Zementis Predictive Analytics Solutions Guide

10.7.0.2

Zementis Predictive Analytics

Solutions Guide

This document applies to Zementis Server 10.7.0.2 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

Zementis Server enables the agile deployment and integration of predictive decision services. It allows organizations to convert predictive models into operational services without requiring any additional custom coding by the information technology (IT) organization. Zementis Server ensures model integrity, optimizes performance and powers scaling as necessary.

This document serves as a guide for creating decision solutions using Zementis Server. It describes how Zementis Server components are used to verify and execute your advanced analytics either in real-time (against in-flight data) or batch mode (against data at rest). This guide also explains how the different Zementis Server components are combined to offer a powerful scoring framework.

It is important to note that Zementis Server leverages the Predictive Model Markup Language (PMML) standard. PMML handles data pre-processing and post-processing as well as the predictive model itself. In this way, the entire predictive workflow can be implemented in PMML.

Note

In respect of considerations concerning EU General Data Protection Regulation (GDPR), our product stores personal information in shape of a user ID for the purpose of logging in. The user ID can be created, modified and removed as described in the Zementis Server Deployment Guide. As the input data might contain sensitive personal information, please anonymize any such data to ensure that the processing of personal data is in accordance with the GDPR.

1.1. Decision Solutions Overview

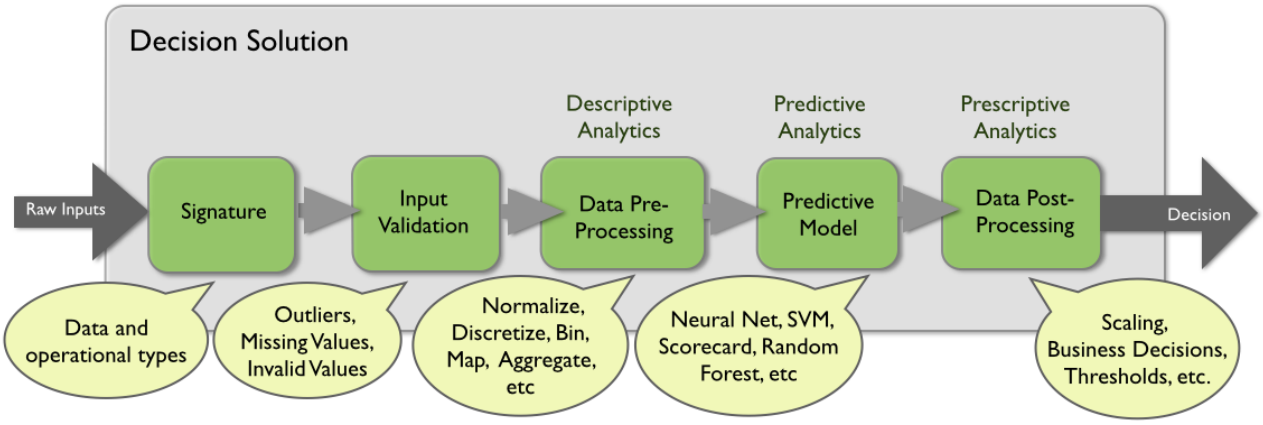
Zementis Server allows data-driven insight and expert knowledge to be combined into a single and powerful decision strategy through the use of PMML. Whereas *expert knowledge* encapsulates the logic used by experts to solve problems, *data-driven knowledge* is based on the ability to automatically recognize patterns in data not obvious to the expert eye. These two different types of knowledge are represented by two powerful technologies: Business Rules and Predictive Analytics. By bringing together both technologies, Zementis Server offers the best combination of control and flexibility for orchestrating critical day-to-day business decisions.

