Integer
MinCumulatedFrequency	When RiskFitting is set to Frequency_Based, indicates the frequency of extreme scores to be skipped. It is used to compute the fitting area: [Quantile (MinCumulatedFrequency) ; Quantile (1.0 - MinCumulatedFrequency)]	<ul style="list-style-type: none"> • 15 (default) • Integer
UseWeights	Indicates whether to use score bin frequency as weights	<ul style="list-style-type: none"> • true (default) • false

7.3.3.2.3.6 DecisionTree

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/DecisionTree`

Description	Values
This folder allows writing a request on the decision tree and obtaining the results. Available only after model training when DecisionTreeMode is set to true.	<ul style="list-style-type: none"> • true • false (default)

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/DecisionTree/DimensionOrder`

Contains one folder per target. Each folder contains the list of the 5 most contributive variables.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/DecisionTree/DimensionOrder/Request`

Allows creating a request to obtain a leaf or a level of the decision tree. The request will be processed after a `validateParameter` when `ProcessRequest` value is `true`.

Parameter	Description	Values
<code>Target</code>	Specifies the target to use in the request.	
<code>ProcessRequest</code>	Indicates whether the request must be executed or not.	<ul style="list-style-type: none"> <code>true</code> (default): the request will be executed <code>false</code>: the request will not be executed
<code>Datasets</code>	Lists the data sets on which the request will be made	
<code>RequestBody</code>	Contains the list of dimensions on which the request will be made. The value of each dimension must be the group code. If a dimension has an empty group code then all the leaves of this node will be computed. Only one dimension can be empty.	

Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/DecisionTree/DimensionOrder/Result`

Contains the result of the request and is displayed after a `getParameter` when `ProcessRequest` is set to `true`.

Parameter	Description
<code>SingleResult</code>	Subparameters:
<code>ExpandResult</code>	<ul style="list-style-type: none"> <code>Count</code>: population of the current node <code>TargetMean</code>: mean of the target in the current node <code>Weight</code>: weight of the current node <code>Variance</code>: variance of the current node

7.3.3.2.3.7 ApplySettings

Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings`

This parameter allows you to set the advanced application settings, that is, to select and fine-tune the outputs that SAP Predictive Analytics will generate. These outputs belong to one of the two following groups: supervised (target-dependent) or unsupervised (non target-dependent).

Supervised

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/Supervised`

It is the section for defining target-dependent outputs for classification/regression models. Each of its subsections corresponds to a target and is outlined depending on this variable type.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/Supervised/<Target_Name>`

Contains the application settings related to the current target.

Parameter	Description	Values
Contribution	Specifies whether contributions of variables should be produced or not.	<ul style="list-style-type: none">• None (default): the flag is deactivated.• All: all the contributions will be produced.• Individual: specific variables of interest can be selected (inserted as sub-nodes).
Confidence	Specifies whether or not to produce the confidence (also known as error bar). Available for continuous or ordinal variables only	<ul style="list-style-type: none">• true• false (default)
Inputs	Allows to transfer apply-in input variables to apply-out variables Note - only the variables used in the model can be transferred.	<ul style="list-style-type: none">• None: the flag is deactivated.• All: all the variables will be used in the apply-in data set.• Individual (default): specific variables of interest can be selected (inserted as sub-nodes).
OutlierFlag	Specifies whether or not to produce the outlier flag	<ul style="list-style-type: none">• true• false (default)
PredictedQuantile	Indicates whether the quantile associated with the score value should be produced or not. This flag is activated by providing a quantile level greater than 0. Available for continuous or ordinal variables only	<ul style="list-style-type: none">• true• false (default)

Parameter	Description	Values
PredictedValue	Indicates whether or not to produce the score Available for continuous or ordinal variables only	<ul style="list-style-type: none"> • true (default) • false
OutputCosts	(Not Used)	
PredictedCategoryConfidence	Allows generating in the output file the confidence (also known as the error bar) corresponding to each data set line for the different categories of the target variable.	<ul style="list-style-type: none"> • none: the flag is deactivated. • all: the confidence is generated for all categories. • individual: the confidence will be generated for selected (inserted as sub-nodes) categories only.
PredictedCategoryProbabilities	Allows generating in the output file the probability for one or more target variable categories, that is, for each observation the probability of the target variable value to be the selected category.	<ul style="list-style-type: none"> • none: the flag is deactivated. • all: the probability is generated for each category. • individual: the probability will be generated for selected (inserted as sub-nodes) categories only.
PredictedCategoryProfits	(Not Used)	
PredictedCategoryQuantile	(Not Used)	
PredictedCategoryScores	Allows generating in the output file the score for one or more target variable categories.	<ul style="list-style-type: none"> • none: the flag is deactivated and no score will be generated. • all: all the scores will be produced. • individual: the score will be generated for selected (inserted as sub-nodes) categories only.

Parameter	Description	Values
PredictedRankCategories	Allows generating in the output file the best decisions.	<ul style="list-style-type: none"> • none : the flag is deactivated. • all : all the decisions (ranked by their associated score value) will be produced. • individual : the specified count of best decisions is generated. This count has to be provided (inserted as a sub-node).
PredictedRankProbabilities	Allows generating in the output file the score for one or more target variable categories.	<ul style="list-style-type: none"> • none : the flag is deactivated and no score will be generated. • all : all the scores will be produced. • individual : the score will be generated for selected (inserted as sub-nodes) categories only.
PredictedRankScores	Allows to generate in the output file the best score(s) for each observation.	<ul style="list-style-type: none"> • none : the flag is deactivated. • all : all the scores will be produced. • individual : the requested number of 'best scores' is generated. This value has to be provided (inserted as a sub-node).
ProfitMatrix	(Not Used)	
RankType	(Not Used)	

UnSupervised

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/UnSupervised`

This folder contains only information non-dependent on a target, for example constants such as the date of learn, the application date, the model version.

Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/UnSupervised/Default`

Default subfolder containing the settings that do not depend on the target. This folder always exists and cannot be changed.

Parameter	Description	Values
<code>DatasetId</code>	This parameter allows you to add a column containing the name of the data set each line belongs to. The possible output values in this column are: <ul style="list-style-type: none">• Estimation• Validation• Test• Applyin	<ul style="list-style-type: none">• <code>False</code> (default): the column is not added• <code>True</code>: the column is added to the output data set
<code>Inputs</code>	Allows you to add input variables to the output data set. When the value is set to <code>individual</code> , this parameter becomes a folder containing the variables selected by the user.	<ul style="list-style-type: none">• <code>none</code> (default): no input variable is added to the output data set• <code>all</code>: all input variables are added to the output data set• <code>individual</code>: only the input variables selected by the user will be added to the output data set
<code>Weight</code>	If a weight variable has been defined during the variable selection of the model, this parameter allows you to add it to the output file	<ul style="list-style-type: none">• <code>false</code> (default): the weight variable will not be added to the output data set• <code>true</code>: the weight variable will be added to the output data set

Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/UnSupervised/Default/Constants`

This folder contains the constants related to the model. Its value is always set to `export`.

Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/ApplySettings/UnSupervised/Default/Constants/<Constant_Name>`

Description	Values
<p>This folder is the name of the constant to be generated in the output data set. The constant name can be one of the following:</p> <ul style="list-style-type: none"> Apply Date (date when the model was applied) Build Date (date when the model was created) Model Name Model Version User defined string (you can define your own constants, for example the name of the person who created the model, or the project it belongs to). The name cannot be the same as the name of an existing variable of the reference data set. If the name is the same as an already existing user defined constant, the new constant will replace the previous one. It contains all the settings needed to define a variable in the output data set. 	<ul style="list-style-type: none"> export: the constant will appear in the output data set skip (default): the constant will not appear in the data set

Parameter	Description	Values
OutVarName	Name of the constant in the output data set	By default the value is the name of the constant, but it can be modified to fit database restrictions for example.
Value	Value of the constant that will appear in the output data set	Depends on the value of the storage parameter of the constant
KeyLevel	This parameter allows you to indicate whether the current constant is a key variable or identifier for the record . You can declare multiple keys. They will be build according to the indicated order (1-2-3-...).	<ul style="list-style-type: none"> 0 (default): the variable is not an identifier 1: primary identifier 2: secondary identifier

Parameter	Description	Values
Storage	This parameter allows you to set which kind of values are stored in this variable	<ul style="list-style-type: none"> Number: "computable" numbers (be careful: a telephone number, or an account number should not be considered numbers) Integer: integer numbers String: character strings Datetime: date and time stamps Date: dates
Origin	This parameter indicates the origin of the constant.	<ul style="list-style-type: none"> BuiltIn: automatically generated by SAP Predictive Analytics UserDefined: created by the user

7.3.3.2.4 Results

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results`

This section describes the results for regression and classification models. Under this folder, there is a sub-folder for each target variable. SAP Predictive Analytics Modeler can handle several targets at the same time. All the parameters dealt with in this section are read-only once the model has been generated.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>`

`<Target_Name>` corresponds to the name of the target for which the results are listed. There can be more than one target in the same model.

Parameter	Description	Values
NbInput	The number of explanatory – or, input – variables of the regression engine. It corresponds to the sum of the values of the parameters <code>NbOrgInput</code> and <code>NbContInput</code> .	Integer
NbOrgInput	The original number of explanatory input variables.	Integer
NbContInput	The number of encoded continuous explanatory variables.	Integer

Parameter	Description	Values
Order	A reminder of the polynomial degree (corresponding to the model complexity).	Integer
NbExtended	The number of explanatory and extended variables. When the polynomial degree is 1, this number equals the number of inputs + 1 (the constant value). When the polynomial degree is more than 1, then the number of extended variables corresponds to the number of cross products that can be created from these original variables.	Integer
ConstantModel	If set to <code>true</code> , the model is constant.	<ul style="list-style-type: none"> • <code>true</code> • <code>false</code>
Constant	The score produced for a constant model.	<ul style="list-style-type: none"> • 0 (default) • Real number
MaximumKeptCorrelations	It links to the other tree parameter called <code>MaximumKeptCorrelations</code> in the <i>Parameters</i> section.	Real value which equals the value of <code>MaximumKeptCorrelations</code>
LowerBound	It links to the other tree parameter called <code>LowerBound</code> .	Same value as <code>Parameters/LowerBound</code>
ZscoreCurve	Stores the encoding curve used to translate scores to the target space when in a postprocessing regression strategy.	
DiscardedVariables	Folder containing the list of discarded variables in the regression engine	

7.3.3.2.4.1 DataSets

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/DataSets`

This folder contains performance indicators for each data set that has been evaluated by the model.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/DataSets/<Dataset_Name>`

`Dataset_Name` is the name of the dataset that has been evaluated by the model. The name of the dataset can be one of the following:

For the training:

- Estimation
- Validation
- Test
- ApplyIn

For the application:

- ApplyIn

Parameters

Parameter	Description
L1	The residual mean (the mean of the absolute value of the difference between the predicted value and the actual value); also known as City Block distance or Manhattan distance.
L2	The square root of the mean of square residuals.
LInf	The maximum residual absolute value
ErrorMean	The mean of the error, that is, the difference between predicted values and actual values
ErrorStdDev	The standard deviation
ClassificationRate	Used in classification case, that is with a nominal target Classification rate. The threshold is used to make the decision.
R2	Used in in regression case, that is with a continuous target The R2 is computed as the square correlation between the target and the model output (prefixed by <code>rr_</code>).

7.3.3.2.4.2 GainChartResults

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/GainChartResults`

This directory contains information about the gain chart computed for each action made on the model (training or application).

Parameter	Description
Learn	This directory contains information about the gain charts computed for the trained model.

Parameter	Description
Transversal	This directory contains information about the gain charts computed for the applied model.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/GainChartResults/<Model_Type>/Quantiles`

This folder contains a folder for each quantile computed for the current gain chart. `<Model_Type>` is either Learn Or Transversal.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/GainChartResults/<Model_Type>/Quantiles/<Quantile_Index>`

This folder contains the metrics computed for the current quantile. `<Quantile_Index>` is the number of the quantile. Quantile 1 is the one containing the highest number of positive observations.

Parameter	Description
Weight	Number of observations in the current quantile
MinScore	All observations contained in the current quantile have a score equal or above this value.
MaxScore	All observations contained in the current quantile have a score equal or below this value.
Predicted	Number of positive observations predicted by the model in the current quantile
Actual	Actual number of positive observations existing in the current quantile
Values	This folder contains the variables for which the value has been aggregated. These variables are defined in the parameter <code>Protocols/Default/Transforms/Kxen.RobustRegression/Parameters/GainChartConfig/<Step Name>/ValueVariables</code>

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/GainChartResults/<Model_Type>/Quantiles/<Quantile_Index>/Values/<Variable_Name>/Value`

`<Variable_Name>` is the name of the variable whose value is aggregated. `Value` parameter is the aggregated value of the current variable for the current quantile.

7.3.3.2.4.3 Coefficients

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Coefficients`

Directory where the actual polynomial coefficients are stored. Coefficients can be used to determine how the system is using the extended variables. This directory contains one sub-directory for each of the extended variables.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Coefficients/<Variable_Name>`

`<Variable_Name>` corresponds to the extended variable for which the coefficients are listed.

Parameter	Description
<code>PseudoCoeff</code>	Coefficient of the corresponding extended variable in the polynomial
<code>Min</code>	Used for normalization purposes
<code>Range</code>	Used for normalization purposes
<code>Mean</code>	Used for normalization purposes
<code>StDev</code>	Standard Deviation, used for normalization purposes
<code>Weight</code>	Extended variable weight. This weight is the coefficient associated with the normalized extended variable, divided by the sum of all the coefficients.
<code>Contrib</code>	Coefficient associated with the normalized extended variable absolute value, divided by the sum of all the normalized extended variables coefficients absolute values

7.3.3.2.4.4 SmartCoefficients

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/SmartCoefficients`

Directory where the smart coefficients are stored. Smart coefficients are just another view of the coefficients in which the redundancy between the variables is removed. When two variables are very correlated, the robust system K2R will almost equalize the contributions on the two variables, the smart coefficients view will put almost all the contribution on the most contributive variable (which we call the leader variable) out of the very correlated ones, and will translate the remaining variables into the difference between the leader variable and this variable. Smart Coefficients can be used to perform variable selection.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/SmartCoefficients/<Variable_Name>`

`<Variable_Name>` corresponds to the extended variable for which the smart coefficients are listed.

Parameter	Description
Weight	Extended variable weight. This weight is the coefficient associated with the normalized extended variable, divided by the sum of all the coefficients.
Contrib	Coefficient associated with the normalized extended variable absolute value, divided by the sum of all the normalized extended variables coefficients absolute values

7.3.3.2.4.5 MaxCoefficients

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/MaxCoefficients`

Directory where the maximum coefficients are stored. Max coefficients are obtained from the smart coefficients, which are sorted and scanned in decreasing order of their contributions. The purpose of `MaxCoefficients` is to cope with the fact that the user wants to have a single view about the importance of a variable, even if this variable is encoded twice in the predictive model (such as the continuous variables). This is also true for order 2 models where the original inputs are found in all the cross products. In that case, the maximum smart contribution is first associated with each variable, then sorted accordingly to this new order, finally the sum of the resulting contributions is used to renormalize the coefficients for the sum to be one. `MaxCoefficients` are the one to use when the user wants to perform variable selection.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/MaxCoefficients<Variable_Name>`

`<Variable_Name>` corresponds to the extended variable for which the max coefficients are listed.

Parameter	Description
Weight	Extended variable weight. This weight is the coefficient associated with the normalized extended variable, divided by the sum of all the coefficients.
Contrib	Coefficient associated with the normalized extended variable absolute value, divided by the sum of all the normalized extended variables coefficients absolute values

7.3.3.2.4.6 Rule

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Rule`

This folder contains information about SAP Predictive Analytics rule mode.

Parameter	Description
Slope	The slope of the linear transform to go from basic score to user-defined score.
Intercept	The intercept of the linear transform to go from basic score to user-defined score.
ProbaSlope	Used with ProbaIntercept to transform the score into probability $1 / (1 + \exp(- (\text{ProbaSlope} * \text{score} + \text{ProbaIntercept})))$
ProbaIntercept	Used with ProbaSlope to transform the score into probability $1 / (1 + \exp(- (\text{ProbaSlope} * \text{score} + \text{ProbaIntercept})))$

7.3.3.2.4.7 Correlations

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Correlations`

This directory contains the correlations observed in the model between different variables (for example, correlation between age and marital-status).

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Correlations/<i>`

`<i>` is the index of the correlation used to identify the various correlations found in the model. This index does not imply any order in the correlations.

Parameter	Description	Value
Var1	First variable involved in the correlation	Variable name
Var2	Second variable in the correlation	Variable name
Details	This folder contains the details of the current correlation.	

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/Correlations/<i>/Details/<i>`

<i> is the index of the detailed correlation.

Parameter	Description	Value
Var1	First variable involved in the correlation	Variable name
Var2	Second variable in the correlation	Variable name

7.3.3.2.4.8 AutoCorrelations

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/AutoCorrelations`

This directory contains the auto-correlations observed in the model. These are correlations between a variable and its encoded form (for example, correlation between age and c_age). Given here for its descriptive value.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/Results/<Target_Name>/AutoCorrelations/<i>`

<i> is the index of the correlation used to identify the various correlations found in the model. This index does not imply any order in the correlations.

Parameter	Description	Value
Var1	First variable involved in the autocorrelation	Variable name
Var2	Second variable in the autocorrelation (usually an encoded form of the first variable).	Variable name

7.3.3.2.5 AdditionalResults

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/AdditionalResults/`

This folder provides additional information about the results of the modeling.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/AdditionalResults/VariableExclusionInfo`

This folder provides the list of excluded variables as well as the reason why each variable was excluded by SAP Predictive Analytics depending on the target variable. These are mainly data quality issues.

Parameter	Description
AllTargets	This folder contains the variables that have been excluded from the model with respect to all targets. These are mainly constant variables. The reason why each variable has been excluded is indicated as a subparameter of the variable.
TargetSpecific	This folder contains the list of excluded variables with respect to the given target and the reason why they have been excluded.

Reason

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/AdditionalResults/VariableExclusionInfo/AllTargets/<Variable_Name>/Reason`

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.RobustRegression/AdditionalResults/VariableExclusionInfo/TargetSpecific/<Target_Name>/<Variable_Name>/Reason`

Description	Values
Reason why the variable has been excluded	<p>For AllTargets:</p> <ul style="list-style-type: none">• <code>Constant</code>: the variable has only one value in the data set• <code>Small Variance</code>: the variation of the variable corresponds to noise <p>For TargetSpecific:</p> <ul style="list-style-type: none">• <code>Fully Compressed</code>: fully compressed variable with respect to the target variable• <code>Small KI on Estimation</code>• <code>Small KI on Validation</code>• <code>Large KI Difference</code>: sensible difference between the estimation and validation data sets• <code>Small KR (on the estimation data set)</code>

7.3.3.3 Kxen.SmartSegmenter

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter`

SAP Predictive Analytics Modeler - Segmentation/Clustering (formerly known as K2S) builds models implementing a mapping between a set of descriptive attributes (model inputs) and the id (model output) of one of several clusters computed by the system. It belongs to the clustering algorithms family building descriptive models. The goal of these models is to gather similar data in the same cluster. The question of similarity is discussed below.

Current version of SAP Predictive Analytics Modeler - Segmentation/Clustering uses a K-Means engine to compute the cluster index output variable. K-Means is a method for finding clusters of similar individual within a population. The K-Means method proceeds as follows: starting from an initial position of would-be centers of clusters, the method associates the closest individual with each center, which leads to a first definition of the clusters. The positions of the centers are then adjusted to the true central positions within the clusters. The new positions are then used to recompute the closest individuals, and the process is restarted. This is repeated iteratively until the centers land in a stable position. In practice, the process converges very quickly to a stable configuration.

The distance used to determine the closest center is the L infinite distance in the encoded space generated by SAP Predictive Analytics Modeler - Data Encoding. Hence the segmentation process is *explicitly* supervised, which makes SAP Predictive Analytics Modeler - Segmentation/Clustering a unique clustering algorithm: any distance-based clustering process is supervised, most of the time without even mentioning it ! Indeed, the encoding phase of the process determines entirely the resulting segmentation, since this is the phase that decides what is far, and what is close, that is what is similar to what. When dealing with non continuous variables, the decision can be really tricky, even impossible... for example, how to answer the following question: given three individuals described by the age, the gender and the marital status [31, Male, Married], [41, Male, Divorced], [28, Female, Divorced], which are the most similar individuals ? A quick mind tour tells that there is no answer, unless there is a referee criteria to rely on that will guide the decision. Basically, if gender is chosen then male individuals are grouped, and if marital status is chosen, then divorced individuals are grouped. This example aims to show that there is no segmentation unless a criteria is decided to be relevant with respect to similarity between individuals. This is the very meaning of SAP Predictive Analytics Modeler - Segmentation/Clustering being supervised: the target, or business question, is necessary for SAP Predictive Analytics Modeler - Data Encoding to create a relevant topology that SAP Predictive Analytics Modeler - Segmentation/Clustering will use to detect similarity in data.

As for SAP Predictive Analytics Modeler - Regression/Classification, the target is any variable relevant to the user's business : for example the purchase amount for a customer, the answer to a marketing campaign, the fact that an individual churned in the last 2 months, and so on.

SAP Predictive Analytics Modeler - Segmentation/Clustering is now able to output a SQL formula of the cluster. For example a cluster may be defined as "`age <= 35 AND marital-status in ['Divorced']`". This has several advantages :

- the textual SQL formula may be very easy and natural to interpret if it's not too complex.
- clustering process is made easier to integrate in operational environment

In order to debrief a SAP Predictive Analytics - Segmentation/Clustering model several statistics are provided beside SQL expressions when available:

- Frequency: percentage of population gathered in the cluster.
- % of 'label' in classification case (binary target): percentage of label in the cluster, where label is the least frequent category of the binary target.
- Target Mean in regression case (continuous target): mean value of the target for data assigned to the cluster.
- A detailed description of each cluster is also available via the distribution within the cluster of every input variable.

The model can also be analyzed in terms of two indicators concerning the generated cluster Id variable:

- Predictive Power (KI)
- Predictive Confidence (KR).

The model learning phase takes one minute for a problem on 50,000 cases described with 13 attributes on a regular PC (64-128 MB). SAP Predictive Analytics Modeler - Segmentation/Clustering processes data with 4 sweeps on the estimation set and one sweep on the entire data set. The behavior of SAP Predictive Analytics Modeler - Segmentation/Clustering is almost linear with the number of lines.

7.3.3.3.1 Parameters

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters`

This section describes the parameters of SAP Predictive Analytics Modeler - Segmentation/Clustering component that can be found under the 'Parameters' section of the component.

Parameters	Access	Description	Possible Values
NbClusters	<ul style="list-style-type: none"> • Read-write • Read-only when transform is in ready state 	It is the number of clusters for the model. This parameter is set by the user before the learning phase and cannot be changed later on for a given model.	<ul style="list-style-type: none"> • Positive integer • Default value: 10
InertiaDivider		No longer used	
Distance	<ul style="list-style-type: none"> • Read-write • Read-only when transform is in ready state 	It specifies the distance used to compare SAP Predictive Analytics Modeler - Data Encoding-encoded input data.	
EncodingStrategy		It is a flag that drives the kind of encoding SAP Predictive Analytics Modeler - Segmentation/Clustering is expecting from SAP Predictive Analytics Modeler - Data Encoding.	

Parameters	Access	Description	Possible Values
ExtraMode	Read-write	It is a special flag that drives the kind of outputs that SAP Predictive Analytics Modeler - Segmentation/Clustering will generate. Depending on its value, the outputs generation will be done either in the export mode or the assisted mode. The former allows selectively choosing the outputs and the latter will generate consistent and system-defined outputs. Note that when switching from the expert mode to the assisted one, the user choices are discarded and replaced with those implied by the specified extra mode value.	
CrossStats		It is a flag that enables/disables the computation of cross statistics between cluster ID and input variables.	
SQL		It is a flag that enables/disables the computation of clusters as SQL expressions.	<ul style="list-style-type: none"> disabled(default value): disables cluster generation as SQL expressions. enabled: enables SQL expression. In this case, the CrossStats parameter is overwritten to enabled.
Supervised	Read-only	An informative item taking the value true when the clustering has been built using target variable, false otherwise.	<ul style="list-style-type: none"> true false
MultiEnginesMode		This is the multi-engines mode top level switch.	<ul style="list-style-type: none"> enabled(default value): allows testing several clusters counts (per target) in one run by specifying the range these counts will vary in. Each cluster count will be handled by a dedicated engine. disabled: SAP Predictive Analytics Modeler - Segmentation/Clustering runs in the standard single-engine mode.

Parameters	Access	Description	Possible Values
MultiEnginesModeForApply			
EngineSelectionStrategy		When the multi-engines mode is on, this parameter specifies the criteria used for selecting the best engine (that is, the winner engine).	<ul style="list-style-type: none"> • <code>KiPlusKr</code>(default value): the engine with the higher sum of KI and KR is designated to be the best. • <code>FirstEngine</code>: The first engine is selected (simulates the single engine mode behavior).
BenchMode			
RunnerConfig		Folder	
EnginesConfiguration		The root location for configuration information related to the multi-engines mode.	
ApplySettings	Read-only	This folder can be considered as the advanced-users-intended version of the 'ExtraMode'. It provides the user with a way of fine tuning the outputs that SAP Predictive Analytics Modeler - Segmentation/Clustering will generate.	<ul style="list-style-type: none"> • Supervised (folder) • Unsupervised (folder)

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Distance`

Possible Values	Description
<code>L1</code>	The sum of absolute difference between coordinates
<code>L2</code> or <code>euclidean</code>	The square root of sum of square difference between coordinates
<code>LInf</code>	The maximum of absolute difference between coordinates
also called <code>city-block</code>	
<code>SystemDetermined</code>	Lets the system determine the best distance to be used according to the model build settings. The current policy is to use <code>LInf</code> either in unsupervised mode or when the clusters SQL expressions have been asked for, and <code>L2</code> otherwise.

≡ Syntax

`Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/EncodingStrategy`

Possible Values	Description
Uniform	Each variable segment is coded in the range [-1;+1] so that distribution of the variables is uniform.
Unsupervised	Default value for unsupervised clustering A target free strategy. Only segment frequency is used to encode variables.
TargetMean	Default value for supervised clustering Each value of a continuous input variable is replaced by the mean of the target for the segment the value belongs to. Each category of a nominal input variable is replaced by the mean of the target for this category. In case of a nominal target variable, the mean of the target corresponds to the percentage of positive cases of the target variable for the input variable category.
Natural	Applicable only when all the variables are continuous. This encoding does not transform the input data.
SystemDetermined	Lets the system select the best encoding according to the model parameters. The TargetMean encoding is used for supervised models. Otherwise, variables are encoded using the Unsupervised scheme.
MinMax	Applicable only when all the variables are continuous. This option encodes the categories of the variable in the range [0,1], where 0 corresponds to the minimum value of the variable and 1 corresponds to the maximum value.
MinMaxCentered	Applicable only when all the variables are continuous This option encodes the categories of the variable in the range [-1,1], where -1 corresponds to the minimum value of the variable and 1 corresponds to the maximum value.
StdDevNormalization	Applicable only when all the variables are continuous. This option performs a normalization based on the variable mean and standard deviation. It is computed using the following formula: $\frac{x - Mean}{StdDev}$

⌵ Syntax

Path: Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/ExtraMode

Possible Values	Description
No Extra (default)	Generates no extra output, this is the default behavior (provides only the cluster ID of the current input data)

Possible Values	Description
For K2R	Additionally generates a disjunctive coding of the cluster ID. This may be used to launch SAP Predictive Analytics Modeler - Regression/Classification with as many targets as clusters.
For K2R copy data	Same as above and additionally copy input data set. This is to be used with small data set.
Target Mean	Additionally generates the target mean value of the cluster ID.
Advanced Apply Settings	Expert mode allowing the user to selectively choose the outputs of interest.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/CrossStats`

Values	Description
disabled	Disables cross statistics. This may be useful when there are a lot of input variables : this indeed speeds SAP Predictive Analytics Modeler - Segmentation/Clustering up and reduces memory load as well as the size of the saved model. As a drawback, clusters debriefing in JNI/CORBA client is no more available. KI, KR and basic clusters statistics are still available.
enabled (default)	Enables cross statistics

7.3.3.3.1.1 EnginesConfiguration

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/EnginesConfiguration/ClustersCountRangePolicy`

Specifies how clusters count range should be configured for each target.

Value	Description
Shared	All the targets use the shared clusters count specification (default).
Custom	Each target will specify its own clusters count range, the initial values being taken from the shared configuration section.
OverwritableCustom	This mode differs from the previous one by the fact that the specific configuration is overwritten by the shared one any time the latter happens to change (provided that both are not altered at the same time).

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/EnginesConfiguration/Kxen.SharedEngineConfiguration/ClustersCountSpec`

This folder contains the configuration information that is shared by engines.

Parameter	Description
Min	Lower cluster number
Max	Upper cluster number
Enumeration	Reserved for future use. It will allow specifying clusters count as a set of custom values instead of a range.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/EnginesConfiguration/ByTargets`

The location for configuration information in a per-target basis.

Parameter	Description	Values
<code><target_name> <clusterId>/ClustersCountSpec</code>	This is the place where clusters count range customization can be taken in a per-target basis provided that the clusters count range policy has been set either to <code>Custom</code> or <code>OverwritableCustom</code> . It can be a <code>Mix/Max</code> range or an enumeration. Currently only the <code>Mix/Max</code> range is available.	<ul style="list-style-type: none"> Min: lower clusters number Max: upper clusters number Enumeration: reserved for future use. It will allow specifying clusters count as a set of custom values instead of a range.
<code><target_name> <clusterId>/Engines/<Engine_index></code>	This section is read-only in the current implementation and will be opened in the future to allow some fine tunings such as the distance, encoding strategy, ... to be used by a given engine.	<ul style="list-style-type: none"> Not applicable

7.3.3.3.1.2 ApplySettings

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/ApplySettings`

Common ApplySettings

The table below describes the Apply Settings which are common to both supervised and unsupervised modes.

Parameter	Access	Description	Values
DisjunctiveTopRankNodeId	Read-write	Allows adding to the output file the disjunctive coding of the clusters.	<ul style="list-style-type: none"> • true • false(default value)
OutlierFlag		(no longer used)	
PredictedNodeIdDistances	Read-write	Allows adding to the output file the distance to the various clusters of each observation.	<ul style="list-style-type: none"> • None(default value): no distance will be produced. • All: the distance from each of the clusters will be produced. • Individual: a set of clusters of interest can be selected (inserted).
PredictedNodeIdProbabilities	Read-write	Allows adding to the output file the probability of each observation to belong to the various clusters.	<ul style="list-style-type: none"> • None(default value): no probability will be produced. • All: the probability for each of the clusters will be produced. • Individual: a set of clusters of interest can be selected (inserted).
PredictedRankDistances	Read-write	Allows adding to the output file the distances of each observation from the nearest clusters.	<ul style="list-style-type: none"> • None(default value): no distance will be produced. • All: the distance from each of the clusters will be produced. • Individual: a set of clusters of interest can be selected (inserted).
PredictedRankNodeId	Read-write	Allows adding to the output file the indices of the nearest clusters for each observation.	<ul style="list-style-type: none"> • None: no cluster index will be produced. • All: all the indices will be generated, sorted (increasingly) according to their associated distance • Individual(default value): the requested number of nearest cluster indices is produced. This value has to be inserted.
PredictedRankNodeName	Read-write	Allows adding to the output file the names (instead of the indices) of the nearest clusters for each observation.	<ul style="list-style-type: none"> • None(default value): no cluster name will be produced. • All: all the cluster names will be generated, sorted (increasingly) according to their associated distance. • Individual: the requested number of nearest cluster names is produced. This value has to be inserted.

Parameter	Access	Description	Values
PredictedRankProbabilities	Read-write	Allows adding to the output file the probabilities that the observation belongs to each of the nearest clusters.	<ul style="list-style-type: none"> None(default value): the option is deactivated. All: all the probabilities will be generated, from the highest to the lowest. Individual: the requested number of highest probabilities is produced. This value has to be inserted.

Specific Apply Settings for Unsupervised Mode

Description of the Default Values

≡, Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/ApplySettings/UnSupervised/Default`

Parameter	Access	Description	Possible Values
Constants	Read-write	Allows to generate user-defined constants in the output. Each direct subnode is named after a user-defined constant.	<ul style="list-style-type: none"> <code><Constant_Name>/<OutVarName></code> [read-write]: the name of the constant <code><Constant_Name>/<Value></code> [read-write]: the constant value <code><Constant_Name>/Storage:</code> the constant storage.
Inputs	Read-write	Allows you to add to the output file one or more input variables from the data set.	User-dependent
Weight	Read-write	Allows you to add to the output file the weight variable if it had been set during the variable selection of the model.	User-dependent

Specific Apply Settings for Supervised Mode

≡, Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/ApplySettings/Supervised`

- Apply Settings for Continuous and Ordinal Targets

Parameter	Access	Description	Value
<Target_Name>	Read-only	It extends the common set of flags.	TargetMean (read-write): a boolean value type which allows adding to the output file the mean of the target for the cluster containing the observation. false is the default value.

- Apply Settings For Nominal targets

Parameter	Access	Description	Value
<Target_Name>	Read-only	It extends the common set of flags.	<ul style="list-style-type: none"> • OutlierFlag: [Not Implemented] • OutputCosts: [Not Implemented] • ProfitMatrix: [Not Implemented] • RankType: [Not Implemented] • TargetMean (read-write): a boolean value type which allows adding to the output file the proportion of the least frequent category of the target variable (key category) in the cluster containing the current observation.

7.3.3.3.2 Results

⌘ Syntax

Path: Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Results/
<Target_Name>

This section describes the results of SAP Predictive Analytics Modeler - Segmentation/Clustering. Under this folder there is a subfolder for each target variable. SAP Predictive Analytics Modeler - Segmentation/Clustering can indeed run several models simultaneously, that is one per target variable. In multi-engines mode this folder will hold the results for the winning engine only. Results for all the engines can then be found in the AdvancedResults folder. All parameters are in read-only mode.

Parameter	Description
NbCentroids	The number of clusters requested by the user.
EffNbCentroids	The number of clusters computed by the KMeans engine.
TargetEstimator	The name of the variable used for the cluster index prediction (also know as the target estimator in supervised mode).
Clusters (folder)	A folder where all the information about clusters may be found. It contains one sub-folder for each cluster.

Parameter	Description
Metrics (folder)	Contains the quality metrics for the found segmentation.
Overlapp (only in SQL mode)	The off diagonal percentage of the confusion matrix between covers. More information on covers is available under the 'Clusters' parameters, below.
GazFrequency (only in SQL mode)	The percentage of input data that are not assigned to a cluster.
Index	The engine index within the set of engines associated with this target.

≡ Syntax

Path: Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Results/
<Target_Name>/Clusters

Parameter	Description	Values
Id	The id of the cluster.	
Coordinates (folder)	The folder where the coordinates of clusters are stored. This folder contains one sub-folder per dimension.	
Frequency	The percentage of population gathered in the cluster.	
TargetMean	The mean value of the target for data assigned to the cluster. The target mean is the percentage of <label> in the cluster, where <label> is the least frequent category.	<p>In classification case (binary target) the encoding of the target is as follows:</p> <ul style="list-style-type: none"> • 1: for the least frequent category • 0: for the other
WeighedCount	The weight of the cluster.	
TargetStdDev	Represents the standard deviation of the target in this cluster, when the segmentation has been supervised by a continuous target.	
Metrics (folder)	Standard clustering metrics for this cluster.	
Cover (only in SQL mode)	A folder where part of the cluster SQL formula is stored. A cover is an SQL formula made of a conjunction (that is, a series of 'AND') of basic SQL statements such as "variable i in ['value1', 'value2']" for nominal variables and "variable i < value1 AND variable i > value2" for continuous variables.	
ANDNOT (only in SQL mode)	A folder that contains the rest of the information necessary to build the cluster SQL formula. The folder contains the ids of the covers to subtract from this cover to obtain the cluster formula as shown in the following example: say 2 is the current cover index and { 3, 4 } is the set of indices found in the ANDNOT folder.	<p>The cluster expression is built as:</p> <pre>Cluster = Cover2 AND NOT (Cover3 OR Cover 4) = Cover2 AND NOT Cover3 AND NOT Cover4</pre>

Parameter	Description	Values
SQL	The cluster SQL expression. Note - only build-time detected overlaps are expressed using exclusions (that is, AND NOT) in the generated SQL. Anyway, at apply time overlaps are handled by testing clusters according to their (increasing) KI.	

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Results/<Target_Name>/Clusters/Coordinates`

Parameter	Description	Values
Value	It is the coordinate of the centroid.	
KL	When used in JNI/CORBA interface to debrief the clusters, it is the Kullback-Leibler divergence between cluster distribution for this dimension (that is, this input variable) and population distribution. This distance is used in JNI/CORBA interface to debrief clusters.	<p>Given a cluster and a dimension, it is computed using the following formula:</p> $\sum_{i=0}^N p_i \log \left(\frac{p_i}{q_i} \right)$ <p>where :</p> <ul style="list-style-type: none"> • N is the number of categories for this dimension, • p_i ((resp. q_i) is the proportion of the ith category (of the concerned dimension) within the cluster (resp. the whole population). The proportion of a given category represents its weight over the total weight of the concerned set (either cluster or global population).

Parameter	Description	Values
ChiSquare	<p>It is another divergence measure between cluster distribution and population distribution for this dimension. More precisely, its value represents the degree of confidence one can have in stating that the two distributions are significantly different.</p> <p>Unlike the KL, it does not provide a measure of how far these distributions are from each other. Instead it allows us to be sure that the two distributions ARE different when we get a value near to 1. Thus we can consider the concerned dimension as really discriminative for this cluster. On the contrary, a value of 0 suggests that both distributions may be identical for the dimension, but it cannot be stated for sure. This indicator can only be of real use when its value nears 1. It is not used in the JNI/CORBA interface for debriefing.</p>	

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Results/<Target_Name>/Clusters/Cover`

Parameter	Description
Cluster	The index of the cover. It must be equal to the Id above.
P	Cover's statistics for internal use only.
AnPn	Cover's statistics for internal use only.
AnPp	Cover's statistics for internal use only.
ApPn	Cover's statistics for internal use only.
ApPp	Cover's statistics for internal use only.
KI	Cover's statistics for internal use only.
Confidence	Cover's statistics for internal use only.
Covering	Cover's statistics for internal use only.
Operator (folder)	See explanation below.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SmartSegmenter/Parameters/Results/<Target_Name>/Clusters/Cover/Operator`

The cover SQL formula is stored in the `Operator` subfolder, and in its subfolder `Ranges`. This folder contains one subfolder for each explanatory variable. Each subfolder contains the following information:

Parameter	Description
<code>Values</code>	(when the variable is nominal/ordinal) The set of categories of data assigned to the cluster.
<code>Min</code>	(when the variable is continuous) The minimum of the values for data assigned to the cluster.
<code>MinEqual</code>	(when the variable is continuous) A boolean that specifies whether the Min value is included in the range.
<code>Max</code>	(when the variable is continuous) The maximum of the values for data assigned to the cluster.
<code>MaxEqual</code>	(when the variable is continuous) A boolean that specifies whether the Max value is included in the range.

Advanced Results

This is the location for all the (intermediate) results when the multi-engines mode is enabled. The results for the winning engine are additionally saved in the `Results` folder. Under this folder there is a sub folder for each target variable containing in its turn a sub-folder for each engine. Each of these second level sub-directories has the same layout as the `Results` folder.

7.3.3.4 Kxen.ConsistentCoder

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.ConsistentCoder`

SAP Predictive Analytics Modeler - Data Encoding is a data preparation transform building consistent (robust) coding scheme for any attribute belonging to a training data set containing a business question (specific target variable to analyze). Each nominal attribute possible value is:

- either discarded as non consistent,
- or coded as a number for later use by subsequent transforms.

The attributes may also be called variables, whereas their possible values are sometimes referred to as categories.

SAP Predictive Analytics Modeler - Data Encoding brings Intelligence to any OLAP system (IOLAP™) through a ranking of the variables based on their robust information to explain a business question. SAP Predictive Analytics Modeler - Data Encoding processes on both estimation and validation sets in a single pass. SAP Predictive Analytics Modeler - Data Encoding finds a robust (consistent) encoding to nominal variables in order to be used with numerical algorithms. SAP Predictive Analytics has refined techniques that have been used for years in this field. The strength of SAP Predictive Analytics Modeler - Data Encoding lies in the fact that it can

dialog with the following transform in order to adapt its encoding scheme. Nominal variable encoding schemes can follow two main paths :

- coding based on disjunctive Boolean coding: (0,1) or (-1, 1)
- coding based on the average of the target value.

SAP Predictive Analytics Modeler - Data Encoding belongs more to the second category. When dealing with a nominal variable, it first computes the statistics associated with each category of this nominal attribute.

For example: for a nominal variable called 'Color' with three possible categories: 'Red', 'Blue' and 'Green', SAP Predictive Analytics Modeler - Data Encoding first computes the average of the target for each of these categories. When the target is a continuous variable, the average of the target for each category is a straightforward computation. When the target is a nominal variable, the user can associate a cost (profit) for the different classes.

The nominal variable coding is based on the target average for each categories.

Let's go back to our 'Color' example, and let's assume that the target average for each color given by the following table.

Category	Target Average
Red	.75
Green	.35
Blue	.50

Then, the most simple coding scheme is to encode each category with its target average. But this technique has some drawbacks.

- First you can lose some information when two categories have the same target average (or very close).
- Second, for the target average to have any meaning at all, you must have enough cases for the category.

To counter the first issue, SAP Predictive Analytics Modeler - Data Encoding allows the user to code the category not directly with the target average but with the rank of the sorted target averages as shown in the next table:

Category	Target average	Rank
Red	.75	3
Green	.35	1
Blue	.50	2

In this scheme, each category is coded as the rank. To counter the second issue, SAP Predictive Analytics Modeler - Data Encoding automatically searches the minimum number of cases for each category needed to keep this category. All categories that are not represented enough into the data base is associated with a miscellaneous class (whose default name is 'KxOther'). This search is done using SRM (Structural Risk Minimization) principle.

7.3.3.4.1 Parameters

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.ConsistentCoder/Parameters`

This section describes the SAP Predictive Analytics Modeler - Data Encoding components parameters which can be found under the 'Parameters' section of the component.

Parameter	Access	Description	Values
<code>Compress</code>	<ul style="list-style-type: none">• Read-write• Read-only (when the transform is in ready state)	This variable can be used to deactivate the compression of SAP Predictive Analytics Modeler - Data Encoding, by setting its value to <code>false</code> .	<ul style="list-style-type: none">• <code>true</code> [Default]• <code>false</code>
<code>UseKxOther2</code>			<ul style="list-style-type: none">• <code>true</code> (default and highly recommended)• <code>false</code>
<code>ExtraMode</code>	<ul style="list-style-type: none">• Read-write• Read-only (when the transform is in ready state)	This parameter belongs to the set ['No Extra', 'K2R Coding', 'K2S Coding']. This value is used when SAP Predictive Analytics Modeler - Data Encoding is used alone in a model. This mode can be used to store back the results of the coding into a space.	<ul style="list-style-type: none">• No Extra [Default]• K2R Coding• K2S Coding• K2S Unsupervised• KSVM Coding

7.3.3.4.2 Results

Most of the information generated by SAP Predictive Analytics Modeler - Data Encoding is stored into the current protocol original nominal variables 'Statistics' section. SAP Predictive Analytics Modeler - Data Encoding changes the original categories by introducing the 'KxOther' category if needed. After SAP Predictive Analytics Modeler - Data Encoding has been trained, the original dictionaries are smaller.

7.3.3.5 Kxen.SocialNetwork

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork`

SAP Predictive Analytics Social is an automatic data preparation transform that extracts and uses implicit structural relational information stored in different kinds of data sets, and thus improves model decisions and prediction capacities. The user configures the loading module and graph filters. The configuration is made through specifying parameters in the SAP Predictive Analytics Social parameter tree. SAP Predictive Analytics

Social will use these loading specifications to create a graph set during the learning process, and store them in the model. Graphs are now available for variable computation.

7.3.3.5.1 Parameters

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters`

Parameter	Description	Value
ExtraMode	Extra mode of the current transform	<ul style="list-style-type: none"> Default_Mode (default)
LoadSettings	Folder contains all the parameters used to load the graphs.	
ApplySettings	This folder contains the parameters used for applying the model.	

7.3.3.5.1.1 LoadSettings

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings`

The `LoadSettings` folder contains all the parameters used to load the graphs.

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/GraphOutputDir`

Used for debug purposes. It indicates the path where generated graphs will be dumped in DOT format. If left empty, nothing is done. Default value is empty.

☞ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/NodeSettings_<Repository>`

It contains the parameters for providing descriptive attributes and identifiers conversion.

Parameter	Description
IndirectionTableDataSet	Indicates the location of the mapping information between the identifier used to build the graphs and the identifier used in the node datasets.

Parameter	Description
IndirectionTableIdColumnName	Indicates the column in the identifier conversion table that contains the node identifier, such as the line number. Any node that has no match in the node data set for the value of the node column will be declared as 'off-net'.
IndirectionTableNodeColumnName	Indicates the column in the node data set that contains the entity identifier, such as the customer identifier.
NodeTableDataSet	Name of the table containing the descriptive attributes of the entity, for example the customer. All columns contained in the node data set may be used in queries or aggregates. The selected columns are indicated under the sub-folder <code>NodeProperties</code> .
NodeTableIdColumn	Name of the column containing the nodes identifiers in the descriptive attribute data set. This column will be used to join attributes with the nodes existing in the graph.
MaxValueCount	Maximum number of distinct categories to remember for discrete variables. The N most frequent categories will be kept, while the others will be represented as a <code>KxOther</code> category.
NodeProperties	The user can use this parameter to specify the variables to be used to decorate the node; the <code>NodeProperties</code> parameter changes to a directory containing the variables inserted by the user.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/GraphFilters`

The `GraphFilters` sub-folder is used to insert parameter templates in the graphs created from a single table containing the relational data. Every inserted template contains parameters for the concerned graph (type, column to filter, etc...):

Parameter	Description	Possible Values
<code>GraphType</code>		Directed/Undirected/Bipartite.
<code>StartDate</code>		The begin of date filter interval.
<code>EndDate</code>		The end of date filter interval.
<code>MinimumEventCount/SumForLink</code>	(for Contact/Transaction mode only) Keep links if more events than the specified threshold have been met.	
<code>MaximumEventCount/SumForLink</code>	(for Contact/Transaction mode only) Keep links if less events than the specified threshold have been met.	

Parameter	Description	Possible Values
First/ SecondMaximumWeightSum	Allows the user to filter nodes with a threshold value on the sum of its weighted degree, for the first/second population.	
NumberOfNeighbors	(for K-Nearest neighbors mode only) The maximum number of neighbors for each node. It is the K parameter.	
DistanceColumnName	(for K-Nearest neighbors mode only) The name of the column that provides distance between the two nodes.	
DeriveFirst and DerivedSecond	(for Bipartite graphs) Indicates the threshold used to generate the unipartite graph. If no value is set, no additional graph is generated.	
PairingGraphName	Indicates the name of the other associated graph to generate the pairing graph.	
Node1Column/Node2Column	Every graph filter can specify its own column containing node identifiers. If no value is set, the graph filter inherits columns set at the upper loadsettings level.	
NodeSet	Can be used to force the graph to store its nodes in a specific node repository.	<ul style="list-style-type: none"> • First: source node • Second: target node
ColumnFilter	<p>Can be used to insert several column names to filter the event data set. The generated graph will only contain links that match all the inserted conditions. Each column filter can be set the following way:</p> <ol style="list-style-type: none"> 1. FilterMaximum/FilterMinimum: cannot be set if Filtervalue is set. Used to filter values between two bounds. 2. FilterValues/FilterNotValues: can be used to filter on a precise discrete values list (include or exclude). 	

7.3.3.5.1.1.1 PostProcessing

☰ Syntax

```
Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing
```


In Transaction mode, SAP Predictive Analytics Social offers the possibility to process some post-loading operations on graphs, by inserting a template in the `PostProcessing` sub-folder. After selecting the `FromModel` option for the `GraphLoadType` parameter, the `Ksn.BipartiteGraphProjection` sub-folder is created.

Since version 6.0.0, it is possible to set a priority order to all graph post-processings. For instance it can be important to force the mega-hub filter to be launched before the community detection. The priority can be set using the `priorityLevel` common to all post-processings.

By default, post-processes are executed in this order:

1. Mega-hub filtering
2. Bipartite graph projection
3. Community detection
4. Node pairing

All post-processing parameters must be specified after the graph specifications and before the model learning stage.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/Node1Column`

The name of the column containing the first node identifier (input node if directed graph is built).

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/Node2Column`

The name of the column containing the second node identifier (output node if directed graph is built).

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/DateColumn`

The name of the column containing the date of the events (optional, may be left blank).

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/KSN.BipartiteGraphProjection/<Graph_Name>`

Bipartite projection is used to transform a bipartite graph into a unipartite graph in which nodes from a same population are connected if they share a certain amount of relevant neighbors. There is one folder for each projected graph.