During the process of building a predictive model, there is usually quite a bit of data analysis and data pre-processing that need to take place. This is done to prepare the raw historical data so that it is suitable for model building and/or to combine and transform different data fields so that they create feature detectors that offer a richer predictive power than the input fields they were derived from. More often than not, such features entail looking at the data from a descriptive point of view as to explain it. For example, a feature detector may be defined as an aggregate value

of an input field containing the transaction amount as part of a credit card transaction. If this feature detector is the average transaction amount for the last month or week, the predictive model can use this information to generate a prediction that takes into account the delta between the current amount and the average past amount. The average amount for the last month is a typical case of descriptive analytics which tries to answer what happened in the past. Descriptive features are extremely important since when fed into a predictive model, they transform the nature of the information itself, allowing a model to answer what will happen next. Through PMML, Zementis Server is capable of capturing this process entirely. It also takes it a step further, by including prescriptive analytics into the mix, which is implemented by a series of post-processing steps expressed via the use of business rules.

Prescriptive analytics takes advantage of the outputs generated by a predictive technique by transforming them into business decisions. As depicted in Figure 1.1 the process of integrating descriptive, predictive and prescriptive analytics into a single solution is easy and straightforward with PMML and Zementis Server.

Figure 1.1. Decision Solution



With Zementis Server, the power of predictive analytics is made available to any other application in your enterprise via web services. Without further configuration or customization, a predictive model is exposed as a web service and seamlessly participates in the overall business process flow.

Zementis Server is the first technology solution that enables an enterprise to score data from any source in batch or real-time while combining the power of descriptive, predictive and prescriptive analytics by leveraging a well-supported open industry standard. By using Zementis Server and PMML both the human resource and technology requirement to deploy these powerful analytics is drastically reduced. At the same time, pace of deployment is increased and model integrity and quality is improved.

In this guide, we start by describing the process of building and testing a predictive model in Chapter 2. This is followed by a description of custom resources in Chapter 3. Chapter 4 gives an overview of the Zementis Server Java Extensions API. Finally, an extensive description of the web service capabilities is provided in Chapter 5.

Chapter 2. Predictive Models

The conversation around Big Data for both technologists and businesses has become pervasive. The challenge many enterprises and teams face is how to deliver measurable value from Big Data initiatives. By enabling rapid deployment from the Data Scientist's desktop to the operational IT environment, Zementis Server and PMML provide a standards-based methodology and process through which value from Big Data initiatives can be gained, quantified and demonstrated.

The predictive model building process begins by working with and developing a deep understanding of historical data which is mined for feature detectors. These are in turn used to build the predictive models. While a time consuming and laborious process, this provides the foundation for creating value from Big Data.

Building models is only the first step to realizing the benefits of predictive analytics. The second and final step is to actually use them within the overall business flow and processes. In other words, the models need to move from the data scientist's desktop into the enterprise operational IT environment where they can be used for scoring new data and drive business decisions.

Deployment of predictive models into the IT operational environment is all but straightforward. It can take as long as the data analysis phase itself or even longer and consume a significant amount of resources. It is not uncommon that by the time models are finally deployed, they are already stale and require to be refreshed with newer (historical) data reflecting a changing market.

Zementis Server makes deployment and use of complex predictive models trivial. Zementis Server has been designed from the ground up to consume, execute, optimize and scale Predictive Models that have been saved in PMML. PMML is the standard for moving predictive models between applications and, as a consequence, is supported by the leading technology companies including IBM, Microsoft, Oracle, SAP, SAS and Software AG to name a few. PMML is developed by the Data Mining Group (DMG), an independent vendor led consortium that develops data mining standards.

PMML is a very mature standard. Its latest version, PMML 4.4, was released in August 2016. Given that different data mining tools may support different versions of PMML, Zementis Server incorporates proprietary IP developed by Software AG that converts any older version of PMML (versions 2.0, 2.1, 3.0, 3.1, 3.2, 4.0, 4.1, 4.2 and 4.3) into version 4.4. This converter also checks the code for any syntactic and semantic problems and corrects known issues found in the PMML code of certain model building tools automatically.