Parameter	Description
<code>SourceGraphName</code>	Name of the bipartite source graph

Parameter	Description
TargetGraphName	Name of the projected graph. If no name is specified, an automatic name is generated.
Population	Population used for the graph projection Values: <ul style="list-style-type: none"> • First • Second
MinimumSupport	Number of minimum common neighbors to build a link in the projected graph
Weight	Similarity measure used to weight links in projected graphs Values: <ul style="list-style-type: none"> • Support • Jaccard • Independence Probability
Filter/MinimumConfidence	Minimum confidence to keep a link in the projected graph
Filter/MinimumKi	Minimum KI to keep a link in the projected graph
Filter/Directed	Produces directed links in projected graphs Values: <ul style="list-style-type: none"> • true • false (default)

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/KSN.MegaHubFilter/<Graph_Name>`

Mega-hub filtering can be activated for a given graph to filter high connected nodes. This folder contains one sub-folder for each graph for which the mega-hub filtering has been activated. The parameters corresponding to each graph are stored in these sub-folders.

Parameter	Description
GraphName	The graph in which the mega-hubs must be filtered.
MethodId	Values: <ul style="list-style-type: none"> • 0: manual threshold method • 1: n-sigma method (default).
StdDevFactor	If MethodId = 1, this parameter can be used to set the sigma parameter. Default value: 4

Parameter	Description
DistributionStrip	If MethodId = 0, this parameter can be used to set the manual mega-hub threshold. Default value: 50000
UserThreshold	
Population	Sets the population to be used to filter the mega-hubs on bi-partite graphs.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/KSN.CommunityDetection/<Graph_Name>`

The Community Detection algorithm can be activated for a given graph. This folder contains one sub-folder for each graph for which the community detection has been activated.

Parameter	Description
GraphName	The graph in which the communities should be detected.
MaxIteration	Maximum number of iterations of the algorithm
EpsilonValue	The stop condition for the algorithm. The detection stops if there is no modularity gain bigger than EpsilonValue.
RandomSeed	Used to initialize the random number generator. Two community detection processes on the exact same graph and same RandomSeed parameter option will always find the same community segmentation. The random value will generate a random node selection during the community detection. This is an optional parameter. If set to 0, nodes are picked in the order of their IDs. Default value: 0
InitFromGraph	Name of the graph to use to initialize the community detection with communities from another graph.
IgnoreSelfLoop	If set to <code>true</code> , this option will create smaller hierarchical graphs. Values: <ul style="list-style-type: none"> <code>true</code> <code>false</code> (default)

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/PostProcessing/KSN.NodePairing/Pairing<n>`

Node pairing will create a new graph where nodes are linked if they share a certain amount/ratio of common neighbors in a graph. It can be activated for one or two graphs. This folder contains one sub-folder for each pairing requested.

Parameter	Description
FirstGraphName	Name of the first graph to pair
SecondGraphName	Name of the second graph to pair Note - if FirstGraphName and SecondgraphName are the same, the graph will be paired with itself.
PairingGraphName	Name of the output graph containing the pairing Value: string. If let empty, an automatic name is generated as follows: np_<FirstGraphName>_<SecondGraphName>.
CountThreshold	Minimum number of common neighbors needed to pair two nodes Default value: 4
RatioThreshold	Minimum ratio of common neighbors needed to pair two nodes Default value: 0.5
WeightedRatio	If activated, will compute weighted similarity measures, taking weights into account. Values: <ul style="list-style-type: none"> • true • false (default)
IncludeCountGraph	If activated, will generate a graph with the "count" similarity measure. Values: <ul style="list-style-type: none"> • true • false (default)
TopN	Maximum number of neighbors to pair for each node. Keep the N nodes with the highest weights as determined by the parameter PairingType. Default value:0 (which means keeping all nodes)

Parameter	Description
PairingType	<p>Similarity measure used to weight the links of the paring graph</p> <p>Values:</p> <ul style="list-style-type: none"> • Ratio (default) • Count • Jaccard • Independence Ratio • Confidence • Clustering

7.3.3.5.1.1.2 FromModelLoadSettings

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/LoadSettings/FromModelLoadSettings`

This folder contains all the parameters required to specify models, including saved models to be imported.

Parameter	Description	Values
StoreClass	Contains the class of the store (file directory or ODBC source).	<ul style="list-style-type: none"> • <code>Kxen.ODBCStore</code> • <code>Kxen.FileStore</code>
StoreName	Name of the store containing the model	
ModelName	Actual name of the model	String
Version	The model version	

7.3.3.5.1.2 GraphApplySettings

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/ApplySettings/GraphApplySettings/<Graph_Name>`

There is one folder for each graph generated. The names of the community graphs created for a graph defined by the user are suffixed `_cm_lvl_<level_number>`.

Parameter	Description
CommunityCount	This folder allows the user to insert all the discrete variables for which they want to compute the count in the community. This generates as many columns as there are categories in the target variable.
CommunityIndex	Index of the community
CommunityLinks	Lists all the links inside the community of the input node. This can be used to extract the subgraph of nodes and links inside a community.
CommunityMean	Allows the user to insert all the continuous variables (age, capital_gain...) for which they want to compute the mean on the community.
CommunityNode	Only for community graphs. Gives a node ID inside the input community (ID) (used for internal display).
CommunityOffNetCount	Count/ratio of offnet nodes in the community
CommunityOffNetRatio	
CommunityRatio	Allows the user to insert all the discrete variables for which they want to compute the modes in the community. This generates as many columns as there are categories in the target variable.
CommunityRole	Role of the node inside the community Values: <ul style="list-style-type: none"> • <code>Textual</code>: output roles as a textual value • <code>Integer</code>: output roles as an integer value • <code>None</code>: deactivated
CommunitySize	Outputs the size of the community of the input node Values: <ul style="list-style-type: none"> • <code>true</code> (default) • <code>false</code>
CommunityVariance	This folder allows the user to insert all the continuous variables (age, capital_gain...) for which they want to compute the variance on the community
Count	This folder allows the user to insert all the discrete variables for which they want to compute the modes in the first circle (set of direct neighbors). This generates as many columns as there are categories in the target variable, containing the count for this category.
Degree	Generates degree-related outputs Values: <ul style="list-style-type: none"> • <code>true</code> (default) • <code>false</code>
Describe	Activates the described mode. Values: <ul style="list-style-type: none"> • <code>true</code> • <code>false</code> (default)

Parameter	Description
IOSegment	Class of the node between sink,source and repeater. Values: <ul style="list-style-type: none"> '0' / 'REPEATER': incoming/outgoing link count is balanced '1' / 'SOURCE': mostly outgoing links '2' / 'SINK': mostly incoming links
InfluenceReach	This folder can be used to insert a template parameter in the Social ApplySettings Influence Reach node. The label name used for the inserted template has no importance.
Leader	Activates the centrality output
Mean	Allows the user to insert all the continuous variables (age, capital_gain...) for which they want to compute the mean on the first circle
Mode	Allows the user to insert all the discrete variables for which they want to compute the modes in the first circle. This generates as many columns as the number of categories in the target variable.
Neighbors	Activates the neighbors multiline mode
OffNetCount	Activate the offnet statistics outputs
OffnetRatio	
Profile	Same as the Mode parameter but using ratios instead of counts
Ratio	This folder allows the user to insert all the discrete variables for which they want to compute the ratio of each of the categories in the first circle (set of direct neighbors). This generates as many columns as there are categories in the target variable, containing the ratio for this category.
Recommendation	This folder can be used to Insert a template parameter in the Social ApplySettings Recommendation node. This must be done in the bipartite graph apply settings.
SPActivation	Spreading activation, or graph diffusion, is used to spread a value into the graph. The diffusion is initiated by labeling all the nodes with weights or "activation" and then iteratively propagating or "spreading" that activation out to other nodes linked to the source nodes. The result of the apply is the weight of each nodes after the diffusion process.
Triangle	Computes the number of triangles the input node is a part of. If two neighbors of a node are themselves connected, they are forming a triangle. Values: <ul style="list-style-type: none"> true false

☰ Syntax

Path: Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/ApplySettings/GraphApplySettings/<Graph_Name>/InfluenceReach/<Label>

The Influence Reach detects the collection of all Influence Cascades starting from a given node. It can be used to detect how many nodes have adopted a given behavior after a source node did. The graph must contain a date variable in its attributes that correspond to the time of the activation (behavior adoption, churn date...).

Parameter	Description
VarName	Name of the variable containing the activation date information. The variable must have a date description.
StartDate	Start of the time frame to observe the activation diffusion. Nodes that do not match the time frame are not taken into account. Can be left blank.
EndDate	End of the time frame to observe the activation diffusion. Nodes that do not match the time frame are not taken into account. Can be left blank.
DeltaMin	Minimum delta of time between two nodes to observe the activation diffusion. Can be left blank.
ActivatedOnly	If activated, will only compute cascade size if the node itself is activated.

Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/ApplySettings/GraphApplySettings/<Graph_Name>/Recommendation`

Parameter	Description
FilterPurchased	<p>If set to <code>true</code>, will exclude already purchased items from the recommendation set; that is, the items the user is connected to in the bipartite graph.</p> <p>Values:</p> <ul style="list-style-type: none"> • <code>true</code> • <code>false</code>
ProjectedGraphName	<p>Name of the source graph containing the recommendation rules</p> <p>Value:</p> <ul style="list-style-type: none"> • string: must be a valid projected graph name
SortCriteria	<p>Sort criteria for the items</p> <p>Values:</p> <ul style="list-style-type: none"> • Support • Confidence • KI

Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Parameters/ApplySettings/GraphApplySettings/<Graph_Name>/SPActivation`

Parameter	Description
<code>VarName</code>	Name of an available descriptive attribute (nominal/continuous) that will be used as a seed weight for each node
<code>SpreadingFactor</code>	Decay factor of the diffusion, part of the score that will be propagated Values: <ul style="list-style-type: none"> 0.75 (default) Any float value in [0 ; 1]
<code>MaxIteration</code>	Maximum number of iterations of the diffusion Values: <ul style="list-style-type: none"> 10 (default) Any integer >1
<code>ActivationThreshold</code>	Nodes with weights above this threshold will be considered as activated. Only activated nodes will spread weights to their neighbors. Values: <ul style="list-style-type: none"> 0.1 (default) any float value in [0 ; 1]

7.3.3.5.2 Results

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Results`

In the `Results` folder, the following parameters are available:

Parameter	Description
<code>NumberOfGraphs</code>	Shows the number of generated graphs
<code>NodesInFirstRepo</code>	Number of unique nodes (from all graph) stored in the first node repository
<code>NodesInSecondRepo</code>	Number of unique nodes (from all graph) stored in the second node repository (for bipartite graphs)
<code>NodesAttributes</code> (folder)	
<code>AvailableGraphs</code> (folder)	Stores graph-specific information for each generated graph
<code>DeriveFrom</code>	This parameter is available with a derived graph of a bipartite graph, or a node pairing graph. It indicates if the graph is a result of a post-processing operation on another graph.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Results/AvailableGraphs`

Parameter	Description	Values
Type	The type of the graph.	<ul style="list-style-type: none"> Directed Undirected Bipartite
Nodeset	Indicates in what node repository nodes are stored. For bipartite graphs, it indicates the node repository of the first population.	<ul style="list-style-type: none"> First Second
Nodes	The number of edges (or vertices) in the concerned graph.	
NodesFirst	The number of nodes in the first population.	
NodesSecond	The number of nodes in the second population.	
OffNetNodesFirst	The number of off-net nodes in the first population.	
OffNetNodesSecond	The number of off-net nodes in the second population.	
Edges	The number of links (or vertices) stored in the concerned graph.	
Density	The density of the network.	
RemovedNodes	Remembers the nodes ignored during the community detection if the AutoModularityFilter parameter was used (see Community Detection Post Processing).	
In/Out/ UndMedianDegree	Gives the median value for the node incoming/outgoing/undirected degree distribution, depending on the graph type.	
In/ OutPowerLawExponent	Indicates the power law exponent of the node degree distribution.	
In/ OutPowerLawExponent Fit	of fit indicator for the node degree distribution detection.	
MegaHubs (folder)		It contains parameters which are available only when the model is live but not saved.
Communities (folder)		It provides some information on the community inside the graph.

Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Results/AvailableGraphs/MegaHubs`

Parameter	Description
Threshold	Detected degree threshold for mega-hubs.
HubsCount	The number of detected mega-hubs.
NodeList	The list of mega-hubs, with their respective degree.

Syntax

Path: `Protocols/Default/Transforms/Kxen.SocialNetwork/Results/AvailableGraphs/Communities`

Parameter	Description
<code>NumberOfCommunities</code>	Number of detected communities
<code>Modularity</code>	Value of the final modularity, a goodness indicator of the community partition
<code>Intra/Inter-LinksMedian</code>	Median value of the intra/inter links distribution for all the nodes
<code>SumOfWeights</code>	Weighted sum of all the links in the graph. This value is needed for several community detection algorithm or modularity optimization
<code>CommunitySizeDistribution</code>	Stores a list of couple of integers that describe the distribution of community size in the given graph
<code>In/OutPowerLawExponentFit</code>	Indicator for the node degree distribution detection

7.3.3.6 Kxen.DateCoder

Syntax

Path: `Protocols/Default/Transforms/Kxen.DateCoder`

Date Coder is an automatic data preparation transform that extracts date information from a date or datetime input variable. This component is automatically inserted by SAP Predictive Analytics Modeler - Data Encoding if one of the input variables is a date or a datetime variable.

7.3.3.6.1 Variables Used

This section of the parameter tree contains all the date and datetime variables used by the Date Coder component.

Date variables are broken down into 7 dateparts:

Datepart	Description	Values	Generated Variable Name
Day of week	day of week according to the ISO disposition	Monday=0 and Sunday=6	<code><OriginalVariableName>_DoW</code>
Day of month	day of month	1 to 31	<code><OriginalVariableName>_DoM</code>

Day of year	day of the current year	1 to 366	<OriginalVariableName>_DoY
Month of quarter	month of the quarter	<ul style="list-style-type: none"> January, April, July and October = 1 February, May, August and November = 2 March, June, September and December = 3 	<OriginalVariableName>_MoQ
Month of year	month	1 to 12	<OriginalVariableName>_M
Year	year	four digits of the year	<OriginalVariableName>_Y
Quarter	quarter of the year	<ul style="list-style-type: none"> January to March = 1 April to June = 2 July to September = 3 October to December = 4 	<OriginalVariableName>_Q

On top of the 7 dateparts above, Datetime variables are broken down into 3 more dateparts:

Datepart	Generated Variable Name
Hour	<OriginalVariableName>_H
Minute	<OriginalVariableName>_Mi
Second	<OriginalVariableName>_S

The generated variables are stored in the parameter tree under Protocols > Default > Variables.

Note- all generated variables are ordinal except for 'DayOfYear' and 'µ seconds' which are continuous.

7.3.3.7 Kxen.AssociationRules

≡ Syntax

Path: Protocols/Default/Transforms/Kxen.AssociationRules

SAP Predictive Analytics Modeler - Association Rules generates association rules. Association rules provide clear and useful results, especially for market basket analysis. They bring to light the relations between products or services and immediately suggest appropriate actions. Association rules are used in exploring categorical data, also called items.

The strengths of SAP Predictive Analytics Modeler - Association Rules are:

- to produce clear and understandable results,
- to support unsupervised data mining (no target attribute),
- to explore very large data sets thanks to its ability to first generate rules on parts of the data set before aggregating them (exploration by chunks),
- to generate only the more relevant rules (also called primary rules).

7.3.3.7.1 Parameters

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters`

This section describes the parameters of SAP Predictive Analytics Modeler - Association Rules that can be found under the 'Parameters' section of the component.

Parameter	Access	Description	Values
ExtraMode	read-write	A special flag allowing to set the type of outputs that Association Rules will generate.	<ul style="list-style-type: none"> • No Extra • Optimized by KI • Optimized by Confidence • Full Description • Full Description and Optimized by Confidence • Full Description and Optimized by KI
DateColumnName	Sequence mode only	The column in which the date is stored.	
SequencesMode	Sequence mode only	A flag specifying if the Sequence mode of Association Rules is activated.	<ul style="list-style-type: none"> • true (or 1): means that the Sequence mode is activated, • false (or 0) means that the Sequence mode is deactivated.

Parameter	Access	Description	Values
Transactions (folder)		Used to set the information relative to the Events data source.	
References (folder)		Used to set the information relative to the Reference data source.	TIDColumnName: indicates the name of the reference key variable.
ARulesEngineParameters (folder)		Meaning Association Rules Engine Parameters.	

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/ExtraMode`

Values	Description
No Extra	Generates basic outputs, that is the session key, the ID of the rule used to find the consequent and the consequent itself.
Optimized by KI	Generates basic outputs. If more than one rule give the same consequent for a session, the rule presenting the best KI will be selected.
Optimized by Confidence	Generates basic outputs. If more than one rule give the same consequent for a session, the rule presenting the best Confidence will be selected.
Full Description	Generates the extended outputs, that is the session key, the rule ID, the consequent, the antecedent, the KI, the confidence and the rule support.
Full Description and Optimized by Confidence	Generates the extended outputs. If more than one rule give the same consequent for a session, the rule presenting the best Confidence will be selected.
Full Description and Optimized by KI	Generates the extended outputs. If more than one rule give the same consequent for a session, the rule presenting the best KI will be selected.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/Transactions`

Parameter	Description
LogSpaceName	Indicates SAP Predictive Analytics role set for the Transactions space name.
TIDColumnName	Indicates the name of the transactions key variable.
ItemIDColumnName	Indicates the name of the item variable.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/ARulesEngineParameters`

Association rules has two usages in SAP Predictive Analytics. The first one is as a standalone module used to detect interactions between items that are associated with a common entity: it is used for example to detect rules in market basket analysis linked products that are bought together by the same customer. It is referred to

as "Default Use" below. The second use case is to search for rules that link elements of a single data set one to the other and is used for data quality tasks: it is used for example to detect that when a variable has this value, another variable has a given value in 95% of the times. We will call this use case the "Data quality mode". It is referred to as "Data Quality Mode" below.

≡ Syntax

Path: Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/
ARulesEngineParameters/FPV

Default Use

Parameters	Description	Values
Activated	Indicates whether the FPV algorithm is activated or not.	<ul style="list-style-type: none"> • true(default value) • false
MinimumSupport	Gives the minimum support required for a rule.	<ul style="list-style-type: none"> • 1 (default value) • > 0: required value <p>Note - With a value > 1, we consider the number of sessions. With a value between 0 and 1, we consider a percentage of the number of sessions</p>
MinimumConfidence	Gives the minimum threshold for the confidence of a rule.	<ul style="list-style-type: none"> • 0.5 (default value) • Required value between 0 and 1
MaxLength	Saves the maximum size of a rule. A rule has a minimum size equal to 2: the antecedent plus the consequent.	<ul style="list-style-type: none"> • 4 (default value) • Required value >=2
SearchMethod	Indicates the search method the FPV has to use. When the value equals 1, the Association Rules engine uses the basic FPV algorithm. When the value equals 2, the Association Rules uses a modified FPV algorithm: the rules are generated regarding the frequency order of their consequent item and the memory used is limited.	<ul style="list-style-type: none"> • 2 (default value) • Required value: 1 or 2
SkipDerivedRules	Indicates to FPV to generate only the primary rules and thus skip the derived rules. For instance, given R1 = X => Z and R2 X,Y =>Z two association rules, R1 is called a primary rule and R2 a derived rule.	<ul style="list-style-type: none"> • true (default value) • false
	Note - the option SkipDerivedRules is only available with a SearchMethod parameter equal to 2.	

Parameters	Description	Values
SkipReducibleItemsets	[deactivated] Indicates if FPV has to generate the reducible itemsets. An itemset is non reducible if one of its items cannot be removed without changing the transaction space.	<ul style="list-style-type: none"> • true • false (default value)
TidSet	Indicates the storage method of the transactions associated to an itemset.	<ul style="list-style-type: none"> • 2 (default value) • Required value: or
ChunkSize	Saves the size of the chunks (in number of sessions) used by FPV to import and generate the rules. With a value equal to 0, the chunk strategy will not be used and ALL the sessions will be imported before generating the rules.	<ul style="list-style-type: none"> • 0 (default value) • Required value >=
GuessChunkSize	When set to true, overrides the Chunk-Size parameter to automatically compute the proper chunk size to use in order to process ten chunks.	<ul style="list-style-type: none"> • true • false (default value)
CheckAntecedents	Is used to check if the consequent of a rule is present in the antecedent items.	<ul style="list-style-type: none"> • true (default value) • false

Data Quality Mode

Parameters	Description	Values
DefaultTargetEnable	Indicates whether to generate the rules that have the most frequent item of a group as the consequent part. This parameter should be set to false, because it is useless to deduce the rules used to fill the missing values.	<ul style="list-style-type: none"> • true • false (default value)
UseMissInfo	Indicates that the confidence of the rules will be computed taking into account the missing information for each variable. This parameter is only used when the Association Rules engine is used to replace the missing values.	<ul style="list-style-type: none"> • true • false
AutomaticParameterSettings	Used in the Data Quality Mode of KAR in order to infer what is usually given by the user such as the maximum length, the default support and confidence. The maximum length is computed with respect to the number of columns in order to fight for combinatorial explosion.	<ul style="list-style-type: none"> • true: means that the Quality Data Mode is selected. • false: means the Default Use mode is selected.

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/ARulesEngineParameters/RulesGenerationFilters`

This folder contains the `ConsequentFilters` folder, which contains the following parameters:

Parameter	Description
<code>IncludedList</code>	All rules with a consequent belonging to the list of values inserted below this parameter will be generated. When this list is empty, all rules are generated (with respect to the <code>ExcludedList</code>).
<code>ExcludedList</code>	All rules with a consequent not specified in the list of values below this parameter will be generated. When this list is empty, all rules are generated (with respect to the <code>IncludedList</code>).

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Parameters/ARulesEngineParameters/ApplyActivationOptions`

Parameter	Description
<code>RulesExcludedList</code>	Can be filled by the user with identifiers of rules not to be used to generate recommendations.
<code>ActivatedConsequentsList</code>	Be filled by the user with items. All rules having one of these items as consequent will be kept to generate recommendations.

7.3.3.7.2 Results

☰, Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Results`

This section describes the results of SAP Predictive Analytics Modeler - Association Rules. This can be found under 'Results' section of the component.

Parameter	Description	Values
<code>ItemCategories</code> (folder)	The folder where the global information of the Item variable is stored.	

Parameter	Description	Values
ARulesEngineResults (folder)	The folder where all the information about the rules found is stored.	<ul style="list-style-type: none"> NumberOfItemSets Generated NumberOfFrequent ItemSets NumberOfFrequent Items NumberOfItems NumberOfRules FillingRatio NumberOfTransactions Rules (folder)
TransactionStats (folder)	Contains all the statistics relative to the number of transactions by session.	<ul style="list-style-type: none"> Mean StDev Min Max

⌵ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Results/ItemCategories`

Parameter	Description	Values
Value		
Storage		
KeyLevel		
OrderLevel		
MissingString		
Group		
Description		
ConstantForMissing (folder)		
SpaceName (folder)		
InSpaceName (folder)		
IsVirtualKey		<ul style="list-style-type: none"> true false
UserPriority		
DescriptionSource		

Parameter	Description	Values
UserEnableKxOther		<ul style="list-style-type: none"> true false
UserEnableCompress		<ul style="list-style-type: none"> true false
NominalGroups		
Monotonicity		
UserModulus		
StatForCode		
EstimatorOf		
ClusterOf (folder)		
TargetKey		
SpaceOrigin		<ul style="list-style-type: none"> true false
Translations		
NativeInformation (folder)		
Statistics (folder)		
Extensions		
BasedOn		

≡ Syntax

Path: Protocols/Default/Transforms/Kxen.AssociationRules/Results/ARulesEngineResults

Parameter	Description	Values
NumberOfItemSetsGenerated	The total number of itemsets created during the learning phase.	An integer value.
NumberOfFrequentItemSets	The number of frequent item sets, that is the number of itemsets whose support is superior to the minimum support set by the user.	An integer value.
NumberOfFrequentItems	The number of frequent items, that is the number of items whose the support is superior to the minimum support set by the user.	An integer value.
NumberOfItems	The number of items found in the transaction data set.	An integer value.
NumberOfRules	The number of association rules generated by Association Rules.	An integer value.