PMML 4.4 incorporates many new elements into the standard, including elements for representing Time Series Analysis (including SpectralAnalysis, ARIMA and SeasonalTrendDecomposition) and Anomaly Detection models.

Note

PMML 4.2 changed the way the target field is referred to in the mining schema element. In PMML 4.2, the target field is simply referred to as "target" while in previous versions of PMML, it was referred to as "predicted". This change avoids any confusion related to the target field which is used to train a model and the true predicted field which is output by a model after scoring. As a consequence, Zementis Server also changed the way it treats predicted fields. If a PMML file is missing the output element, Zementis Server will add it to the file and will name the predicted output field "predictedValue" if no target field name is specified in the model's mining schema. If however, the target field is given, Zementis Server will name the predicted output field "predictedValue_X" where X is the name of the target field as specified in the mining schema. Zementis Server will not add any output fields to a PMML file if it already has an output element.

If you would like to learn more about PMML, we highly recommend that you visit the Software AG web site for a list of resources. We also recommend the book PMML in Action (2nd Edition): Unleashing the Power of Open Standards for Data Mining and Predictive Analytics by Alex Guazzelli, Wen-Ching Lin, and Tridivesh Jena, which is available for purchase on Amazon.com. "PMML in Action" gives an introduction to PMML as well as a PMML-based description of all the predictive modeling techniques supported by Zementis Server.

Software AG also offers a two-day on-site training course in PMML which is usually enough training for data scientists to become highly productive in using PMML. No pre-requisites for this course are required to be effective.

Zementis Server supports an extensive collection of statistical and data mining algorithms. These are:

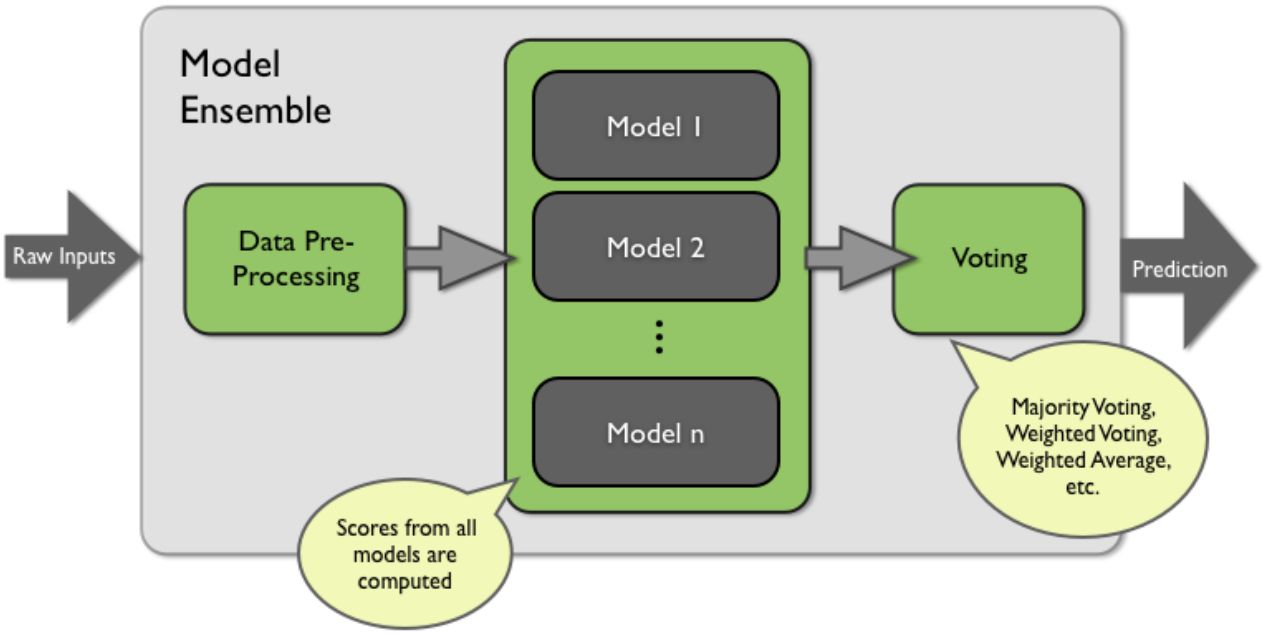
- Anomaly Detection Models (Isolation Forest and One-Class SVM)
- Association Rules Models (Rectangular or Transactional format)
- Clustering Models (Distribution-Based, Center-Based, and 2-Step Clustering)
- Decision Trees (for classification and regression) together with multiple missing value handling strategies (Default Child, Last Prediction, Null Prediction, Weighted Confidence, Aggregate Nodes)
- Deep Neural Networks (MobileNet, VGGNet, ResNet, RetinaNet)
- K-Nearest Neighbors (for regression, classification and clustering)
- Naive Bayes Classifiers (with continuous or categorical inputs)
- Neural Networks (Back-Propagation, Radial-Basis Function, and Neural-Gas)
- Regression Models (Linear, Polynomial, and Logistic) and General Regression Models (General Linear, Ordinal Multinomial, Generalized Linear, Cox)