Parameter	Description	Values
FillingRatio	The rate of filling of the missing values.	An integer value.
NumberOfTransactions	The number of sessions treated by Association Rules.	An integer value.
Rules (folder)	The folder where all the rules generated are stored.	<ul style="list-style-type: none"> • Antecedent (folder) • Consequent (folder) • Confidence • KI • Lift • AntecedentSupport • ConfidenceSupport • RuleSupport • SequenceSupportPct • SequenceSupportRatio • SequenceConfidence • SequenceKI • SequenceLift • DurationMin • DurationMax • DurationMean

Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Results/ARulesEngineResults/Rules`

Parameter	Description	Values
Antecedent (folder)	The folder where the names of the items composing the antecedent are stored.	
Consequent (folder)	The folder where the name of the item composing the consequent is stored.	
Activated		<ul style="list-style-type: none"> • true • False
Confidence	The confidence of the rule.	
KI	The KI of the rule.	
Lift	The Lift of the Rule.	A value strictly greater than 0.
AntecedentSupport	The support of the rule antecedent.	

Parameter	Description	Values
AntecedentSupportPct		
ConsequentSupport	The support of the rule consequent.	
ConsequentSupportPct		
RuleSupport	The support of the rule	
RuleSupportPct		
SequenceSupportPct [Sequence mode only]	Indicates the absolute sequence support of the rule, that is the number of sessions in which the antecedent occurs before the consequent.	
SequenceSupportPct [Sequence mode only]	Indicates the relative sequence support of the rule.	A real value between 0 and 1.
SequenceSupportRatio [Sequence mode only]	Indicates for a specified rule the percentage of session where the consequent occurs after the antecedent.	
SequenceConfidence [Sequence mode only]	Indicates the rule confidence in the sequence mode.	A real value between 0 and 1.
SequenceKI [Sequence mode only]	Indicates the rule KI in the sequence mode.	A real value between -1 and 1.
SequenceLift [Sequence mode only]	Indicates the rule Lift in the sequence mode.	A value strictly greater than 0.
DurationMin [Sequence mode only]	Indicates the minimum amount of time observed between an antecedent and its consequent.	A value expressed in seconds if the date is in a date or datetime format.
DurationMax [Sequence mode only]	Indicates the maximum amount of time observed between an antecedent and its consequent.	A value expressed in seconds if the date is in a date or datetime format.
DurationMean [Sequence mode only]	Indicates the average amount of time observed between an antecedent and its consequent.	A value expressed in seconds if the date is in a date or datetime format.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.AssociationRules/Results/TransactionsStats`

Parameter	Description
Mean	Indicates the average number of transactions.
StDev	Indicates the standard deviation for the number of transactions.
Min	Indicates the minimum number of transactions encountered.
Max	Indicates the maximum number of transactions encountered.

7.3.3.8 Kxen.EventLog

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.EventLog`

The purpose of SAP Predictive Analytics Explorer - Event Logging is to build a mineable representation of events history. It is not a data mining algorithm but a data preparation transform. All algorithms to perform regression, classification or segmentation only work on a fixed number of columns, but sometimes a customer can be associated with a list of events (the purchase history for example) with different size for every customer, so this list of events must be translated in a number of fixed columns in one way. These type of operations are called pivoting in data mining because they translate information contained in the same column of different lines into different columns on a single line (for a given customer identifier for example). SAP Predictive Analytics Explorer - Event Logging and SAP Predictive Analytics Explorer - Sequence Coding belong to these type of transformations. SAP Predictive Analytics Explorer - Event Logging can be used to represent the history of a customer, or a log history of defects associated with a machine in a network. This component merges static information (coming from a static table) and dynamic information (coming from a log table). The user must have these two tables before using the component. The table containing static information is generally called the "reference" table, and it is associated in the models with the classical data set names or roles such as Training, Estimation, Validation, Test or ApplyIn. The table containing the log of events (sometimes called the "transactions" table) is associated with a name beginning with the name "Events". SAP Predictive Analytics Explorer - Event Logging is said to build coarse grain representations as it summarize the events on different periods of interest; this is done with some information loss. A good example of using SAP Predictive Analytics Explorer - Event Logging is when trying to represent RFA (Recency-Frequency-Amount) views of a customer based in his purchase history.

7.3.3.8.1 Parameters

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.EventLog/Parameters`

This section describes the parameters of SAP Predictive Analytics Explorer - Event Logging that can be found under the 'Parameters' section. All the parameters are read-only when transform is in ready state.

Parameter	Description	Values
Reference (folder)		
Representation (folder)	The variables concerned by each operator are listed under it in the parameter tree.	
InternalStorage	Contains the placeholder that allows us to activate our own internal storage to avoid large memory consumption.	<ul style="list-style-type: none">• Memory• Db• ByChunk(default value)

Parameter	Description	Values
ExtraMode	Indicates the mode in which SAP Predictive Analytics Explorer - Event Logging will be applying the aggregations; changing this will influence the output produced by SAP Predictive Analytics Explorer - Event Logging.	<ul style="list-style-type: none"> No Extra(default value): produces all aggregates, and also outputs its input variables. Output Only Aggregates: outputs only the variables generated by SAP Predictive Analytics Explorer - Event Logging.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.EventLog/Parameters/Reference`

Parameter	Description
IdColumnName	The name of the column containing the identifier of the main object (customer ID, machine ID, session ID) in the reference table. A proper value is mandatory.
DateColumnName	<i>[read-write]</i> : when not empty, this name is the name where the component can find a reference date. The reference date is used to compute aggregates on periods starting at the reference date. In this case the reference date can be different for every line. If this value is left empty, then it is assumed that the user will specify a proper RefDefaultDate which is defaulted to 1-01-01 00:00:00.
Period	<i>[read-write]</i> indicates what is the period used for aggregation (day, week, month, year,...). This value is defaulted to 'week'.
PeriodStartNumber	<p><i>[read-write]</i> indicates the lower bound of the time window. This number must be less than the RefPeriodEndNumber. A minus sign indicates that the user is interested by periods before the reference date. As an example if the period is taken as 'Week' and the RefPeriodStartNumber is -4, the first period of interest will be 4 weeks before the reference date of the line. This value is defaulted to 0.</p> <p>Note - The PeriodStartNumber and the PeriodEndNumber are the bounding limits of the time window, not the number of the first and last period. For example, if you choose "2005-01-10 12:00:00" as reference date and set start and end to be -5 and 0 (with day as unit) your time window will go from -5 days before the reference date up to 0 zero days.</p>

Parameter	Description
PeriodEndNumber	<p><i>[read-write]</i> indicates the upper bound of the time window. This number must be larger than the RefPeriodStartNumber. A minus sign indicates that the user is interest by periods before the reference date. As an example if the period is taken as 'Week' and the RefPeriodEndNumber is +3, the last period of interest will be 3 weeks after the reference date of the line. This value is defaulted to 0.</p> <p>Note - The PeriodStartNumber and the PeriodEndNumber are the bounding limits of the time window, not the number of the first and last period. For example, if you choose "2005-01-10 12:00:00" as reference date and set start and end to be -5 and 0 (with day as unit) your time window will go from -5 days before the reference date up to 0 zero days.</p>
DefaultDate	<p><i>[read-write]</i>: when not empty, this reference date is used either when there is no reference date column or when this value is missing. It must be noted that if the user specify both a ReferenceDateColumnName and a RefDefaultDate, only the later is taken is taken under consideration.</p>
CyclicPeriod	<p><i>[read-write]</i>: indicates whether the periods are cyclic, and the type if they are (possible values: NotCyclic (default), HourOfDay, DayOfWeek, DayOfMonth, MonthOfYear, QuarterOfYear).</p>
PeriodLength	<p><i>[read-write]</i>: as the label indicates, it is the length of one period.</p>

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.EventLog/Parameters/Representation`

Parameter	Description
Count	Indicates the number of states contained in the selected period. By default, this parameter is always computed whether you select it or not. However, it can be disabled using the ExtraMode.
Sum	Indicates the sum for the selected variable during the defined period.
Average	Indicates the average for the selected variable during the defined period.
Min	Indicates the min for the selected variable during the defined period.
Max	Indicates the max for the selected variable during the defined period.

Note

Meta-operators can be attached to each variable impacted by an operator. The available meta-operators are listed below:

- Delta calculates the difference between the values of two consecutive periods for all the periods.
- PercentIncrease calculates the difference in percentage between the values of two consecutive periods for all the periods.
- Accumulation calculates the current total accumulation for each period.
- BackAccumulation calculates the current total accumulation for each period calculated backwards.
- GlobalSum calculates the sum of all periods values.

Syntax

Path: `Protocols/Default/Transforms/Kxen.EventLog/Parameters/Transactions`

Parameter	Access	Description	Values
EventSpaceName	Read-write	The name that allows this SAP Predictive Analytics Explorer - Event Logging to find back information on the transactions. We had to allow different names for this role because the user can stack several SAP Predictive Analytics Explorer - Event Logging in a single protocol each SAP Predictive Analytics Explorer - Event Logging can deal with one specific transaction file. It is up to the user to specify then several event space names such as Events HotLine, Events Products and so on. All these names must start with "Events".	A character string. 'Events': default value. Note - a proper value is mandatory. Since it is defaulted to 'Events', this default cannot be used if two SAP Predictive Analytics Explorer - Event Logging are inserted in the same model.
IdColumnName	Read-write	Indicates the name of the column containing the identifier of the main object (customer ID, machine ID, session ID, etc...) in the log (transaction) table. This is used to join the event log with the proper case.	A character string. Note - a proper value is mandatory.
DateColumnName	Read-write	Indicates the name of the column containing the event (or transaction) in the events (or transactions) table. Together with the reference date, it is used to determine which period each event will be aggregated into.	A character string.
EventColumnName	Read-write	Some transactions can be associated with an event code (such as buy, test, receive_mailing, answer_mailing for a customer history, or type of failure for a machine on a network), and the user can force the aggregates only on a sub list of accepted event codes.	The default value is left empty, which means that no filter is applied. When this value is not empty, it should be used in conjunction with AcceptedEvent.

Parameter	Access	Description	Values
AcceptedEvent		This list allows filtering some events (only transactions with events present in the listed codes will be kept).	The value is left empty by default, which means that all events are kept.
DismissedEvent		This list allows filtering some events (only transactions with events present in the listed codes will be ignored).	The value is left empty by default, which means that all events are kept.

7.3.3.8.2 Results

There is no results provided by SAP Predictive Analytics Explorer - Event Logging. SAP Predictive Analytics Explorer - Event Logging only outputs new variables with their statistics on all data sets. These variables can be found in the Variables section of the parameter tree.

Different variables are created depending on the selected operators and meta operators. Different elements are used to build the variables names:

<prefix>	By default, the prefix is set to "el", but it can be modified.
<Period>, Pn	Number of the current period. The periods are numbered from 0. If there are 4 periods from -2 years to +2 years, the periods are numbered 0, 1, 2, 3. With 0 being the oldest one and 3 the last one.
n	the total number of periods minus 1. For the example above n=3.
<Operator>	The operator applied
<Meta>	The meta operator applied
<Variable>	The variable on which the operator applies

The following table details the generated output variables.

Operator / Meta Operator Name	Syntax	Example
Count	<prefix>_Count_<PeriodNumber>	el_Count_0
SumMinMaxAverage	<prefix>_<Operator>_<Variable>_<Period>	el_Sum_OS_1 el_Min_OS_0 el_Average_OS_3 el_Max_OS_2

DeltaPercentIncrease	<pre><prefix>_<Meta>_<Operator>_<Variable>_P0<->P1 <prefix>_<Meta>_<Operator>_<Variable>_P1<->P2 ... <prefix>_<Meta>_<Operator>_<Variable>_Pn-1<->Pn</pre>	el_Delta_Min_OS_0<->1 el_PercentIncrease_Sum_OS_1<->2
Accumulation	<pre><prefix>_<Meta>_<Operator>_<Variable>_P0+P1 <prefix>_<Meta>_<Operator>_<Variable>_P0+P1+P2 ... <prefix>_<Meta>_<Operator>_<Variable>_P0+P1+...+Pn</pre>	el_Accumulation_Min_OS_0+1+2
BackAccumulation	<pre><prefix>_<Meta>_<Operator>_<Variable>_P0+P1+...+Pn <prefix>_<Meta>_<Operator>_<Variable>_P1+...+Pn ... <prefix>_<Meta>_<Operator>_<Variable>_Pn-1+Pn</pre>	el_BackAccumulation_Max_OS_2+3
GlobalSum	<pre><prefix>_<Meta>_<Operator>_<Variable>_0<->n</pre>	el_GlobalSum_Average_OS_0<->3

7.3.3.9 Kxen.SequenceCoder

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder`

The purpose of SAP Predictive Analytics Explorer - Sequence Coding is to build a mineable representation of events history. It is not a data mining algorithm but a data preparation transform. All algorithms to perform regression, classification or segmentation only work on a fixed number of columns, but sometimes a customer can be associated with a list of events (the purchase history for example) with different size for every customer, so this list of events (or transactions) must be translated in a number of fixed columns in one way. These type of operations are called pivoting in data mining because they translate information contained in the same column of different lines into different columns on a single line (for a given customer identifier for example). SAP Predictive Analytics Explorer - Event Logging and SAP Predictive Analytics Explorer - Sequence Coding belong to these types of transformations.

SAP Predictive Analytics Explorer - Sequence Coding can be used to represent the history of a customer, or an history of a log of defects associated with a machine in a network. This component merges static information (coming from a static table) and dynamic information (coming from a log table). The user must have these two tables available before using the component. The table containing static information is generally called the "reference" table, and it is associated in the models with the classical data set names or roles such as Training, Estimation, Validation, Test or ApplyIn. The table containing the log of events (sometimes called the "transactions" table) is associated with a name beginning with the name "Events".

SAP Predictive Analytics Explorer - Sequence Coding is said to build fine-grained representations as it summarizes the count of different events or even the transitions between different events for a given reference object. A good usage example of SAP Predictive Analytics Explorer - Sequence Coding is when trying to represent web log sessions. The reference table contains information about the sessions, and the transactions table contains the click-stream. SAP Predictive Analytics Explorer - Sequence Coding is able to represent each session as the transitions between possible pages (or meta-information about the pages).

7.3.3.9.1 Parameters

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder/Parameters`

This section describes the parameters of the KSC component, which can be found under the Parameters section of the component.

⌘ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder/Parameters/Transactions`

This folder contains information related to the log (transaction) dataset.

Parameter	Description
EventsSpaceName	[read-only when transform is in ready state] The mandatory column in which is stored the name of the data set that holds information about the events. The default value is 'Events'. The name of this dataset must start with 'Events'.
IdColumnName	[read-only when transform is in ready state] Indicates the name of the column containing the identifier of the main object (customer ID, machine ID, session ID) in the transaction table. A proper value – that is, a value of a discrete variable – is mandatory.
DateTimeColumnName	[read-only when transform is in ready state] The mandatory column in which is stored the date of each event in the transaction data set. The system does not test for the fact that the events of this data set are extracted in the increasing date order. It is up to the environment to ensure this is correctly done (using proper index and sorted by SQL statement for example).

Parameter	Description
StartLogOnDate	[read-only when transform is in ready state] This allows to consider only sequences identifiers for which the Date column contains a value superior to the specified date.
EndLogOnDate	[read-only when transform is in ready state] This allows to only consider sequences identifiers for which the Date column contains a value inferior to the specified date.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder/Parameters/Reference`

This is the folder for information related to the sequences (static or reference table for sequence) dataset.

Parameter	Description
IdColumnName	[read-only when transform is in ready state] indicates the name of the column containing the identifier of the main object (customer ID, machine ID, session ID, etc...) in the reference table. This is used to join the event log with the proper case. A proper value – that is, a value of a discrete variable is mandatory.
DateColumnName	[read-only when transform is in ready state] This column (if filled) contains a date that will be compared to StartDate and EndDate. This allows to clip the references object between two dates. It must be noted that this is done of the reference table and NOT on the transactions table.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder/Parameters/Representation`

This folder contains all the information needed to specify what type of encoding of sequences is chosen by the user.

Parameter	Description
InterSeq	This is a boolean value that allows to generate intermediate sequence when set to 'true'. This value can be changed for runtime where the transaction set contains only the last valid events and the user wants to make real-time evaluation at this point of the sequence. This flag is mainly used to generate training data set from a transaction file when the desired predictive model is to be used at each step of a sequence of events. The default value is "false".
LastStepOnly	This is a boolean value only meaningful in intermediate sequences mode. This instructs KSC to only generate lines corresponding to last steps of sessions. The default value is "false".

Parameter	Description
InternalStorage	This contains the place holder that will allows us to activate our own internal storage to avoid large memory consumption. The default value is "Memory".
PercentageToKeep	This value is used to filter the CountXXX/CountTransitionXXX output columns generated by KSC (see KSC generated variables for more information).

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.SequenceCoder/Parameters/Representation/Operations`

Parameter	Description
Count	[folder] contains a simple template that allows the user to insert variable names. Each value of the inserted variables will be translated in a new column by KSC. Count encodes the sequences using one column per valid state in the specified column (states that are seen only once for the transactions associated with the reference id present in the Estimation data set are discarded).
CountTransition	[folder] contains a simple template that allows the user to insert variable names. Each transition of values of the inserted variables will be translated in a new column by KSC. CountTransition encodes the sequences using one column per valid pair wise state transition in the specified column (state transitions that are seen only once for the transactions associated with the reference id present in the Estimation data set are discarded).
FirstLast	[folder] contains a simple template that allows the user to insert variable names. All the standard columns not related to a specific operator (see the Result section) will be generated (that is: all the columns except for the ksc_Countxxx and ksc_CountTransitionxxx ones).

7.3.3.9.2 Results

There is no "real results" provided by SAP Predictive Analytics Explorer - Sequence Coding. SAP Predictive Analytics Explorer - Sequence Coding only outputs new variables with their statistics on all data sets. These variables can be found in the Variables section of the parameter tree.

As far as integration is concerned, there is a little trick to know. When integrating a SAP Predictive Analytics Explorer - Sequence Coding transform into a protocol, it is indeed useful to ask this transform to compute the output columns. The catch is that, in order to do this, SAP Predictive Analytics Explorer - Sequence Coding must make a first pass on the transaction table to find what the valid states and transitions are between states. This is done through a checkMode call to the model. The generated output variables (for example, the output columns) are provided below.

- `ksc_Start_Date`: the starting date of the session, deduced from the transaction space (can be unknown if bad date format is encountered).
- `ksc_End_Date`: the ending date of the session.
- `ksc_TotalTime`: the total time of the session (namely, the difference between `ksc_Start_Date` and `ksc_End_Date`).
- `ksc_Number_Events`: the total number of steps (events) in the session, computed from the transaction space.
- `ksc_XXX_FirstState`: the first state of the variable XXX at the first step of the session.
- `ksc_XXX_FinalState`: the final state of the variable XXX at the closing step of the session.
- `ksc_CountXXX-C_sn`: the number of times the 'sn' state has been encountered in the session for the variable XXX. The number of CountXXX columns can be regulated via the PercentageToKeep parameter: only those column whose total number of events (computed over all the sessions) account for the given percentage of all events will be kept.
- `ksc_CountXXX-KxOther`: a column agglomerating results for events not encountered enough to have their own Count column. Filtered out CountXXX columns are agglomerated here.
- `ksc_CountTransitionXXX-T_sn->sm`: the number of times the transition between 'sn' and 'sm' states has been encountered for the variable XXX. Those columns can be filtered out as with the `ksc_CountXXX-C_sn` ones.
- `ksc_CountTransitionXXX-KxOther`: a column agglomerating results for transitions not encountered enough to have their own Count column. Filtered out CountTransitionXXX columns are agglomerated here.
- `ksc_CountTransitionXXX-T_OUT->sn`: similar to sn->sm columns. The OUT state is SAP Predictive Analytics naming meaning that the session had not begun yet.

On intermediate sequences mode, the following variables are also generated:

- `ksc_LastStepNumber`: the number of steps of the session up to the previous step.
- `ksc_Last_date-time`: the date of the previous step in the session.
- `ksc_Last_duration`: the duration of the session up to the previous step.
- `ksc_Session_continue`: a boolean value saying whether the current step is ending the session.
- `ksc_XXX_LastState`: the previous step state of the variable XXX.
- `ksc_XXX_NextState`: the next step state of the variable XXX.

7.3.3.10 Kxen.TimeSeries

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries`

SAP Predictive Analytics Modeler – Time Series lets you build predictive models from data representing time series. Thanks to SAP Predictive Analytics Modeler – Time Series models, you can:

- Identify and understand the nature of the phenomenon represented by sequences of measures, that is, time series.
- Forecast the evolution of time series in the short and medium term, that is, to predict their future values.

How does SAP Predictive Analytics Modeler – Time Series work?

SAP Predictive Analytics Modeler – Time Series breaks down the signal into four components:

- The trend. The trend represents the evolution of a time series over the period analyzed. The trend is represented either by a function of time or by signal differentiating, which is calculated in SAP Predictive Analytics Modeler – Time Series using the principle that a value can be predicted well enough based on the previous known value. Calculating the trend allows to build a stationary representation of the time series (that is, the time series does not increase or decrease any more). This stationary representation is essential for the analysis of the three others components.
- The cycles. The cyclicity describes the recurrence of a variation in the signal. It is important to distinguish calendar time from natural time. These two time representations are often out of phase. The former - which is referred to as seasonality - represents dates (day, month, year and so on), while the latter - which is referred to as periodicity - represents a continuous time (1, 2, 3 and so on).
- The fluctuations. Fluctuations represent disturbances that affect a time series. In other words a time series does not only depend on external factors but also on its last states (memory phenomena). We try and explain parts of the fluctuations by modeling them on past values of the time series (ARMA or GARCH models).
- The residue. The information residue is the information that is not relevant to explain the target variable. As such, predictive models generated by SAP Predictive Analytics Modeler – Time Series are characterized only by the three components trend, cycles and fluctuations.

Another important part of a Time Series modeling is to make some forecasts. An SAP Predictive Analytics Modeler – Time Series model will use its own prediction in order to predict the next value.

7.3.3.10.1 Parameters

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Parameters`

This section describes parameters of SAP Predictive Analytics Modeler – Time Series which can be found under the 'Parameters' section of the component.

Parameter	Description	Values
<code>AutoFeedCount</code>	The number of steps in the future for which the model will be optimized (Learning Horizon).	An integer value.
<code>MaxCyclics</code>	The maximal number of cycles analyzed by SAP Predictive Analytics Modeler – Time Series. 450 is the maximal number of cycles that SAP Predictive Analytics Modeler – Time Series is able to analyze. During the learning phase, this number may be reduced to half of the estimation data set.	An integer value. 450 is the default value.
<code>MaxLags</code>	The number of lagged variables equal to a quarter of the estimation set size with no maximum value.	An integer value. Note - The fluctuations step can be skipped by setting this parameter to 0.
<code>LastRowWithForecastingInformation</code>	Saves the index of the last line of the file. This parameter is required if you want to use extra predictable inputs.	An integer value.

Parameter	Description	Values
ModelsGenerationOption	Controls how the models are generated internally by SAP Predictive Analytics Modeler – Time Series.	<ul style="list-style-type: none"> • Default • "Only Based on Extra Predictables" • "Disable the Polynomial Trend" • Customized
CustomizedModelGeneration(folder)	Used when the model generation is customized. It contains a boolean entry for each model.	Not applicable.
VariableSelection(folder)	This parameter groups some controls for the variable selection feature. When a variable selection is used, an automatic selection process is performed on trends or AR models during the competition and the result is kept only if it improves the final model.	<ul style="list-style-type: none"> • PercentageContribution • ActivateForEXP Trends • ActivateForAutoregressiveModels
ProcessOutliers	Activates an outliers processing strategy. Some extreme values are avoided when estimating the time-based trends leading to a more robust trend estimation.	<ul style="list-style-type: none"> • true (default value) • false
ForcePositiveForecast	Activates a mode where the negative forecasts are ignored (replaced by zero). This is useful when the user knows that the nature of the signal is positive (number of items in stock, amounts, number of passengers, and so on).	<ul style="list-style-type: none"> • true • false (default value)
AutoFeedCountApplied(Apply Horizon)	The number of steps in the future on which the model will be applied. This parameter may be different from the learning horizon (AutoFeedCount). This will generate as many forecasts.	<p>An integer value.</p> <p>Note - If this parameter is not set, it is equal to the learning horizon.</p>
ForecastsConnection	Gives the format of the forecasts in the output of SAP Predictive Analytics Modeler – Time Series.	<ul style="list-style-type: none"> • true(default value): the forecasts are transposed at the end of the kts_1 variable with the corresponding dates. • false: the forecasts stay in the last line of the file.
ExtraMode	A special flag that allows to set the type of outputs that SAP Predictive Analytics Modeler – Time Series will generate.	<ul style="list-style-type: none"> • No Extra • Forecasts and Error Bars • Signal Components • Component Residues

Parameter	Description	Values
ErrorBarsConfidence	Used to control the degree of confidence requested to compute the confidence interval for each horizon forecast (default 95%).	A percentage value. 95% is the default value.
DateColumnName	The column in which is stored the date. This parameter is set by the user and it cannot be changed later for a given model.	A character string.
DateNeedSetKeyLevel	Unused internally when the key level is not set.	Not applicable.
PredictableExtras	The folder that contains all the exogenous variables whose future values are known (like a variable describing "the first Friday of the month", an "isAWorkingDay" variable describing working days).	User-dependent
UnPredictableExtras	The folder that contains all the exogenous variables whose future values are not known (like variables describing "monthly benefits" or "oil crisis").	User-dependent

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Parameters/ModelsGenerationOption`

Parameter	Description
Default	No specific action. Generate all the known models.
"Only Based on Extra Predictables"	Restricts the models to those using extra predictable variables.
"Disable the Polynomial Trend"	Generates all the models but those containing a polynomial trend.
Customized	Gives the possibility to enable/disable any model generated by SAP Predictive Analytics Modeler – Time Series. A boolean flag is associated to each trend/cycle/fluctuation model type. These flags are detailed in the CustomizedModelGeneration parameter below.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Parameters/CustomizedModelGeneration`

The following options are relevant only if the ModelGenerationOption parameter is set to `Customized`.

Parameter	Description
Lag1	Controls the Lag1 trend (the previous value of the signal).
Lag2	Controls the Lag2 trend (the value before previous).
Linear	Controls the Linear trend (Linear regression on the time).
Polynomial	Controls the Polynomial trend (Polynomial regression on the time).

Parameter	Description
ExtraPredictors	Controls the ExtraPredictors trend (Linear regression on the extra-predictable Variables).
LinearPlusExtraPredictors	Controls the Time and ExtraPredictors trend (Linear regression on the time and extra-predictable Variables).
PolynomialPlusExtraPredictors	Controls the polynomial in time and linear in ExtraPredictors trend (polynomial regression on the time and linear in extra-predictable variables).
Cycles	Controls the cyclic variable detection.
Seasonal	Controls the Seasonal Variable detection.
PeriodicExtraPredictables	Controls the extra-predictable usage as periodics.
AutoRegression	Controls the autoregressive modeling.

For example, to disable the linear trend, you need to set `CustomizedModelGeneration/Linear` to `false`.

Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Parameters/VariableSelection`

Parameter	Description	Values
PercentageContrib	The percentage of contributions that are kept in the automatic selection process.	95% is the default value.
ActivateForEXPTrends	When set to <code>true</code> , it performs a variable selection on all extra-predictable-based trends. User variables are kept only if they have sufficient contributions in the trend regression.	<ul style="list-style-type: none"> <code>true</code> (default value) <code>false</code>
ActivateForAutoRegressiveModels	When set to <code>true</code> , it performs a variable selection on all auto-regressive models. This is an automatic selection on the past values (lags) of the signal. This leads to a more parsimonious (lower order and simpler) AR model.	<ul style="list-style-type: none"> <code>true</code> <code>false</code> (default value)

Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Parameters/ExtraMode`

Parameter	Description
No Extra	Generates basic outputs, that is the target variable and its predicted values.
Forecasts and Error Bars	Generates the same as above, with the confidence interval for each learning horizon.
Signal Components	Generates basic outputs plus signal components (trend, cycles and so on). Predicted values correspond to the sum of all of these signal components.
Component Residues	Generates the same outputs as Signal Components plus to the residues' values for each predicted signal component.

7.3.3.10.2 Results

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Results`

This section describes the results of SAP Predictive Analytics Modeler – Time Series.

Parameter	Description	Values
Variables (folder)	The folder where all variables used to build the model are stored. Each variable appears as a folder containing two sub-folders:	The <Variable_Name> folder is described by two subfolders: <ul style="list-style-type: none"> • Variable (folder) • KTS_Specifics (folder)
Model (folder)	The folder describing all the components used by the generated model. The specified value is the name of the model created by SAP Predictive Analytics Modeler – Time Series. Note - This variable can be found in the 'Variables' folder under 'Results' section. If this section does not exist, it means that no model has been found.	<ul style="list-style-type: none"> • MaximalHorizon • Trend (folder) • Fluctuations • Outliers (folder)
Perfs (folder)	The folder where all performance indicators on all data sets for all forecasts are stored (details on these indicators is given previously). These performances were computed between signal and all autofeed variables.	For detailed explanation, see KTS_Specifics Perfs values.
TimeAmplitude	Set only for datetime or date variable. This describes granularity of the amplitude between the first and the last date of the estimation data set (hour, day, month, year).	<ul style="list-style-type: none"> • hourAmplitude • dayAmplitude • monthAmplitude • yearAmplitude
TimeGranularity	Set only for datetime or date variable. This describes the average granularity between each dates of the Estimation data set (second, minute, hour, day, week, month, year).	<ul style="list-style-type: none"> • secondAmplitude • minuteAmplitude • hourAmplitude • dayAmplitude • weekAmplitude • monthAmplitude • yearAmplitude
TimeStorage	the storage type of date variable.	<ul style="list-style-type: none"> • date • datetime • integer • number
FirstDate	The first date in learning file.	date
LastDate	The last date is the learning file.s	date
IsDeltaTimeConstant	A boolean value that indicates if variation between each date are constant.	<ul style="list-style-type: none"> • true • false

Parameter	Description	Values
DeltaTime	The mean difference between two consecutive times observed on the estimated dataset.	A real value.
NbInit	The number of lines reading in the file before learning or applying. This index is used to initialize model variables.	Not applicable (it is an internal parameter).

Forecast n MAPE Indicators

Parameters	Description
Estimation/MAPE	MAPE indicator value for each horizon in the training dataset
Validation/MAPE	MAPE indicator value for each horizon in the validation dataset

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Results/Variables`

The Variables folder is the folder where all variables used to build the model are stored. Each variable appears as a folder containing two subfolders. The table below describes the two subfolders available.

Parameters	Description	Values
Variable (folder)	The folder for global information related to internal SAP Predictive Analytics Modeler – Time Series variables.	All SAP Predictive Analytics regular variable parameters, together with some specific ones (see KTS_Specifics parameter, detailed below).
KTS_Specifics (folder)	The folder for specific information for each variable build by SAP Predictive Analytics Modeler – Time Series. The value specified is the type of the variable generated by SAP Predictive Analytics Modeler – Time Series.	Variable-dependent.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Results/Model`

Parameter	Description	Values
MaximalHorizon	The maximal reliable horizon for the final SAP Predictive Analytics Modeler – Time Series Model. This horizon may be lower than the horizon requested by the user.	An integer value.
Trend (Folder)	The name of the trend used by the model.	Model-dependent.

Parameter	Description	Values
Cycles	Contains all the periodic and seasonal variables, separated by commas, used by KTS. For each cyclic, seasonal, extra-predictable variable in the cyclic component, the duration (when relevant) is given under Cycles/CycleName/DurationInSeconds.	A list of cyclic components.
Fluctuations	Describes the auto-regressive process used by the model. This process is noted AR, and the number between the parentheses gives the order of this one.	The "AR" process, followed by its order in parentheses. e.g.: [AR(37)]
Outliers (folder)	Provides the outliers for the current model.	For each data set and each outlier, the date, signal and model values are provided.

Note

One or more of the previous elements (Trend, Cycles, Fluctuations) may not exist. In this case, it means that the related component has not been detected by the model.

Outliers

For each outlier, the following three pieces of information are provided:

Parameter	Values
Date	A date format.
Signal	A real value.
Model	A real value.

7.3.3.10.3 Infos

Syntax

Path: `Protocols/Default/Transforms/Kxen.TimeSeries/Infos`

This folder contains the LearnTime parameter, that is the time (in seconds) needed for the model learning.

7.3.3.11 Kxen.TextCoder

Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder`

SAP Predictive Analytics Explorer - Text Coding (formerly known as KTC) is a solution for Text Analytics. It automatically prepares and transforms unstructured text attributes into a structured representation to be used within the SAP SAP Predictive Analytics modeling components.

SAP Predictive Analytics Explorer - Text Coding automatically handles the transformation from unstructured data to structured data going through a process involving “stop word” removal, merging sequences of words declared as 'concepts', translating each word into its root through “stemming” rules, and merging synonyms. SAP Predictive Analytics Explorer - Text Coding allows text fields to be used “as is” in classification, regression, and clustering tasks. It comes packaged with rules for several languages such as French, German, English and Spanish, and can be easily extended to other languages.

SAP Predictive Analytics Explorer - Text Coding improves the quality of predictive models by taking advantage of previously unused text attributes. For example, messages, emails sent to a support line, marketing survey results, or call center chats can be used to enhance the results of models for cross-sell or attrition.

7.3.3.11.1 Parameters

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder/Parameters`

This section describes parameters of SAP Predictive Analytics Explorer - Text Coding that can be found under the 'Parameters' section of the component.

Parameter	Description	Values
<code>WordSeparators</code>	Indicates a list of possible word separators in the textual fields. The default value contains the following separators: <code>, . [] () < > " + & = + # @ * ! ' - : ; ? / \ tabulation return</code>	

Parameter	Description	Values
LanguageStore		<ul style="list-style-type: none"> Location: location of new language files. A language is represented by two files: StemmingRules_<language name> and StopList_<language name> (by default this parameter is empty). ExcludedLanguage: list of the languages excluded from the language automatic detection (by default this parameter is empty).
Language		<ul style="list-style-type: none"> Language: the language of the textual fields (default value en). LanguageDefaultValue: the default language if no language has been detected during the language automatic detection (the default value is en).
LanguageDetectionEnabled	A Boolean value that indicates whether the language automatic detection is enabled.	<ul style="list-style-type: none"> False: If set to False, the user-defined language will be used. True: default value.

☰ Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder/Parameters/ExtraMode`

A special flag that allows setting the type of outputs that SAP Predictive Analytics Explorer - Text Coding will generate during an apply.

Parameter	Description
Transactional	Generates one row for each identified root with: <ul style="list-style-type: none"> • a column containing the key index from the original row, • a column containing the index of the current root in the textual field, • a column indicating from which textual field the root is extracted, • a column containing the current root.
Vectorization	Generates all the columns provided in the original data set and for each textual field: <ul style="list-style-type: none"> • a column is created for each root identified by the model. If the root represented by the column is present in the record, the value is set to 1, else it is set to 0. • one column provides the number of elements recognized by SAP Predictive Analytics Explorer - Text Coding in the record • one column provides the number of distinct root found in the record.
Language Detection	Generates for each textual field a column indicating the language recognized by SAP Predictive Analytics Explorer - Text Coding for this record. The value can be the ISO language code, or the empty value if no language is recognized.
Generate Only Roots	Generates the following columns for each textual field: <ul style="list-style-type: none"> • one column for each root identified by the model. If the root represented by the column is present in the record, the value is set to 1, else it is set to 0. • one column providing the number of elements recognized by SAP Predictive Analytics Explorer - Text Coding in the record • one column providing the number of distinct root found in the record.

Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder/Parameters/ProcessingOptions`

Parameter	Description	Values
StopListEnabled	A Boolean value that indicates whether the stop list will be used or not.	<ul style="list-style-type: none"> • true(default value) • false
StemmingEnabled	A Boolean value that indicates whether the stemming rules will be used or not.	<ul style="list-style-type: none"> • true(default value) • false
ConceptListEnabled	A Boolean value that indicates whether the concept list will be used or not.	<ul style="list-style-type: none"> • true • false(default value)
SynonymyEnabled	A Boolean value that indicates whether the synonymy will be used or not.	<ul style="list-style-type: none"> • true • false(default value)
DebugMode	A Boolean value that indicates whether the debug mode is activated.	<ul style="list-style-type: none"> • true • false(default value)
VolatileStopList	A parameter used to define a user's list of stop words.	The default value is empty.

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder/Parameters/RootSelection`

Parameter	Description	Values
<code>RankingStrategy</code>	Allows you to select the ranking strategy for the root selection, that is, to select which roots to keep in the dictionary.	<ul style="list-style-type: none">• <code>Frequency</code>(default value)• <code>shannonEntropy</code>• <code>kullbackInformation</code>• <code>mutualInformation</code>• <code>chiSquare</code>• <code>informationGains</code>
<code>NbRootGenerated</code>	Indicates the maximum number of roots generated by SAP Predictive Analytics Explorer - Text Coding.	1000: default value
<code>MinThreshold</code>	Indicates a threshold in percentage. If a root appears in less than the indicated percentage of all textual fields, it will be eliminated.	5: default value
<code>MaxThreshold</code>	Indicates a threshold in percentage. If a root appears in more than the indicated percentage of all textual fields, it will be eliminated.	100: default value

≡ Syntax

Path: `Protocols/Default/Transforms/Kxen.TextCoder/Parameters/EncodingStrategy`

Each root is converted into a variable and when the root appears in a text, its presence can be encoded with one of the strategies listed.

Parameter	Description	Values
<code>boolean</code>	Specifies whether the word is present or not.	1: the word is present. 0(default value): the word is absent.
<code>termFrequency</code>	The frequency of apparitions of the root in the current text.	An integer value.
<code>termCount</code>	The frequency of apparitions of the root in the current text.	An integer value.
<code>termFrequencyInverseDocumentFrequency</code>	Stands for the apparition frequency of the root in the current text divided by the apparition frequency of the root in the whole set of texts.	An integer value.
<code>termCountInverseDocumentFrequency</code>	The number of apparitions of the root in the current text divided by the number of apparitions of the root in the whole set of texts.	An integer value.

7.4 DataSets

☰, Syntax

Path: `DataSets`

A data space is an ordered list of cases (or events). It can be viewed as a file in a folder, a table in a database, a `SELECT` statement (using SQL), or an Excel worksheet. A data space is generally associated with a model through a role name: we call this association a 'dataset'. A classical example is when the transform must be trained (you may prefer the term 'estimated') on a set of examples: in this case the set of examples used to estimate the transform parameters will be known to the model as the "Estimation" dataset. Data spaces belong to stores. A store federates several physical spaces. It can be viewed as a file folder, a database, or an Excel workbook. Stores can be created directly because models can be saved into specific spaces contained in stores: this mechanism allows to view the models saved into a store and to restore them.

7.4.1 Parameters

☰, Syntax

Path: `DataSets/<Dataset_Name>/Parameters`

Parameters	Access	Description	Possible Values
<code>MappingByPositionCheckPolicy</code>	Read-write	When SAP Predictive Analytics needs to use a mapping by position, it checks that technical columns and variables have compatible types. This check can be setup to be very strict and accept only equals types or more loose and accept some type conversions. For example, a integer to datetime conversion is always rejected, but a text to datetime conversion is accepted with <code>SoftCheck</code> setup and rejected with <code>StrongCheck</code> setup.	<ul style="list-style-type: none">• <code>StrongCheck</code>• <code>SoftCheck</code> (Default value)

Parameters	Access	Description	Possible Values
MappingReportSuccessUILevel	Read-write	This parameter allows tuning the quantity of information displayed to the user when a successful mapping is done.	<ul style="list-style-type: none"> • NoReport (Default value): it does not provide any information. • SmallReport: it displays only a short report on important information (for example, the number of unmapped columns). • FullReport: it displays all detailed informations (for example, the full list of columns not mapped).
MappingReportFailureUILevel	Read-write	This parameter allows tuning the quantity of information displayed to the user when a mapping has failed.	<ul style="list-style-type: none"> • NoReport it does not give any information. • SmallReport (the default value): it displays only a short report on important information related to the failure (for example, the number of mandatory variables not mapped). • FullReport it displays all detailed information (for example, the full list of variables not mapped).

Parameters	Access	Description	Possible Values
MappingReportParameterLevel	Read-write	For each mapping, whatever its result, a report is stored in the parameters tree, which the user can programmatically investigate. As previously, the level of information stored in the parameters tree can be tuned with this parameter.	<ul style="list-style-type: none"> • <code>NoReport</code> it does not give any information. • <code>SmallReport</code> it displays only a short report on important information related to the failure (for example, the number of mandatory variables not mapped). • <code>FullReport</code> (Default Value): it displays all detailed information (for example, the full list of variables not mapped).

Parameters	Access	Description	Possible Values
MappingResults	Read-only	A set of parameters describing what is the detailed result of the mapping process.	<ul style="list-style-type: none"> • MappingOK • PhysicalToLogical • MappingByName • HasAddedKxIndex • NbMandatoryVariables • NbOptionalVariables • NbFieldsForMap • NbMandatoryVariablesNotMapped • NbVariablesNonCompatible • NbVariablesMappedWithForbiddenConversion • NbVariablesMapped • NbVariablesAutoName • NbVariablesMappedUserName • NbVariablesMappedWithConversion • NbFieldsMultiUsed
Explain	Read-only	If the parameter is set to <code>true</code> , then when submitting an SQL request, instead of returning the resulting values, the DBMS returns a specific result set describing step by step how the SQL request will be executed and how much time each step will take.	<ul style="list-style-type: none"> • <code>true</code> • <code>false</code>(default value)
CanExplain	Read-only	Asks whether SAP Predictive Analytics can manage the Explain mode.	<ul style="list-style-type: none"> • <code>true</code> • <code>false</code>
FilterCondition (Directory)		Describes the filter as a tree of atomic tests to be evaluated.	<ul style="list-style-type: none"> • Operator • Variable • Value

Parameters	Access	Description	Possible Values
FilterCondition String	Read-only	A character string featuring a filter condition.	A character string, for example 'where'
Locale	Read-write	It is used to specify the character set of the data set (ASCII, Latin-1,UTF-8), the decimal symbol (dot or comma) and the number of digits after the decimal. The following table details these elements and their values.	<ul style="list-style-type: none"> • L • S • P • D
Connector	Read-write	It indicates whether or not the space is a data manipulation.	<ul style="list-style-type: none"> • true: the root of the data manipulation structure. • false
SkippedRows	<ul style="list-style-type: none"> • Read-write • Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is the number of rows that the system must skip before actually reading data. This is filled in when the user specify a periodic cut strategy for a model.	An integer
LastRow	<ul style="list-style-type: none"> • Read-write • Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is the last valid row that the system will take as part of the data set. This is filled when the user specify a periodic cut strategy for a model.	An integer value.
ModuloMin	<ul style="list-style-type: none"> • Read-write • Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is used for periodic cut training strategy and filled by the system.	An integer value.
ModuloMax	<ul style="list-style-type: none"> • Read-write • Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is used for periodic cut training strategy and filled by the system.	An integer value.

Parameters	Access	Description	Possible Values
ModuloPeriod	<ul style="list-style-type: none"> Read-write Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is used for periodic cut training strategy and filled by the system.	An integer value.
RandomMin	<ul style="list-style-type: none"> Read-write Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is used for random cut training strategy and filled by the system.	An integer value.
RandomMax	Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space	It is used for random cut training strategy and filled by the system.	An integer value.
RandomSeed	Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space	It is used for random cut training strategy and filled by the user.	An integer value.
HeaderLines	<ul style="list-style-type: none"> Read-write Read-only for all input spaces except ApplyIn when the model is in ready state, not available for output space 	It is the number of lines that the system must skip before actually reading the header line.	<p>If this parameter is set to 0, then the data file begins with the column names.</p> <p>Note - this parameter is used only if the ForceHeaderLine parameter is set to true.</p>
RowsForGuess	Read-write	It is the number of rows that SAP Predictive Analytics will read in order to analyze the actual data and guess the value of variables (nominal, ordinal, continuous) and depending on the kind of data access, their storage (integer, number, string, date, datetime).	An integer value.

Parameters	Access	Description	Possible Values
GuessDescriptionUsesConnectorInfo	Read-write	It is a boolean flag used by connectors to give priority to or ignore the user-defined information (fields storage, value type, description, ...) in the guess description process.	<ul style="list-style-type: none"> • <code>true</code>: the information coming from the user will be preferred and used. • <code>false</code>: the guess description will always follow the full process (which involves the analysis of the first N data rows).
ForceHeaderLine	Read-write		<ul style="list-style-type: none"> • <code>true</code>: if the data has a header line but the analysis done by SAP Predictive Analytics did not find it. For example, this happens when the data set contains only strings or only numeric values. In such a case, by design, the data set does not have any header and all rows are considered as data. When set to <code>true</code>, the parameter <code>HeaderLines</code> must also be set. • <code>false</code>: explain???
Open	Read-only	The name of the space (that can be used to open the physical space containing data) or the full SQL string when the space is a data manipulation.	A character string.
Specification	Read-only	It is either the name of the data manipulation being used (when working with a data manipulation) or the value of the <code>Open</code> parameter (see above).	A character string.

Parameters	Access	Description	Possible Values
Mode	Read-only		<ul style="list-style-type: none"> • <code>r</code>" (default value): for input space • <code>w</code>"': for output spaces • <code>"a"</code>: also for output spaces
Store	Read-only	The open string used to open the store to which belongs the space. A store can be either a folder or an ODBC source.	A character string.
StoreClass	Read-only	The class of the store 'Kxen.ODBCStore' or 'Kxen.FileStore'.	<ul style="list-style-type: none"> • Kxen.ODBCStore • Kxen.FileStore • Kxen.UserStore • Kxen.FlatMemoryStore
State	Read-only	The state of the data space.	<ul style="list-style-type: none"> • <code>created</code> • <code>withVariableStatistics</code>
FilterConditionString	Read-only	A user-friendly version of the filter applied to the training data set, if there is any.	A character string.

7.4.1.1 MappingResults

Parameters	Description	Values
MappingOK	Specifies if the mapping is successful or not.	<ul style="list-style-type: none"> • <code>true</code>: when the mapping is successful • <code>false</code>: when the mapping failed.
PhysicalToLogical	Is set to <code>true</code> when the mapping is done in read mode (mapping a technical column name to a variable) or in write mode (mapping a variable to a technical column name).	<ul style="list-style-type: none"> • <code>true</code> • <code>false</code>

Parameters	Description	Values
MappingByName	Is set to <code>true</code> if the mapping has successfully associated all mandatory variable names to a unique technical column name. The comparison is case-insensitive but will prefer case-sensitive matching. For example, for a variable named <code>Age</code> and technical names <code>AGE</code> and <code>Age</code> , the mapping will select <code>Age</code> . If <code>AGE</code> is the only technical column name available, it will be selected. If <code>MappingByName</code> is <code>false</code> , it means that a mapping by position has been tried. The variable and technical column names have matched related to their position.	<ul style="list-style-type: none"> • <code>true</code> • <code>false</code>
HasAddedKxIndex	Specifies if SAP Predictive Analytics has automatically added its own key. This occurs when no other key has been declared.	<ul style="list-style-type: none"> • <code>true</code>: if SAP Predictive Analytics has automatically added its own key. • <code>false</code>: if a key exists, but has not been added by SAP Predictive Analytics.
NbMandatoryVariables	Refers to the number of variables that must be successfully mapped. This set of variables depends on the data set and on the operation requested. For example, for a regression, a training dataset must have its target mapped. But for an <code>applyIn</code> dataset, the target is not mandatory.	An integer value.
MandatoryVariables	Refers to the list of variables that must be successfully mapped. This set of variables depends on the data set and on the operation requested. For example, for a regression, a training dataset must have its target mapped. But for an <code>applyIn</code> dataset, the target is not mandatory.	List of the variables separated by a comma.
NbOptionalVariables	Refers to the number of variables that need not be mapped.	An integer value.
OptionalVariables	Refers to the list of variables that need not be mapped.	List of the variables separated by a comma.
NbFieldsForMap	Refers to the number of technical column names that SAP Predictive Analytics has found in the data set.	An integer value.
FieldsForMap	The list of technical column names that SAP Predictive Analytics has found in the data set.	List of the fields separated by a comma.

Parameters	Description	Values
NbMandatoryVariablesNotMapped	Refers to the number of mandatory variables not mapped.	An integer value. Any value other than 0 indicates a mapping error.
MandatoryVariablesNotMapped	Refers to the list of mandatory variables not mapped.	List of the variables separated by a comma.
NbVariablesNonCompatible	Refers to the number of variables that matched but needed a conversion depending on the CheckPolicy. For example, with StrongCheck an integer technical column cannot be mapped to a text variable.	An integer value.
VariablesNonCompatible	Refers to the list of variables that matched but needed a conversion depending on the CheckPolicy.	List of the variables separated by a comma.
NbVariablesMappedWithForbiddenConversion	Refers to the number of variables that matched but needed a forbidden conversion. For example, an integer technical column cannot be mapped to date-time variable.	An integer value.
VariablesMappedWithForbiddenConversion	Refers to the list of variables that matched but needed a forbidden conversion.	List of the variables separated by a comma.
NbVariablesMapped	Refers to the number of variables successfully mapped whatever their mandatory status.	An integer value.
VariablesMapped	Refers to the list of variables successfully mapped whatever their mandatory status.	List of the variables separated by a comma.
NbVariablesAutoName	Refers to the number of variables which have been automatically matched by SAP Predictive Analytics without any user action.	An integer value.
VariablesAutoName	Refers to the list of variables which have been automatically matched by SAP Predictive Analytics without any user action.	List of the variables separated by a comma.
NbVariablesMappedUserName	Refers to the number of variables which have been explicitly matched by the user (with the use of the InSpaceName or Spacename mechanisms).	An integer value.
VariablesMappedUserName	Refers to the list of variables which have been explicitly matched by the user (with the use of the InSpaceName or Spacename mechanisms).	List of the variables separated by a comma.
NbVariablesMappedWithConversion	Refers to the number of variables which have been matched but needed a type conversion.	An integer value.