- Ruleset Models (Each rule contains a predicate and a predicted class value)
- Support Vector Machines (for regression and multi-class and binary classification)
- Scorecards (point allocation for categorical, continuous, and complex attributes as well as support for reason codes)
- Time Series Models (Univariate Seasonal and Non-Seasonal ARIMA with computation of confidence intervals for non-seasonal models, Multi-variate State Space Models with computation of confidence intervals, Support for multi-step output in JSON format)
- Multiple models (model ensemble, segmentation, chaining, composition and cascade), including Random Forest Models and Stochastic Boosting Models

Zementis Server also implements the definition of a data dictionary, missing and invalid values handling, outlier treatment, as well as a myriad of functions for data pre- and post-processing, including: text mining, value mapping, discretization, normalization, scaling, logical and arithmetic operators, conditional logic, built-in functions, business decisions and thresholds.

Due to the highly publicized Netflix prize and the many tools that now make it easier for data scientists to develop a solution containing multiple models, model ensembles are now being used to build many predictive solutions. As depicted in Figure 2.1, in a model ensemble, every model is executed and the overall result or output is a combination of the partial results obtained from each model.

Figure 2.1. Model Ensemble

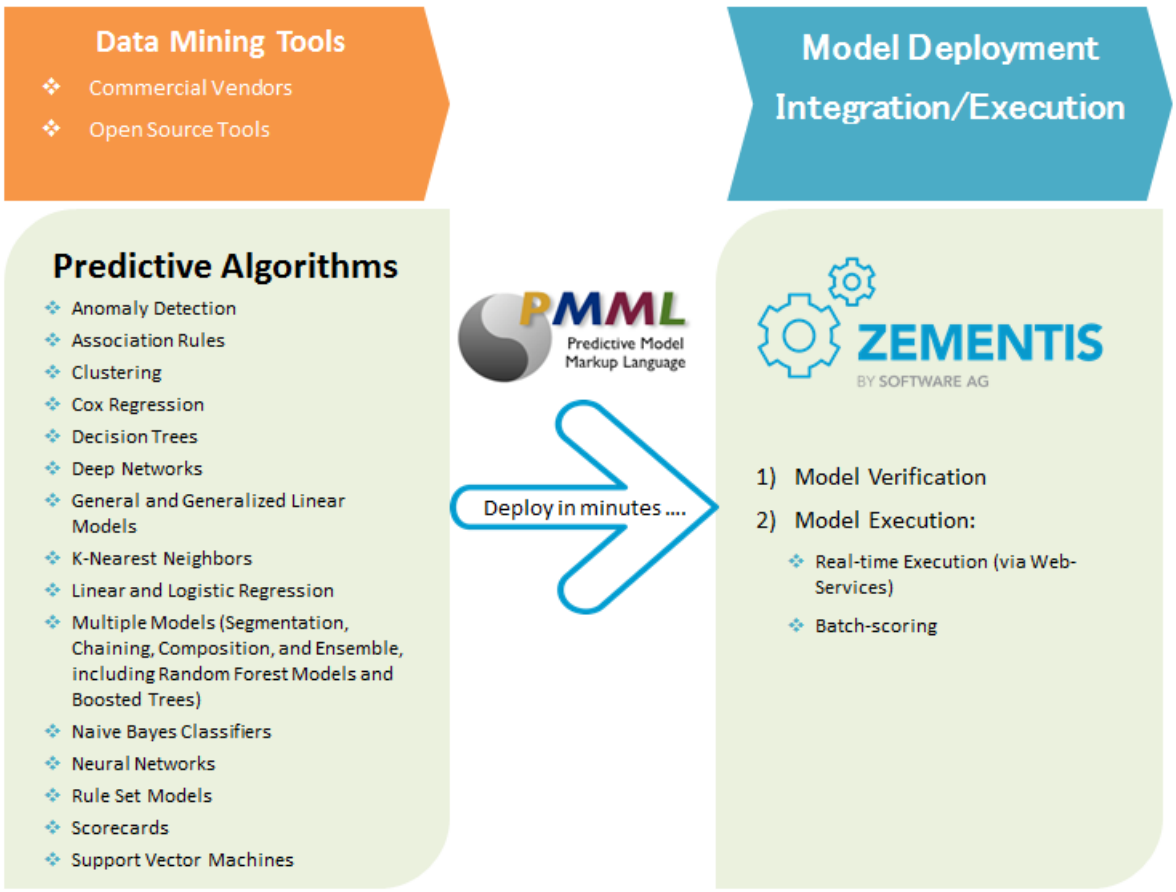


PMML is capable of representing not only a model ensemble but also model composition, segmentation, chaining and cascade. The same is true for Zementis Server, which consumes and executes PMML files containing multiple models. With Zementis Server and PMML, after a model is built, export it as (or convert it to) PMML, upload it to Zementis Server, and start scoring right away.