Parameters	Description	Values
VariablesMappedWithConversion	Refers to the list of variables which have been matched but needed a type conversion.	List of the variables separated by a comma.
NbFieldsMultiUsed	Refers to the number of technical column names that have been used several times in the current mapping. A technical column name can be used only once.	An integer value.
FieldsMultiUsed	Refers to the list of technical column names which have been used several times in the current mapping. A technical column name can be used only once.	List of the fields separated by a comma.

7.4.1.2 Locale

The Locale parameter must be filled in the following format:

L=<locale>;S=<decimal symbol>;P=<precision>;D=<date representation>

The elements must be separated by semicolons.

Parameters	Description	Possible Values
L	Stands for "locale". It refers to a character set.	file locale. Example: fr_FR@8859-1, en_US.ISO8859-1, UTF-8,...
S	Stands for "decimal symbol".	<ul style="list-style-type: none"> 0, for dot 1, for comma
P	Stands for "precision". It refers to the number of digits after the decimal. This element is only used when writing.	for example: 2 0 has the special meaning of "best width for a readable value"
D	Stands for "date". It refers to the date format used within this file.	The value is of the form: XXX[:Z] XXX is a group of 3 letters (among 'Y', 'M' and 'D'), indicating in which order Year, Month and Day are represented. Z is a symbol giving the separator used between each of these. Examples of valid dates: YMD DMY:/ MDY:-

7.4.1.3 Parameters Specific to File Spaces

Possible paths to the Parameters of the data set (according to the type of dataset):

- `DataSets/Estimation/Parameters`
- `DataSets/Training/Parameters`
- `DataSets/Validation/Parameters`

The following parameters are specific to file spaces:

Parameters	Description	Values
<code>WarnIfLessFields</code>	A flag indicating if the user wants to be warned when the system does not find all fields on some line.	
<code>Separators</code>	This parameter can be changed until the space is actually read or write for the first time. The separator character used to separate fields in the line. In the user interface, user is required to describe characters as strings such as " <code><tab></code> ", " <code><comma></code> ", " <code><semicolon></code> ", " <code><space></code> ", or any other non special character. More than one character can be specified here.	
<code>TrimmedChars</code>	Characters that are trimmed when appearing around fields. The separator character used to separate fields in the line. In theThis parameter can be changed until the space is actually read or write for the first time.	
<code>QuotingPolicy</code>	The quoting policy of the fields content.	<ul style="list-style-type: none">• <code>never</code>: no matter what the content of the fields are, it is never quoted.• <code>ifNeeded</code>: the field content is quoted only when it contains a space or a special character.• <code>always</code>: the field content is always quoted.

i Note

Many parameters are available to tune the parameters for ODBC Spaces. See ODBC Fine Tuning documentation.

7.5 Plan

☰ Syntax

Path: Plan

The Plan groups together all parameters involved when performing In-database Application (IDBA). The IDBA is also an optimized scoring mode.

☰ Syntax

Path: Plan/Conditions

All following parameters must be true to fully perform the in-database-apply process.

Parameter	Access	Default Value	Description
OnODBCStore	Read-only	true	the applyIn store and applyOut store are an ODBC type
OnSameDataBase	Read-only	true	the applyIn space and applyOut space are on the same ODBC source
OnDifferentTable	Read-only	true	the applyIn space and applyOut space are different
NoSubSamplingDefined	Read-only	true	no sub-sampling has been defined on the applyIn space
KMXExistForThisODBC	Read-only	true	this ODBC source is available in SAP Predictive Analytics scorer
KMXLicenseAvailable	Read-only	true	a license scorer is valid for the ODBC source
KMXDefinedForTransformChain	Read-only	true	all transforms in the current model could be exported with the in-database-apply process
PrimaryKeyDefinedForApplyIn	Read-only	true	the current model has a physical primary key
LastTransformCompliant	Read-only	true	the tuning parameter of the last transform for the current model is exported with the in-database-apply process

☰ Syntax

Path: Plan/Options

Parameter	Access	Default Value	Description
Condition	Read-only	[Exist]	
LogEnabled	Read-only	[false]	

Parameter	Access	Default Value	Description
NbColumnByUpdate	Read-only	[n]	<p>The value of this parameter defines the number of element that is updated by pass.</p> <p>For example, when the in-database-apply process exports model contributions, it updates n columns by pass (where n is the value of this parameter). 0 indicates no limitation on element by update pass.</p>

☰ Syntax

Path: Plan/Steps

Parameters	Access	Description	Values
Query	Read-only	This parameter represents the internal SQL code that be executed at the steps .	[CREATE] [INSERT] [UPDATE]
Volume	Read-only	When it is possible, this parameter contains volumetric information of the internal SQL code for the current step.	
Comment	Read-only	This parameter stores a short comment of the current internal SQL code.	[comment]

☰ Syntax

Path: Plan/Results

Parameters	Access	Description	Values
PreparationTime	Read-only	Represents the time to initialize the in-database-apply process.	
ExecutionTime	Read-only	Represents the time to execute the entire SQL plan defined by each steps.	
Status	Read-only	<p>Stores the result of the in-database-apply process.</p> <p>In case of Failure, when information are available, the error will be stored in Plan/Results/ERROR.</p>	Failure/Success

7.6 External Executables

An external executable lets you define a list of programs (executables or scripts) in a configuration file. These executables can be run from a client on a server.

See the *SAP Predictive Analytics Administrator Guide* for more information.

7.6.1 ExternalExecutableAvailable

☞ Syntax

Path: `ExternalExecutableAvailable`

This parameter contains the list of available external executables. Each external executable is defined by a label and a description.

☞ Syntax

Path: `ExternalExecutableAvailable/<External_Executable_Name>`

This parameter contains the name of the external executable as defined by the key `ExternalExecutable.Name` in the configuration file.

Parameter	Description
Label	<p>This parameter contains the label of the current external executable. The label is in the current language if it has been translated in the configuration file.</p> <p>This label is commonly a word that describes the behavior of the external executable.</p>
Description	<p>This parameter contains the description of the current external executable. The description is in the current language if it has been translated in the configuration file.</p> <p>The description is commonly one or two sentences that describe the behavior of the external executable.</p>

☞ Syntax

Path: `ExternalExecutableAvailable/ExternalExecutableName`

This node appears after the node `ExternalExecutableName` has been set and a command `validateParameter` has been performed. It contains all information about the current script.

7.6.2 External_Executable_Name

≡ Syntax

Path: <External_Executable_Name>

This node appears after the node `ExternalExecutableName` has been set and a command `validateParameter` has been performed. It contains all information about the current script.

Parameter	Description	Value
<code>ExternalExecutableName</code>	This parameter contains the name of the file that will be executed.	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.External_ExecutableName</code> defined in the configuration file.
<code>Description</code>	This parameter contains the description of current external executable.	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.Description</code> defined in the configuration file.
<code>Label</code>	This parameter contains an identifier that will be displayed by the SAP Predictive Analytics Modeler. It is generally a word used to identify the external executable.	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.Label</code> defined in the configuration file.
<code>DefaultOutput</code>	This parameter is a Boolean that indicates whether the output is in the standard stream (in which case it will be displayed in the SAP Predictive Analytics Modeler).	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.DefaultOutput</code> defined in the configuration file.
<code>FormatOutput</code>	This parameter is a string that indicates the format of the output. If its value is set to <code>txt</code> or <code>html</code> , the output will be displayed by the SAP Predictive Analytics Modeler.	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.DefaultOutputFormat</code> defined in the configuration file. Possible Values: <ul style="list-style-type: none">• <code>txt</code>• <code>html</code>• User value
<code>IsScript</code>	This parameter is a Boolean that indicates if the current external executable is a script or an executable.	Its value corresponds to the value of the key <code>ExternalExecutable.<External_Executable_Id>.isScript</code> defined in the configuration file.

Parameter	Description	Value
NbArgument	This parameter is an integer that indicates the number of arguments required by the external executable.	Its value corresponds to the value of the key <code>ExternalExecutable.<External Executable Id>.NbArgument</code> defined in the configuration file. Possible values: a positive integer
Arguments	This node contains a list of nodes corresponding to the arguments used by the current external executable.	

≡ Syntax

Path: `<External_Executable_Name>/Arguments/Argument_<n>`

This parameter contains the information on the nth parameter of the current external executable.

Parameter	Description	Value
Label	The name of the nth parameter of the current external executable	Its value corresponds to the value of the key <code>ExternalExecutable.<External Executable Id>.Argument.<id>.Label</code> defined in the configuration file.
DefaultValue	The default value of the nth parameter of the current external executable	Its value corresponds to the value of the key <code>ExternalExecutable.<External Executable Id>.Argument.<id>.DefaultValue</code> defined in the configuration file.
Description	The description of the nth parameter of the current external executable	Its value corresponds to the value of the key <code>ExternalExecutable.<External Executable Id>.Argument.<n>.Description</code> defined in the configuration file.

Parameter	Description	Value
ArgumentType	The type of the nth parameter of the current external executable	<p>Its value corresponds to the value of the key <code>ExternalExecutable.<External Executable Id>.Argument.<n>.Type</code> defined in the configuration file.</p> <p>Possible Values:</p> <ul style="list-style-type: none"> ExistingStore ExistingSpace Store Space Bool Index Number Integer Double String
Value	The value of the external executable parameter to be used when executing it	
AllowedValue	This node exists only if the parameter requires a list of allowed value. In this case it contains a list of nodes, each node representing one of the allowed values.	
NbAllowedValue	This node contains the number of allowed values.	If the value is set to 0 any value of the correct type will be accepted. This number also represents the number of nodes under the node <code>AllowedValue</code> .
Prefix	This node contains the prefix that will be concatenate to the value.	

7.6.3 Other Parameters

☰ Syntax

Path: `ScriptInformation`

Parameter	Description	Values
ScriptLauncher	The command used to launch the external executable when it is a script.	The default values are the following: <ul style="list-style-type: none"> Linux: /bin/sh Windows: c:\windows\system32\cmd.exe
ScriptLauncherOption	This parameter contains the option used to launch the external executable when it is a script.	The default values are the following: <ul style="list-style-type: none"> Linux: -c Windows: /C
ScriptExtension	This parameter contains the extension that will be added at the end of the name of the script if needed.	The default values are the following: <ul style="list-style-type: none"> Linux: .sh Windows: .bat

Note

The syntax of the command is:

```
ScriptLauncher ScriptLauncherOption <External_Executable_Name> ScriptExtension
```

Syntax

Path: ExternalExecutableName

This parameter is used to specify the name of the script that you want to execute. The value is one of the values listed in the node `ExternalExecutablesAvailable`.

Syntax

Path: UseDefaultPath

This parameter is a Boolean that indicates whether the value of the key `ExternalExecutablePath` must be added in front of the name of the external executable before calling it.

8 KxShell: SAP Predictive Analytics Command Line Interpreter

Learn how to write scripts for the most common data-mining tasks with the scripting tool, KxShell.

Find the KxShell reference guide in the [Automated Analytics API Reference](#) on the [SAP Help Portal](#).

Related Information

[Overview \[page 214\]](#)

[Regression and Classification \[page 215\]](#)

[Segmentation \[page 222\]](#)

[KxCORBAShell \[page 227\]](#)

8.1 Overview

This document explains how to write scripts for the most common data-mining tasks with the scripting tool, KxShell.

KxShell is distributed with its source code as an example of how to use the C++ library directly in a C++ program.

There are three ways of using KxShell:

- Running the command `kxshell.exe script.txt` where `script.txt` is a text file containing the script.
- Launching KxShell and typing the command `read script.txt` where `script.txt` is a text file containing the script.
- Launching KxShell and typing interactively the commands, which can for example be copied from a document and pasted into the KxShell console.

The KxShell can be used to automate data-mining tasks because it does not require interaction. For example, a program can automatically generate a script file named `script.txt` and launch the external command `kxshell.exe script.txt`.

Every command/instruction executed is terminated with a 'OK' or 'not OK' status: For example:

```
Store.openStore "C:\"
```

This command is correctly executed because the instruction was executed successfully.

```
Store.openStore "DoesNotExist"
```

This command displays an error because the instruction did not complete successfully.

A KxShell script is a sequence of KxShell instructions; the `script.txt` script can be described as the sequence as follows:

```
Inst1..
Inst2..
Inst3..
```

Typing the command `kxshell.exe script.txt` helps executing the sequence of instructions contained in the `script.txt` script.

If an error is generated by a given instruction in the script, the execution of the script stops.

A modifier ('-' or '+') can be used at the beginning of each instruction / line of the script.

- if the command is preceded by '-': If there is an error, it is ignored and the execution of the script goes on, the next instruction is executed. For example, before applying a model, it is required to delete an output table if it exists:
`-store.dropSpace "MyOutputTable"`
`model.apply`
If the `MyOutputTable` table does not exist, the error is ignored and the `model.apply` instruction is executed.
- if the command is preceded by '+': here, the execution of the script goes on only if the current instruction fails. The '+' prefix is mostly used by developers but its use may be useful in some cases. For example, it may be required to check that a table does not exist. An attempt to read the table is performed and this attempt must fail. Otherwise, it means that the table exists.
`+store.readSpace "MyTestTable"`
`model.apply`
If the `MyTestTable` table does not exist, the first instruction causes an error. Then the `model.apply` instruction is executed.
If the `MyTestTable` exists, the first instruction completes successfully. That stops the execution of the script.

The scripts of the first section will show how to use KxShell in a regression/classification task context, while the scripts of the second section focus on clustering tasks. All the scripts use the data set known as Adult Census, which is distributed with the software along with its description file. When files are used, it is assumed that `Census01.csv` and `desc_Census01.csv` are in the current working directory; the models and output files are also saved in the current directory. When an ODBC source is used, it is assumed that an ODBC connection with the name Database is set up for user Username/password.

8.2 Regression and Classification

8.2.1 Basic Script Using Text Files

Basic Script Using Text Files

This script trains a regression model on a training data set. It uses files as the data source. The script executes the steps listed below:

1. Create a model variable of the type `Kxen.SimpleModel`.

```
createModel Kxen.SimpleModel census1
```

2. Open the data store containing the data set needed to create the model and the data set description. Since the model uses files, the type of the store will be `Kxen.FileStore`.
`census1.openNewStore Kxen.FileStore .`
3. Set the data set as the training data set.
`census1.newDataSet Training Census01.csv`
4. Read the description of the training data set.
`census1.readSpaceDescription Training desc_Census01.csv`
5. A model processes the data to extract information and knowledge. This process is called a 'protocol', which is a sequence of transforms to applied to the data. Models usually hold a single protocol (called 'Default'). A regression model contains a coding transform followed by a regression transform:
`census1.addTransformInProtocol Default Kxen.ConsistentCoder`
`census1.addTransformInProtocol Default Kxen.RobustRegression`
6. The model is now ready to process the data, which is called learning. This is the potentially lengthy operation.
`census1.sendMode learn`
7. Once the model has learned the data and stored the information and knowledge in the memory, it can be saved for future use.
`census1.saveModel Models.txt "This is the basic model, trained on a flat file." census1`
8. Add either `delete census1` to free the memory and be able to create new models, or `quit`.

8.2.2 Basic Script Using an ODBC Source

Basic Script Using an ODBC Source

This script has exactly the same behavior as the previous one, except for the fact that it uses an ODBC source.

1. The first step is the same:
`createModel Kxen.SimpleModel census1`
2. Open the data store. Since the script uses an ODBC source, the type of this store is `Kxen.ODBCStore`. At this step, the name of the ODBC source, the user name and the password need to be specified.
`census1.openNewStore Kxen.ODBCStore Database UserName password`
3. Specify the name of the table. In this example, the table is named `Census01`.
`census1.newDataSet Training Census01`
4. Read the description of the training data set, which is contained in the `desc_Census01` table.
`census1.readSpaceDescription Training desc_Census01`
`census1.addTransformInProtocol Default Kxen.ConsistentCoder`
`census1.addTransformInProtocol Default Kxen.RobustRegression`
`census1.sendMode learn`
Note - If the description is stored in a file, open a new store of type `Kxen.FileStore` before writing this command with the suitable filename.
5. Save the model in the database.
`census1.saveModel Models "This is the basic model, trained on an ODBC source." census1_odbc`
Note - To save the model in a text file, open a `Kxen.FileStore` and specify the file in which you want to save it.
6. Add either `delete census1` to free the memory and be able to create new models, or `quit`.

8.2.3 Using the Default Cutting Strategy: Random

Using the Default Cutting Strategy: Random

Now take a look at the way the data set is cut into estimation, validation and test data. There are two possibilities:

- specify three files that will be used as Estimation, Validation and Test respectively,
 - define a single file with the three roles and choose between the periodic and random cutting strategies. The default cutting strategy is random, which means that each record will be chosen to be part of the estimation, validation or test set depending on the value of a random number calculated for that record.
1. Create the model variable:
`createModel Kxen.SimpleModel census1`
 2. Open the data store:
`census1.openNewStore Kxen.FileStore`
 3. Define the training data set:
`census1.newDataSet Training Census01.csv`
 4. Read the description file for the Estimation data set:
`census1.readSpaceDescription Training desc_Census01.csv`
Note - Since all the data sets must have the same structure, the Estimation data set description is used for all the data sets.
 5. Load the model parameters:
`census1.getParameter ""`
 6. Set the minimum and maximum values between which the random number must be in order for the line to be attributed to the Estimation data set. In the case of a random cutting strategy, a random number is calculated for each record. According to its value, the line is attributed to estimation, validation or test. In this example, the record will be attributed to Estimation when the random number is between 0 and .7 (which should be the case for 70 % of the records approximately).
`census1.changeParameter DataSets/Estimation/Parameters/RandomMin 0`
`census1.changeParameter DataSets/Estimation/Parameters/RandomMax .7`
 7. Set the minimum and maximum values between which the random number must be in order for the line to be attributed to the Validation data set. In this example, a record will be in validation when the random number is between .7 and .9 (which should be the case for another 20 % of the records approximately).
`census1.changeParameter DataSets/Validation/Parameters/RandomMin .7`
`census1.changeParameter DataSets/Validation/Parameters/RandomMax .9`
 8. Set the minimum and maximum values between which the random number must be in order for the line to be attributed to the Test data set. The test data set is a part of the data from which the model does not learn, but on which its results are applied in order to see how it performs on completely new data.
`census1.changeParameter DataSets/Test/Parameters/RandomMin .9`
`census1.changeParameter DataSets/Test/Parameters/RandomMax 1`
 9. Validate the parameter changes:
`census1.validateParameter`
 10. Define the protocols to apply:
`census1.addTransformInProtocol Default Kxen.ConsistentCoder`
`census1.addTransformInProtocol Default Kxen.RobustRegression`
 11. Learn the model:
`census1.sendMode learn`
 12. Save the model with the same name and with a new automatic version number.
`census1.saveModel Models.txt "This is a model for which estimation was randomly 70% of the data set,`

validation was 20% and test was 10%." census1

13. Add either `delete census1` to free the memory and be able to create new models, or `quit`.

8.2.4 Changing the Cutting Strategy: Periodic

Changing the Cutting Strategy: Periodic

This script will change the cutting strategy to the periodic method. In this example, out of 10 lines, 7 lines will be used for Estimation, the 8th and 9th for Validation and the 10th for Test.

1. Create the model variable:
`createModel Kxen.SimpleModel census1`
2. Open the data store:
`census1.openNewStore Kxen.FileStore`
3. Define the training data set:
`census1.newDataSet Training Census01.csv`
4. Read the description file for the Estimation data set:
`census1.readSpaceDescription Training desc_Census01.csv`
Note - Since all the data sets must have the same structure, the Estimation data set description is used for all the data sets.
5. Load the model parameters:
`census1.getParameter ""`
6. Change the cutting strategy for each data set. In the tree representation of a model, there are three main branches, which are `Parameters`, `Protocols` and `Data sets`. The cutting strategy is located in the `Parameters` branch and has the name `CutTrainingPolicy`. It can be changed as follow:
`census1.changeParameter Parameters/CutTrainingPolicy periodic`
7. Set the following parameters to attribute a line to the Estimation data set when the modulo is between 0 and 7 (exclusive of 7), with a period of 10.
`census1.changeParameter DataSets/Estimation/Parameters/ModuloPeriod 10`
`census1.changeParameter DataSets/Estimation/Parameters/ModuloMin 0`
`census1.changeParameter DataSets/Estimation/Parameters/ModuloMax 7`
8. Set the following parameters to attribute a line to the Validation data set when the modulo is between 7 and 9 (exclusive of 9), with a period of 10.
`census1.changeParameter DataSets/Validation/Parameters/ModuloPeriod 10`
`census1.changeParameter DataSets/Validation/Parameters/ModuloMin 7`
`census1.changeParameter DataSets/Validation/Parameters/ModuloMax 9`
9. Set the following parameters to attribute a line to the Test data set when the modulo is 9, that is between 9 and 10 exclusive of 10, with a period of 10.
`census1.changeParameter DataSets/Test/Parameters/ModuloPeriod 10`
`census1.changeParameter DataSets/Test/Parameters/ModuloMin 9`
`census1.changeParameter DataSets/Test/Parameters/ModuloMax 10`
10. Validate the parameters in order for the changes to take effect:
`census1.validateParameter`
11. The rest of the script does not change. The name of the model being the same (`census1`), it will be saved as `census1` version 2 if there was already one saved.
`census1.addTransformInProtocol Default Kxen.ConsistentCoder`
`census1.addTransformInProtocol Default Kxen.RobustRegression`

```
census1.sendMode learn
```

```
census1.saveModel Models.txt "This is a model for which, out of every 10 lines, 7 were used as  
estimation, 2 as validation and 1 as test." census1
```

12. Add either `delete census1` to free the memory and be able to create new models, or `quit`.

8.2.5 Changing the Compression Parameter

Some variables are defined as nominal, which means that their values represent categories. In that case the coding engine (SAP Predictive Analytics Modeler - Data Encoding) analyzes how relevant the categories are and 'compresses' them by grouping together categories that have the same behavior regarding the target and by creating a 'KxOther' category for unimportant categories. This script shows how to disable this feature of the Data Encoder.

1. Create a new model named `census2` and define a training data set with its description:

```
createModel Kxen.SimpleModel census2  
census2.openNewStore Kxen.FileStore .  
census2.newDataSet Training Adult01.csv  
census2.readSpaceDescription Training desc_Adult01.csv
```
2. Add the the encoding protocol with a specific name to be able to change its parameters. The syntax of the `addTransformInProtocol` function allows giving a symbolic name to a transform as shown below:

```
census2.addTransformInProtocol Default Kxen.ConsistentCoder myK2CTransform  
census2.addTransformInProtocol Default Kxen.RobustRegression
```
3. Load the parameters of the `Kxen.ConsistentCoder` transform:

```
myK2CTransform.getParameter ""
```
4. Modify the compression parameter called `Compress`, located in the `Parameters` branch of its sub tree:

```
myK2CTransform.changeParameter Parameters/Compress false
```
5. Validate the change by entering the command:

```
myK2CTransform.validateParameter
```
6. The rest of the script does not change:

```
census2.sendMode learn  
census2.saveModel Models.txt "This is a model for which compression has been disabled." census2
```
7. To complete the script, add either `delete census2` to free the memory and be able to create new models, or `quit`.

8.2.6 Excluding a Variable and Changing the Target

This script shows how to exclude a variable and set the target variable. In a model, each variable must have role one of the following roles: `input`, `skip`, `target` or `weight`. In order to exclude a variable, its role must be set to `skip`. To select the target, the variable role must be set to `target`. By default, all variables have the `input` role except for the last one, which has the `target` role.

Note - Remember that the target has to be numeric and either nominal with exactly two values (binary), or continuous.

1. Create a new model named `census3`.

```
createModel Kxen.SimpleModel census3
```

2. Open a new store and read the data and description files:
`census3.openNewStore Kxen.FileStore .`
`census3.newDataSet Training Adult01.csv`
`census3.readSpaceDescription Training desc_adult01.csv`
3. Load the tree containing the properties of the variables, including their roles. This tree is located under `Protocols/Default/Variables/<the name of the variable>.`:
`census3.getParameter ""`
4. Change the role of the variable `occupation` from `skip` (SAP Predictive Analytics is case-sensitive, therefore the case of the letters is important):
`census3.changeParameter Protocols/Default/Variables/occupation/Role skip`
5. Changing the target involves changing the role of the previous target (`class`) to `input` or `skip` and the role of the new target (`age`) to `target`:
`census3.changeParameter Protocols/Default/Variables/class/Role skip`
`census3.changeParameter Protocols/Default/Variables/age/Role target`
6. Validate the changes:
`census3.validateParameter`
7. The rest of the script does not change:
`census3.addTransformInProtocol Default Kxen.ConsistentCoder`
`census3.addTransformInProtocol Default Kxen.RobustRegression`
`census3.sendMode learn`
`census3.saveModel Models.txt "For this model, the occupation was manually excluded from the predictors and the target was changed from 'class' to 'age'" census3`
8. To complete the script, add either `delete census3` to free the memory and be able to create new models, or `quit`.

8.2.7 Applying a Model Using Text Files (Scoring)

Applying a Model Using Text Files (Scoring)

We are now going to show how to apply a model which has just been created or which we restore from the disk to a new data set. (In fact, for this tutorial, we will use the same data set but it would be the same method for another file.)

Since we already showed how to create a new model, we are going to try to restore one of the models we have created.

1. Create a store variable of type `Kxen.FileStore` in order to be able to open a file:
`createStore Kxen.FileStore myStore`
2. Open the current directory:
`myStore.openStore`
3. Load the first version of the model `census1` previously saved in the current directory:
`myStore.restoreModelID census1 1 model`
 The `1` between `census1` and `model` refers to the version number; each time a model is saved with the same name as a previous one, the version number is incremented.
4. Open the store containing the application data set:
`model.openNewStore Kxen.FileStore .`
5. Define the application input data set:
`model.newDataSet ApplyIn Census01.csv`

The `ApplyIn` data set is the one that contains the predictors. Although the target value is generally unknown for an `ApplyIn` data set, the field should be present (for example a question mark can replace the unknown value).

6. Define the application output data set:

```
model.newDataSet ApplyOut scoring.txt
```

The `ApplyOut` data set is a new file that will be generated by default with three fields for each record:

- the row number or record key if one was specified,
- the target value in the `ApplyIn` data set (usually a question mark)
- the score attributed to the record.

The score is the original output of the model. The scores allow sorting the records by probability of having a target value of 1, but they are not the probabilities themselves.

7. To output the probabilities as well, you need to change a parameter of the `Kxen.RobustRegression` transform located in the protocol: `'ExtraMode'`. To do that, you need a variable referring to the `Kxen.RobustRegression` transform. The solution is to 'bind' the `Kxen.RobustRegression` transform to a symbolic name. This transform belongs to the `'Default'` protocol and is the furthest from the data (the last one we 'added' or the first one we 'pushed'.) Therefore it has the index 0.

```
model.bind TransformInProtocol Default 0 myK2RTransform
```

The name `myK2Rtransform` now refers to the appropriate transform.

8. Change the parameter that governs the output in the `ApplyOut` file.

```
myK2RTransform.changeParameter Parameters/ExtraMode "Min Extra"
```

9. Validate the changes:

```
myK2RTransform.validateParameter
```

10. The model is now ready to apply the transforms it had calculated to the new data and print the result in the specified output file. Launch the process by entering:

```
model.sendMode apply
```

11. To complete the script, add either `delete model` and `delete myStore` to free the memory and be able to create new models, or `quit`.

8.2.8 Applying a Model in an ODBC Source (Scoring)

Applying a Model in an ODBC Source (Scoring)

This script will do exactly the same thing as the previous one, except that the data will come from an ODBC source. The scoring file will be a table in the database as well.

1. Create a store variable of the type `Kxen.ODBCStore` in order to be able to open a table:

```
createStore Kxen.ODBCStore myStore
```
2. Open the database:

```
myStore.openStore Database UserName password
```
3. Let's assume that a model with the name `census1_odbc` has been saved in `Database`, load it in the variable `model`:

```
myStore.restoreModelID census1_odbc 1 model
```
4. Open the store containing the application data set:

```
model.openNewStore Kxen.ODBCStore Database UserName password
```
5. Define the application input data set:

```
model.newDataSet ApplyIn Census01.csv
```

The `ApplyIn` data set is the one that contains the predictors. Although the target value is generally unknown for an `ApplyIn` data set, the field should be present (for example a question mark can replace the unknown value).

6. Define the application output data set:
`model.newDataSet ApplyOut scoring`
7. As in the previous script, you will output the probabilities as well as the scores:
`model.bind TransformInProtocol Default 0 myK2RTransform`
`myK2RTransform.changeParameter Parameters/ExtraMode "Min Extra"`
8. Validate the changes:
`myK2RTransform.validateParameter`
9. The model is now ready to apply the transforms it had calculated to the new data and print the result in the specified output file. Launch the apply process:
`model.sendMode apply`
10. To complete the script, add either `delete model` and `delete myStore` to free the memory and be able to create new models, or `quit`.

8.3 Segmentation

8.3.1 Basic Script Using Text Files

This script trains a clustering model on a training data set. The first steps are the same as for a regression/classification model: create a model (still of type `Kxen.SimpleModel`) and define a training data set or three data sets (estimation, validation and test sets).

1. Create a model variable named `census4`:
`createModel Kxen.SimpleModel census4`
2. Open a new store to be able to load the data set and its description:
`census4.openNewStore Kxen.FileStore .`
3. Specify the data set's role as `Training`:
`census4.newDataSet Training Census01.csv`
The data set is automatically split into Estimation and Validation.
4. Read the description of the data set:
`census4.readSpaceDescription Training desc_Census01.csv`
5. Define the coding transform:
`census4.addTransformInProtocol Default Kxen.ConsistentCoder`
6. Define the segmentation transform with a symbolic name to be able to modify its parameters:
`census4.addTransformInProtocol Default Kxen.SmartSegmenter myKMeansTransform`
7. Load the parameter tree of the `myKMeansTransform` object:
`myKMeansTransform.getParameter ""`
8. Set the number of clusters to 10. The number of clusters is a parameter of the `Kxen.SmartSegmenter` transform.
`myKMeansTransform.changeParameter Parameters/NbClusters 10`
9. Validate the changes:
`myKMeansTransform.validateParameter`

10. Start the learning process:
`census4.sendMode learn`
11. Save the model in the current directory, which is the last open store.
`census4.saveModel ClusteringModels.txt "This is the first clustering model." census4`
12. To complete the script, add either `delete census4` to free the memory and be able to create new models, or `quit`.

8.3.2 Basic Script Using an ODBC Source

This script trains the same clustering model, but using an ODBC source and saving the model in the database.

1. Create a model named `census4`:
`createModel Kxen.SimpleModel census4`
2. Open a new ODBC store to be able to load the data set table and its description:
`census4.openNewStore Kxen.ODBCStore Database UserName password`
3. Specify the data set's role as `Training`:
`census4.newDataSet Training Census01`
4. Read the description of the training data set:
`census4.readSpaceDescription Training desc_Census01`
5. Add the data encoding transform:
`census4.addTransformInProtocol Default Kxen.ConsistentCoder`
6. Add the segmentation transform with a symbolic name to be able to modify its parameters:
`census4.addTransformInProtocol Default Kxen.SmartSegmenter myKMeansTransform`
7. Load the parameters of the `Kxen.SmartSegmenter` transform:
`myKMeansTransform.getParameter ""`
8. Set the number of cluster to 10:
`myKMeansTransform.changeParameter Parameters/NbClusters 10`
9. Validate the changes:
`myKMeansTransform.validateParameter`
10. Start the learning process:
`census4.sendMode learn`
11. Save the model with the name `census4_odbc` in the table `ClusteringModels`:
`census4.saveModel ClusteringModels "This is the first clustering model trained with an ODBC source." census4_odbc`
12. To complete the script, add either `delete census4` to free the memory and be able to create new models, or `quit`.

8.3.3 Using a Classification Model to Characterize a Cluster

On a clustering problem, the basic approach is to create a clustering model using SAP Predictive Analytics Modeler - Segmentation/Clustering. Nevertheless, the available definitions of the clusters may not be entirely satisfactory and the clusters may be difficult to understand.

An interesting additional approach could be to run a classification model (with SAP Predictive Analytics Modeler - Regression/Classification) afterwards on the data with the cluster we are interested in as target. This

means first creating a clustering model, then applying it to the data to have the cluster number for each record, and then running a classification model on the data with a given cluster as target. Fortunately it is possible to do this automatically with KxShell and ODBC.

The following section describes the general methodology to be used and illustrates it below with the Adult Census data:

1. First, you need to build the clustering model using SAP Predictive Analytics Modeler - Segmentation/ Clustering and apply it to your data in order to have a table with the cluster number for each record.
2. Then you need to join our data and this table. Assuming that our data set is stored in a table named `Dataset`, has an key field named `ID`, and that the result of the model application has been saved in the table named `Clustering`, the following SQL statement (where the 10th cluster is used as example) can be used:

```
SELECT Dataset.*, target = CASE kc_clusterId WHEN 10 THEN 1 ELSE 0 END FROM Dataset, Clustering
WHERE Dataset.ID = Clustering.ID
```

Note - The `KeyLevel` of the `ID` field must be set to 1 in the description file of the data set to be able to be used by SAP Predictive Analytics as key in the output table. Otherwise, a field called `KxIndex` is created and used to reference the records.
3. Finally, you have to create a new description file for this view, which is the same as for your data set, except that it has an additional line for the variable 'target' that you created in the `SELECT` statement (name: 'target', storage: 'number', value: 'nominal').
4. You are then ready to run a classification model on the view created with the SQL statement, and see how the regression engine characterizes it.

The following script is the application of this methodology to the Adult Census data to characterize the 10th cluster.

For practical purposes, let's suppose that:

- An `ID` field has been added to the `Adult01` table, which has been renamed `Adult01_ID`.
- In the `desc_Adult01` description table, the `KeyLevel` of the `ID` field has been set to 1 and the table has been renamed `desc_Adult01_ID`.
- A description for the SQL `SELECT` statement that will be used has been prepared and named `desc_Cluster`. It is the same as in point 2, except that it has an additional line for the `target` variable.
- The database supports the `CASE` statement.
Note - Contrary to the other scripts in this document that use ODBC, this one has not been tested with the Access ODBC driver because Access does not support the `CASE` SQL function.

This script shows how to build a clustering model, apply it to a data set -either the same one or a new one, and to automatically characterize one of the clusters using SAP Predictive Analytics Modeler - Regression/ Classification.

1. Create a model with the variable name `adult_id`:
`createModel Kxen.SimpleModel adult_id`
2. Open an ODBC store:
`adult_id.openNewStore Kxen.ODBCStore Database UserName password`
3. Load the data set and its description:
`adult_id.newDataSet Training Adult01_ID`
`adult_id.readSpaceDescription Training desc_Adult01_ID`
4. Add the transforms in the protocol:
`adult_id.addTransformInProtocol Default Kxen.ConsistentCoder`
`adult_id.addTransformInProtocol Default Kxen.SmartSegmenter myKMeansTransform`

5. Load the parameters of the `Kxen.KMeans` transform:
`myKMeansTransform.getParameter ""`
6. Set the number of clusters:
`myKMeansTransform.changeParameter Parameters/NbClusters 10`
7. Validate the change:
`myKMeansTransform.validateParameter`
8. Exclude the `ID` field since it is not a predictor:
`adult_id.getParameter ""`
`adult_id.changeParameter Protocols/Default/Variables/ID/Role skip`
`adult_id.validateParameter`
9. Start the learning process:
`adult_id.sendMode learn`
10. Apply the model to the data set to get the number of the cluster to which each record belongs:
`adult_id.newDataSet ApplyIn Adult01_ID`
`adult_id.newDataSet ApplyOut Clustering`
`adult_id.sendMode apply`
11. Load the corresponding description, which has been prepared with the name `desc_Cluster`:
`classif_cluster.readSpaceDescription Training desc_Cluster`
12. Add the coding and regression transforms:
`classif_cluster.addTransformInProtocol Default Kxen.ConsistentCoder`
`classif_cluster.addTransformInProtocol Default Kxen.RobustRegression`
13. Exclude the variables `ID` and `class` so that they will not be used as predictors:
`classif_cluster.getParameter ""`
`classif_cluster.changeParameter Protocols/Default/Variables/ID/Role skip`
`classif_cluster.changeParameter Protocols/Default/Variables/class/Role skip`
`classif_cluster.validateParameter`
14. The model is ready to start the learning process, and give us the characterization of the 10th cluster:
`classif_cluster.sendMode learn`
15. Save the model:
`classif_cluster.saveModel classif_cluster.txt "This model helps characterize the 10th cluster built by Smart Segmenter."`
16. You can ask for the attributes contribution, the categories' importance and so on to get an idea of the 10th cluster.

8.3.4 Applying a Model Using Text Files

This script is the equivalent of the scoring script presented for regression/classification. Its goal is to load a model and apply it to a new data set (in practice the same one) and ask the model to determine to which cluster each record belongs.

1. Create a store variable of type `Kxen.FileStore` in order to be able to open a file:
`createStore Kxen.FileStore myStore`
2. Open the current directory:
`myStore.openStore .`
3. Load the first version of the model `census4` previously saved in the current directory:
`myStore.restoreModelID census4 1 model`

The 1 between `census1` and `model` refers to the version number; each time a model is saved with the same name as a previous one, the version number is incremented.

4. Open the store containing the application data sets:
`model.openNewStore Kxen.FileStore .`
5. Define the application input data set:
`model.newDataSet ApplyIn Census01.csv`
The `ApplyIn` data set is the one that contains the predictors. Although the target value is generally unknown for an `ApplyIn` data set, the field should be present (for example a question mark can replace the unknown value).
6. Define the application output data set:
`model.newDataSet ApplyOut clustering.txt`
The `ApplyOut` data set is a new file that will be generated by default with three fields for each record:
 - the row number or record key if one was specified,
 - the target value in the `ApplyIn` data set (usually a question mark),
 - the cluster number attributed to the record.The model is now ready to apply the transforms it has calculated to the new data and print the result in the specified output file.
7. Launch the apply process:
`model.sendMode apply`
8. To complete the script, add either `delete model` and `delete myStore` to free the memory and be able to create new models, or `quit`.

8.3.5 Applying a Model in an ODBC Source

Applying a Model in an ODBC Source

This script is the equivalent of the previous one for ODBC. The goal of this script is to open the model saved with the second script of this section (*Basic Script Using an ODBC Source*) and apply it to build a table containing the cluster numbers for each record.

1. Create a store variable of type `Kxen.ODBCStore` in order to be able to open a file:
`createStore Kxen.ODBCStore myStore`
2. Use this store variable to open the `Database`:
`myStore.openStore Database UserName password`
3. Load the clustering model named `census4_odbc`:
`myStore.restoreModelD census4_odbc 1 model`
4. Open the store containing the application data sets:
`model.openNewStore Kxen.ODBCStore Database UserName password`
5. Define the application input data set:
`model.newDataSet ApplyIn Census01`
6. Define the application output data set:
`model.newDataSet ApplyOut Clustering`
The model is now ready to apply the transforms it has calculated to the new data and insert the result in the specified output table.
7. Launch the apply process:
`model.sendMode apply`

- To complete the script, add either `delete model` and `delete myStore` to free the memory and be able to create new models, or simply `quit`.

8.4 KxCORBAShell

Additionally to the standard KxShell interpreter, another interpreter is included to be able to run KxShell scripts in a client-server environment.

During SAP Predictive Analytics installation, a sub folder named `KxCORBAShell` is created at the same level as `KxShell`.

`KxCORBAShell` contains two files:

- `KxCORBAShell`, the actual executable.
Note - Under Windows, it is named `KxCORBAShell.exe`.
- `KxCORBAShell.sh` or `KxCORBAShell.bat`. A shell script that will execute `KxCORBAShell` with the following default values as arguments:
 - Remote Server: `localhost`
 - Connection Port: `12345`
 - Service Name: `FactoryEntries3`

KxCORBAShell options

`KxCORBAShell` uses the same options and the same syntax as `KxShell`. However some additional options are provided to specify the connection to the Remote server:

- `-ORBInitRef`
This option can be used to specify the physical server (`<RemoteHostName>`) and the default port (`<RemotePort>`) used by SAP Predictive Analytics Server.
Syntax: `-ORBInitRef NameService=corbaname::<RemoteHostName>:<RemotePort>`
Example: `-ORBInitRef NameService=corbaname::kxserv:12345`
- `-Service Name`
This option can be used to specify the name of the logical service name used by the KXEN server, if it has been changed at the installation by the administrator (this is only to be used if several SAP Predictive Analytics Server have to be started on the same physical machine).
Syntax: `-ServiceName <RemoteServiceName>`
Example: `-ServiceName FactoryEntries3`
- `-authenticated`
This option must be specified if the SAP Predictive Analytics Server is in "Authenticated mode", which means that a proper authentication is required to connect to the server.
Syntax: `-authenticated`
- `-user`
This option must be used in the case of an Authenticated Server to provide the user name to be used to connect to the server. The user policy depends on the actual SAP Predictive Analytics installation, but most of the time, it must be a valid user name for the Server's Operating System.

Syntax: `-user <UserName>`

Example: `-user ustat1`

- `-password`

This option must be used in the case of an Authenticated Server to specify the password associated with the user specified by the '-user' option.

Syntax: `-password <Password>`

Example: `-password xbkxenU1`

Customize the Script

Copy `KxCORBAShell.sh` (or `.bat` for Windows) then update the copy to reflect your installation `RemoteServerName`, `RemoteServerPort` and `ServiceName`.

9 Integrating with the Data Access API

Learn how to use the Data Access API, which is the way for integrators and OEMs to extend how SAP Predictive Analytics accesses external data.

Some integration or operational environments have proprietary data storage. For example, presentation tools use their own internal layer to access data on many platforms and OLAP tools have their own internal way of storing their data. In such cases, it can be useful to provide integrators with a solution to connect SAP Predictive Analytics to their internal storage. This requires specifying a data access API that should be implemented by the integrators. It can be useful for programming an additional data driver.

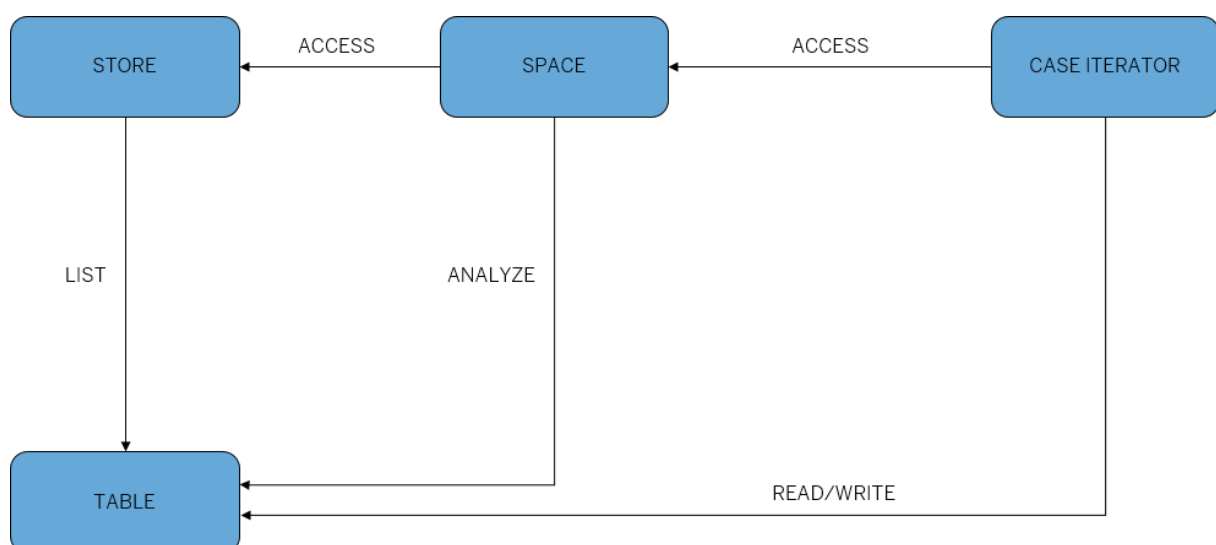
i Note

Integrators must implement such extensions in C. This language is used for stability reasons, because C++ name mangling is not yet very stable in many environments. Even written in C, the functions defined in the API can be viewed as methods defined for three classes: store, space and case iterator.

9.1 Architecture Elements

In SAP Predictive Analytics, data access is done through an abstraction layer that is decomposed under the main classes of Store, Space, and Case Iterator.

This image is interactive. Hover over areas for a short description.



- [Store \[page 23\]](#)

- [Space \[page 23\]](#)
- [Case Iterator \[page 25\]](#)

9.2 Integration of a User-Defined Data Storage

There are two specializations of the abstraction layer:

- FileStore, FileSpace, FileCaselster
- ODBCStore, ODBCspace, ODBCCaselster

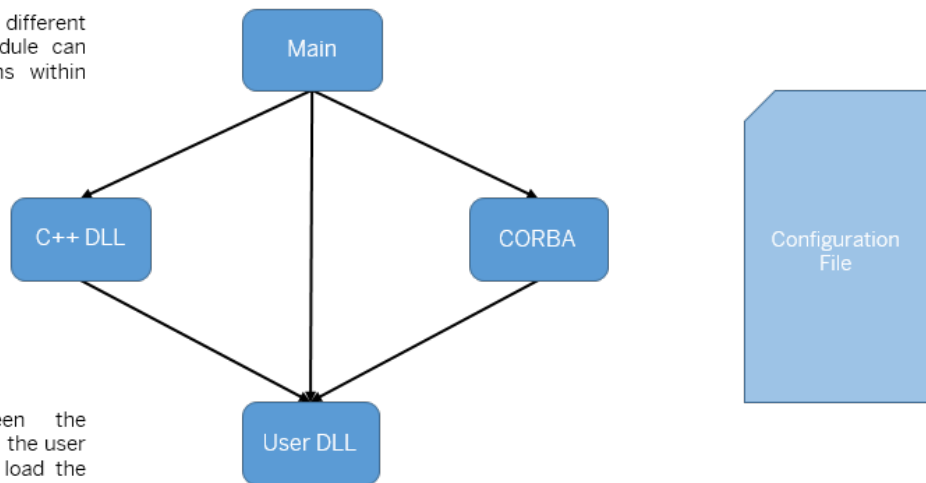
These classes allow SAP Predictive Analytics to access data from sources that can be text files with separators, or tables or SQL select statements accessible through an ODBC driver. In order to allow integrators to define their own data access functions, the following third set of classes is available:

- UserStore, UserSpace, UserCaselster

Together, these classes are not enough to run the data access. C++ wrappers are used to call functions written in a dynamic loadable library. The internal architecture allows you to create several data access types.

The final architecture of the solution is shown below:

The lines between the different modules mean that a module can load and activate functions within another module.



The direct line between the integration environment and the user DLL means it can directly load the user DLL wrapping the data access (for example, to initialize the resources within the library).

i Note

The functions used to perform these initial operations cannot be described in this document, as they are depending on each integrator environment. In the case where some of these initializations must be done, it is required that the integration environment loads the library and initializes it, because SAP Predictive Analytics would load it without running the proper initialization.

Main Integration Steps

The process of integrating a user-defined data access is the following:

Step	Description
Implement the dynamic loadable library.	Minimum Implementation [page 231]
Install the dynamic loadable library so that the run-time of SAP Predictive Analytics components can load it (in most environments, libraries are looked for in a set of predefined locations).	Library Installation [page 235]
Declare the new user class name associated with its library in the configuration file and its configuration options.	Declaration of a New User Class Name [page 235]

9.3 Minimum Implementation

The DLL to be implemented is specified through the `KxDataAccess.h` header file.

This section presents the minimum implementation required to perform the first tests of integration. These minimum requirements are decomposed for store, space, and case iterator. It also shows how a first running implementation can be further refined.

9.3.1 Space

- `Space_Cst`

```
KXDA_DLLEXPORT extern void * Space_Cst (void);
```

This function returns a handle (implemented as a void*) on a newly created space. A classical implementation could return a memory pointer to a C++ class kept inside the memory space managed by the library written by the integrator.

- `Space_Dst`

This function deletes a space and the memory location associated with this space. This function does not have to close the space. SAP Predictive Analytics components will close the space before (see below).

```
KXDA_DLLEXPORT extern void Space_Dst (void *iUserThis);
```

- `Space_Open`

This function opens a space within a store at a given location. For a directory, `iOpen` would be the name of a file in the directory, for an ODBC source, it would be the name of the view, or the table or a complete select statement within the specified ODBC source. In a specific implementation `iOpen` can be any of the logical name that allows the system to run a query to extract from or put back in the user internal storage. The string `iMode` can be either "r" for space opened in read mode, and "w" for space opened in write mode.

```
KXDA_DLLEXPORT extern KxenResult Space_Open (void * iUserThis,  
void * iUserStoreThis,
```

```
const char * iOpen,  
const char * iMode,  
const char * iTestMode);
```

- `Space_Close`
This function closes a previously opened space.

```
KXDA_DLLEXPORT extern KxenResult Space_Close (void *iUserThis);
```

i Note

The proposed level of the API does not make any assumption about where the actual connection to the data source will be performed. Let us take the analogy with the ODBC connection, and assume that we want to implement a user data access through ODBC by developing a user data access library. It is up to the user design, taking into consideration the concurrent data access problems, to actually open an ODBC connection at the store level when performing the `Store_Open`, the space level when performing the `Space_open`, or simply at the case iterator level, when performing the `begin`.

9.3.2 Store

- `Store_Cst`
This function returns a handle (implemented as a `void*`) on a newly created store. A classical implementation could return a memory pointer to a C++ class kept inside the memory space managed by the library written by the integrator.

```
KXDA_DLLEXPORT extern void * Store_Cst (void);
```

- `Store_Dst`
This function deletes a store and the memory location associated with this store. This function does not have to close the store. SAP Predictive Analytics components will close the store before.

```
KXDA_DLLEXPORT extern void Store_Dst (void * iUserThis);
```

- `Store_Open`
This function opens a store at a given location. For a directory, `iOpen` would be the path name of the directory, for an ODBC source, it would be the ODBC source logical name as seen on the machine running the SAP Predictive Analytics components. In some cases, the environment can provide a user name and a password to check access rights. The result of this operation is either `KXDA_OK` if successful, or `KXDA_FAIL` is failure.

```
KXDA_DLLEXPORT extern KxenResult Store_Open (void * iUserThis,  
const char * iOpen,  
const char * iUser,  
const char * iPassword);
```

- `Store_Close`
This function closes a previously opened store. When used from SAP Predictive Analytics components, all spaces opened from this store have been previously closed, so the implementation of this function does not have to check for opened spaces within that store.

```
KXDA_DLLEXPORT extern KxenResult Store_Close (void * iUserThis);
```


i Note

The class `cUserStore` keeps an internal flag to indicate if the store is currently open or closed. The calls to `Store_Open` or `Store_Close` should not break the expected state diagram for these operations.

Sometimes, the integration environment does not have, in its original design, an object corresponding to the notion of store. In this case, the integrator can create an empty C++ class with no method associated, any call to `Store_Open` with any `iOpen` will return success, and all the implementation will focus on the notion of space and case iterator. Having several stores is only important when the integrator wants to save models within its own internal storage. In SAP Predictive Analytics design, any store can/should contain a specific space called "KxAdmin" that holds information about the actual locations where models are stored within this store, it is then important that the SAP Predictive Analytics components can retrieve the models description using this name (KxAdmin can be reconfigured with another name through the configuration file).

This minimum implementation does not allow a graphical interface to present to the user the names of the stores that can be opened by the user. Most of the graphical user interfaces create an empty store, and ask to open this store with an `iOpen` equals to the empty string ("") in order to get the list of possible stores. An advanced implementation can use this feature but is not required at the beginning.

9.3.3 Case Iterator

- `CaseIter_Cst`

This function returns a handle (implemented as a `void*`) on a newly created case iterator. A classical implementation could return a memory pointer to a C++ class kept inside the memory space managed by the library written by the integrator.

```
KXDA_DLLEXPORT extern void * CaseIter_Cst (void *iSpaceThis);
```

- `CaseIter_Dst`

This function deletes a case iterator and the memory location associated with this case iterator.

```
KXDA_DLLEXPORT extern void CaseIter_Dst (void * iUserThis);
```

```
KXDA_DLLEXPORT extern KxenResult CaseIter_Begin_GetNumberOfColumns  
( void * iUserThis,  
  unsigned long * iNumberOfColumnsPtr);
```

9.3.4 Refinement Steps

Refining the Space

There are some types of user-defined spaces that know the storage of each column (variable, dimension). When this is the case, SAP Predictive Analytics components can ask for this description instead of using a default algorithm that will force a case iterator of cells of strings in order to derive this information from the first 100 lines of the space.

Some types of user-defined space can even go further and can access to a metadata repository where all information (the value type of each column between ordinal, nominal and continuous, the code to represent missing values, the fact that a column can be used to sort the lines) about variable description is stored and forward this to the SAP Predictive Analytics components.

Example

```
KXDA_DLLEXPOR extern KxenResult CaseIter_GetDllSupportValueType (void *
iUserThis, unsigned long * iDllSupportValueTypePtr);
KXDA_DLLEXPOR extern KxenResult CaseIter_Begin_GetTypeOfColumns (void *
iUserThis, unsigned long iNumberOfColumns, KxenStorageType * iTypesOfColumnsPtr);
KXDA_DLLEXPOR extern KxenResult CaseIter_GetDllSupportDescription (void *
iUserThis, unsigned long * iDllSupportDescriptionPtr);
KXDA_DLLEXPOR extern KxenResult CaseIter_Begin_GetDescOfColumns (void *
iUserThis, unsigned long iNumberOfColumns, KxenStorageType *
iTypesOfColumnsPtr);
```

Refining the Store

Available Stores

When the user-defined DLL is used from a graphical interface, you can provide the list of available stores that the application can open.

In this case, the user must provide a function call that returns the number of available stores and their descriptions. In order to initiate this process, user interfaces always ask for a store with an empty open string, and then it asks for the available stores in this stub store to have all the possible stores. When stores have a hierarchical structure, this mechanism can be used to browse all available stores.

Available Spaces

When the user-defined DLL is used from is used form a graphical interface, it can be nice to have the list of available spaces that the application can open from this store. In this case the user must provide a function that returns the number of available spaces (and then their description).

Using Internal Storage to Save and Restore Models

When the user wants to use its own internal storage to save and restore models, it has to provide a certain number of functions. First of all, it should be possible to open space in the write mode (to save the models) and to erase lines within a space in a store that has some keys equal to some values.

9.3.5 Compilation Process

The following table presents some of the compilers required to generate dynamic loadable libraries for different platforms. The compilers required are C compilers, as C language has been chosen for stability reasons.

Operating System	Compiler
Win64	CL
Linux	gcc

9.4 Library Installation

This section presents what to do in order for SAP Predictive Analytics to find a dynamic loadable library within different environments.

The DLL file should be installed somewhere so that the application will find it. Typically a path variable is searched for such DLL, or the directory where the initial application is located.

To configure a new DataAccess library, a call to `setConfiguration` is issued, for example:

```
lFactory.setConfiguration("UserStore", "MyAccess:XXXX");
```

This call tries to load the corresponding dynamic library, but the loading of such a library is done in an OS-dependent way. The following convention are used:

Operating System	Library Name	Search Variable	System Call Used
Win64	XXXX.dll	PATH	LoadLibrary()
Linux	XXXX.so or libXXXX.so	LD_LIBRARY_PATH	dlopen()

i Note

The Search Variable is the variable used by the OS to locate dynamic libraries.

On Linux systems, we try first to load `XXXX.so`, then `libXXXX.so`. So any of the two names is valid.

9.5 Declaration of a New User Class Name

This section presents what you must do to declare a new user-defined data access class within SAP Predictive Analytics components.

The user-defined data access dynamic library must be "declared" to SAP Predictive Analytics components environment. To do this, you must add a configuration entry, either in the configuration file loaded by the executable (for example, `KxShell.cfg` or `KxCORBA.cfg`), or using the `setConfiguration` call.

Here is an example of a configuration file with the last lines presenting declarations of new user classes:

Key	Value
MessageFile	../KxCORBA/KxTool_us.msg
MessageFile	../KxCORBA/KxTool_fr.msg
KxDesc	KxDesc
KxAdmin	KxAdmin
UserStore	MySpecialStore:MyLib
UserStoreOption.MySpecialStore.MultiLang uage	true

Once the configuration file is written, there are two ways of loading it. A default configuration file is always loaded at init time at the place where the executable code is present. Then, the user can force to load supplementary configuration files through the following commands.

This configuration entry's key should be "UserStore", and the value should be a string composed of 2 fields, separated by a ':' character:

- The first one is the symbolic name that will be attached in SAP Predictive Analytics components to this class of Store, for example `MySpecialStore`. This name is used for example in the class `Factory`, in `createInstance`, and by the model's function `openNewStore`.
- The second one is the actual name of the library, without any extension of system specific prefix, for example `MyLib`, but not `MyLib.dll`, neither `libMyLib.dll`.

Of course, you can have several such entries in a configuration file, or call several times the `setConfiguration` function.

Note

If the dynamic library cannot be loaded at when the configuration entry is set, it is currently silently ignored. Calls to the `getClassInfo` function will not report any information on such store, and calls to `createInstance` with this class name will fail.

Optionnally, a configuration option can be added to describe processing of charsets by the dynamic library. The key name is `UserStore.<My_Dynamic_Library>.MultiLanguage`. Possible values are:

- `False`: all strings returned or consumed by the dynamic library are encoded using the current OS's native charset. In such a case, the kernel applies its own UTF8 encoding/decoding. This is the default value of the option.
- `True`: all strings returned or consumed by the dynamic library are already encoded in UTF8, avoiding and encoding/decoding step to the kernel.

Example

```
UserStore.SasWindows7.MultiLanguage=true
```

10 Appendix

[Data Type Mapping \[page 237\]](#)

[File Format Specifications \[page 240\]](#)

[Language ISO Codes \[page 244\]](#)

10.1 Data Type Mapping

The data types of the Automated Analytics API are based on the CORBA scheme, which makes them language-independent. However, each type is mapped to a real type in each scheme. The following table shows the API data types with their corresponding language data types.

Note

Using the Java Common Interface layer or the Python scheme, objects are now returned without holder types. See the Java Common Interface API documentation for more information.

API Type	C++ Type	Java CORBA Type	JNI Type	Python Type
long (return)	long	int	int	long
in string	const char*	String	String	str
in unsigned long	unsigned long	int	int	int
in Object	void *	org.omg.CORBA.Object	com.kxen.KxModelJni.KXObject	Object
in long	long	int	int	Int
in boolean	bool	boolean	boolean	bool
in eKxenMode	KxModel::eKxenMode	com.kxen.KxModel.KxenModeTag	com.kxen.KxModelJni.KxenModeTag	aalib.Kxen_learn
in IKxenXXX	KxModel::IKxenXXX*	com.kxen.KxModel.IKxenXXX	com.kxen.KxModelJni.IKxenXXX	aalib.IKxenXXX
out boolean	bool &	org.omg.CORBA.BooleanHolder	com.kxen.KxModelJni.BooleanHolder	N/A

API Type	C++ Type	Java CORBA Type	JNI Type	Python Type
out string	char* &	org.omg.CORBA.StringHolder	com.kxen.KxModelJni.StringHolder	N/A
out Object	void *&	org.omg.CORBA.ObjectHolder	com.kxen.KxModelJni.ObjectHolder	N/A
out unsigned long	unsigned long &	org.omg.CORBA.IntHolder	com.kxen.KxModelJni.IntHolder	N/A
out IKxenXXX	KXModel::IKxenXXX* &	com.kxen.KxModel.IKxenXXXHolder	com.kxen.KxModelJni.IKxenXXXHolder	N/A

Example: C++

```
void printParameter( KxModel::IKxenModel* iModel,
                   const char* iParamPath ) {
    KxModel::IKxenParameter* lParam;
    HRESULT hr = iModel->getParameter( iParamPath, lParam );
    myProcessKxenResult( hr, "getParameter" );
    // result containers
    KxenString lName = 0;
    KxenString lValue = 0;
    KxenBool lReadOnly;
    hr = lParam->getNameValue( lName, lValue, lReadOnly );
    myProcessResult( hr, "getNameValue" );
    printf( "[%s] = [%s]\n", lName, lValue );
    KxModel::KxDelete( lName );
    KxModel::KxDelete( lValue );
    KxModel::KxDelete( lParam );
}
```

Example: C++ common Interface

```
void printParameter(KxCommonInterf::IkxenModel& iModel,
                   const char* iParamPath ) {
    KxCommonInterf::IKxenParameter * lParam =
        iModel.getParameter(iParamPath);
    // result containers
    KxCommonInterf::NameValue lNameValue = lParam->getNameValue();
    printf("[%s] = [%s]\n", lNameValue.mName, lNameValue.mValue);
    KxCommonInterf::KxDelete(lParam);
}
```

or

```
void printParameter(KxCommonInterf::IkxenModel& iModel,
                   const char* iParamPath ) {
    KxCommonInterf::cAutoPointer<KxCommonInterf::IKxenParameter>
        lParam(iModel.getParameter(iParamPath);
```

```

// result containers
KxCommonInterf::NameValue lNameValue = lParam->getNameValue();
printf("[%s] = [%s]\n", lNameValue.mName, lNameValue.mValue);
}

```

Example: Java/CORBA

```

import org.omg.CORBA.StringHolder;
import org.omg.CORBA.IntHolder;
import org.omg.CORBA.BooleanHolder;
import com.kxen.KxModel.*;
void printParameter(IKxenClassFactory iFactory,
                  IKxenModel iModel,
                  String iParamPath) {
    IKxenParameterHolder lParam = new IKxenParameterHolder();
    int hr = iModel.getParameter(iParamPath, lParam);
    myProcessKxenResult(hr, "getParameter");
    // result containers
    StringHolder lName = new StringHolder();
    StringHolder lValue = new StringHolder();
    BooleanHolder lReadOnly = new BooleanHolder();
    hr = lParam.value.getNameValue(lName, lValue, lReadOnly);
    myProcessResult(hr, "getNameValue");
    System.out.println "[" + lName.value + "] = [" + lValue.value + "]\n";
    iFactory.deleteInstance(lParam.value);
}

```

Example: Java/JNI

```

import com.kxen.KxModelJni.*;
void printParameter(IKxenClassFactory iFactory,
                  IKxenModel iModel,
                  String iParamPath) {
    IKxenParameterHolder lParam = new IKxenParameterHolder();
    int hr = iModel.getParameter(iParamPath, lParam);
    myProcessKxenResult( hr, "getParameter" );
    // result containers
    StringHolder lName = new StringHolder();
    StringHolder lValue = new StringHolder();
    BooleanHolder lReadOnly = new BooleanHolder();
    hr = lParam.value.getNameValue(lName, lValue, lReadOnly);
    myProcessResult(hr, "getNameValue");
    System.out.println "[" + lName.value + "] = [" + lValue.value + "]\n";
    iFactory.deleteInstance(lParam.value);
}

```

Example: Java Common Interface

```

import com.kxen.CommonInterf.*;
import com.kxen.CommonInterf.KxenParameter.NameValue;
void printParameter( KxenModel iModel, String iParamPath ) {

```

```

KxenParameter lParam = iModel.getParameter(iParamPath);
NameValue lNameValue = lParam.getNameValue();
System.out.println "[" + lNameValue.getName()
                    + "]" = [" + lNameValue.getValue() + "]\n";
lParam.release();
}

```

Example: Python

```

import aalib
def printParameter(model, paramPath):
    param = model.getParameter(paramPath)
    name_value = param.getNameValue()
    print("[%s] = [%s]" % (name_value.name, name_value.value))

```

10.2 File Format Specifications

The format of the internal files used in the modeling phases (space description, model space, and model description space). You can use it for setting up the database to support SAP Predictive Analytics features.

There are several files or tables that are directly read by Predictive Analytics for OEM components. Everything has been designed in the components to be saved/restored under column tables formats. Three specific space formats are predefined, these are:

Variable Description Space Format

This format is used to save a variable description. A description contains the information to describe from a meta-model point of view the variables contained into a dataset. These spaces are generally saved under a name KxDesc. This default name can be overloaded through a loadConfigurationFile of a store.

Each line of the KxDesc file contains information about one variable.

The different columns of these specific spaces are the following:

Name	Storage	Value	Key Level	Order Level	Comment
Rank	number	ordinal	1	0	Rank of the variable in the data file.
Name	string	nominal	0	0	The name of the variable of the associated dataset.
Storage	string	nominal	0	0	The storage of the variable. This string can only be: number, string, or date.

Name	Storage	Value	Key Level	Order Level	Comment
Value	string	nominal	0	0	The value type of this variable. This string can only be: nominal, ordinal or continuous.
KeyLevel	number	ordinal	0	0	Put a number different from 0 here if the variable can be considered as a key. Our convention is to put 1 for the primary key, 2 for the secondary key, and so on. This is used by the model when new values are generated, they are associated with all variables with a key not null. If no variable in a description file is declared with a key, KXEN Components automatically generates an KxIndex variable with the row index and declares it as a key.
OrderLevel	number	ordinal	0	0	Put a number different from 0 here if the variable can be considered as a natural sort order for the cases. Example of such a variable is a date, or sometimes a customer id. When such variable is present, the final version of the KXEN components should be able to use this information to better cut training datasets into estimation, validation and test datasets.
MissingString	string	nominal	0	0	This string indicates a specific code for missing value. the default value is the null length string, which is used most of the times. Even if ODBC access can return empty values as such, sometimes, there is a specific code for missing values in old data base systems or architecture such as 99 for two digit numbers, or "unknown" for some categorical information.

Name	Storage	Value	Key Level	Order Level	Comment
Group	string	nominal	0	0	Sometimes, some variables belongs to a single semantic group. Think of disjunctive coding of a categorical variables for example, all of these variables can take boolean values, but when one of these columns is activated all the others are de-activated: in this case, make all these variables belong to the same group. This would forbid transforms (and thus models) to take into account cross studies between variables belonging to the same group.
Description	string	nominal	0	0	The comment describes this variable.

Model Description Space Format

This format is used to save description of models that are saved into a single store (a store is either an ODBC source or a file directory). These descriptions contain all information that could be wanted by a user to restore a previously saved model in this store.

Each line of the KxAdmin file contains one model description.

One model can be associated with several lines (one for each saved version).

Name	Storage	Value	KeyLevel	OrderLevel	Comment
Name	string	nominal	1	0	The name of model associated with this model description.
Class	string	nominal	0	0	The name of the class of the model. In the current release, there is only one possible model class name which is Kxen.SimpleModel, but there will be very soon two other classes: Kxen.Classification and Kxen.Regression. The main differences between these three classes is the packaging of the results of the different transforms.
Class version	number	ordinal	0	0	The version of the class of the saved model. This can be used internally to restore very old formats

Name	Storage	Value	KeyLevel	OrderLevel	Comment
Space Name	string	nominal	0	0	The name of the space in which this version of the model has been saved. Of course the space name is relative to the store in which this KxAdmin is found (be it a ODBC source or a directory). All versions of a model are saved into a single space (this constraint could be removed in the next KXEN Components version).
Version	number	ordinal	2	0	The version of the model. This is initialized with 1 for the first initial save order of this model, and then increased by one for every commit of the model into the same store (and as explained earlier in the same space).
CreationDate	string	nominal	0	0	The date at which the model has been saved in this version.
Comment	string	nominal	0	0	The comment describing this version of the model.

Model Parameter Space Format

This format is used to save models. In order to perform this operation, models are first converted into their parameter hierarchy counterpart and then saved into a flat parameter file that is described here. This basic architecture would allow us to save individual transforms into separate spaces, but we do not allow this from a user perspective in order to ease the process of re-building a model (and internal dependencies) from different sources. It must be noted that this allow to have a self-contained view of the model.

Each line of this space contains information about one parameter in the parameter hierarchy of one model.

Name	Storage	Value	KeyLevel	OrderLevel	Comment
Name	string	nominal	1	0	The name of model in which belongs this parameter.
Version	number	ordinal	2	0	The version number of the model in which belongs this parameter.
Id	number	ordinal	3	0	The identifier of this parameter in the hierarchical structure. Is filled using a breadth-first mechanism.

Name	Storage	Value	KeyLevel	OrderLevel	Comment
Parent Id	number	ordinal	0	0	The identifier of the parent parameter for the current parameter. This is used to reconstruct the hierarchy when restoring a previously saved model. A parameter with a parent id 0 is a model root parameter.
Enum Flag	string	nominal	0	0	This flag indicates if the children of this parameter must be considered as enumerated values (one child parameter for each possible value), or as sub-directory parameters. Possible values are "TRUE" or "FALSE". We recognize that this can take extra space and could be optimized in a future release.
Param Name	string	nominal	0	0	The name of the current parameter.
Param Type	string	nominal	0	0	The type of the current parameter. In the current release, all internal parameters are saved as strings, so the only possible value here is "string", future release could accept also "number" or "date".
Param Value	string	nominal	0	0	The value of the current parameter.

Note

A parameter is defined, in this release, by both its model name, its model version, and its id. The read-only flag is not saved in the model space, as each model class can rebuild this information internally.

10.3 Language ISO Codes

ISO 639-1	Language Name
ab	Abkhazian
aa	Afar

af	Afrikaans
ak	Akan
sq	Albanian
sq	Albanian
am	Amharic
ar	Arabic
an	Aragonese
hy	Armenian
as	Assamese
av	Avaric
ae	Avestan
ay	Aymara
az	Azerbaijani
bm	Bambara
ba	Bashkir
eu	Basque
be	Belarusian
bn	Bengali
bh	Bihari
bi	Bislama
bs	Bosnian
br	Breton
bg	Bulgarian
my	Burmese
ca	Catalan; Valencian
ch	Chamorro
ce	Chechen

ny	Chichewa; Chewa; Nyanja
zh	Chinese
cu	Church Slavic; Old Slavonic; Church Slavonic; Old Bulgarian; Old Church Slavonic
cv	Chuvash
kw	Cornish
co	Corsican
cr	Cree
hr	Croatian
cs	Czech
da	Danish
dv	Divehi
nl	Dutch; Flemish
dz	Dzongkha
en	English
eo	Esperanto
et	Estonian
ee	Ewe
fo	Faroese
fj	Fijian
fi	Finnish
fr	French
fy	Frisian
ff	Fulah
gd	Gaelic; Scottish Gaelic
gl	Gallegan
lg	Ganda

ka	Georgian
de	German
el	Greek, Modern (1453-)
gn	Guarani
gu	Gujarati
ht	Haitian; Haitian Creole
ha	Hausa
he	Hebrew
hz	Herero
hi	Hindi
ho	Hiri Motu
hu	Hungarian
is	Icelandic
io	Ido
ig	Igbo
id	Indonesian
ia	Interlingua (International Auxiliary)
ie	Interlingue
iu	Inuktitut
ik	Inupiaq
ga	Irish
it	Italian
ja	Japanese
kl	Kalaallisut; Greenlandic
kn	Kannada
kr	Kanuri
ks	Kashmiri

kk	Kazakh
km	Khmer
ki	Kikuyu; Gikuyu
rw	Kinyarwanda
ky	Kirghiz
kv	Komi
kg	Kongo
ko	Korean
kj	Kuanyama; Kwanyama
ku	Kurdish
lo	Lao
la	Latin
lv	Latvian
li	Limburgan; Limburger; Limburgish
ln	Lingala
lt	Lithuanian
lu	Luba-Katanga
lb	Luxembourgish; Letzeburgesch
mk	Macedonian
mg	Malagasy
ms	Malay
ml	Malayalam
mt	Maltese
gv	Manx
mi	Maori
mi	Maori
mr	Marathi

mh	Marshallese
mo	Moldavian
mn	Mongolian
na	Nauru
nv	Navajo; Navaho
nr	Ndebele, South; South Ndebele
ng	Ndonga
ne	Nepali
se	Northern Sami
no	Norwegian
nn	Norwegian Nynorsk; Nynorsk, Norwegian
oc	Occitan (post 1500); Provençal
oj	Ojibwa
or	Oriya
om	Oromo
os	Ossetian; Ossetic
pi	Pali
pa	Panjabi; Punjabi
fa	Persian
pl	Polish
pt	Portuguese
ps	Pushto
qu	Quechua
rm	Raeto-Romance
ro	Romanian
rn	Rundi
ru	Russian

sm	Samoaan
sg	Sango
sa	Sanskrit
sc	Sardinian
sr	Serbian
sn	Shona
ii	Sichuan Yi
sd	Sindhi
si	Sinhalese
sk	Slovak
sl	Slovenian
so	Somali
st	Sotho, Southern
es	Spanish; Castilian
su	Sundanese
sw	Swahili
ss	Swati
sv	Swedish
tl	Tagalog
ty	Tahitian
tg	Tajik
ta	Tamil
tt	Tatar
te	Telugu
th	Thai
bo	Tibetan
ti	Tigrinya



to	Tonga (Tonga Islands)
ts	Tsonga
tn	Tswana
tr	Turkish
tk	Turkmen
tw	Twi
ug	Uighur
uk	Ukrainian
ur	Urdu
uz	Uzbek
ve	Venda
vi	Vietnamese
vo	Volapük
wa	Walloon
cy	Welsh
wo	Wolof
xh	Xhosa
yi	Yiddish
yo	Yoruba
za	Zhuang; Chuang
zu	Zulu

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.