Zementis Server makes the task of verifying a model extremely easy. After a model gets uploaded in Zementis Server, a test data file containing the expected results can be uploaded so that the necessary validation can be performed, before the model is actually used to score new data. When presented with a scored data file, Zementis Server will automatically operate in score-matching test mode. In this mode, Zementis Server will compare expected scores against computed scores for each data record and warn the user if any mismatches are found.

The overall process of model building, using a commercial or open-source data mining tool as well as model deployment, verification, and execution is depicted in Figure 2.2. In the next sections, we elaborate on each phase of this process in more detail.

Figure 2.2. Predictive Modeling Process



2.1. Predictive Model Building Process

The process of creating predictive models starts by defining a clear business goal that needs to be achieved. This is followed by the data analysis phase in which the data scientist mines historical data looking for all the pieces deemed necessary for model building. Data is usually processed and feature detectors are created before a predictive algorithm such as a neural network is trained. Data analysis, model building and model validation is usually performed within the scientist's desktop through the use of an array of tools and scripts. Today, the leading statistical packages are able to export models in PMML, the language recognized by Zementis Server. Examples of such statistical packages are IBM SPSS, SAS, R, and KNIME. For a more comprehensive list of tools that support the PMML standard, check the Powered by PMML on the Data Mining Group (DMG) web site.

Besides this guide and as part of the overall documentation for Zementis Server, a number of sample models represented in PMML format are also available for inspection and use. Our sample models provide the PMML files listed in Table 2.1. These models were obtained from a variety of datasets, including the Iris, Heart, Audit and Diabetes datasets. We use three of the sample models built with the Iris dataset to showcase the power of web services through a series of examples. These are featured in the code shown in Chapter 5.

The Iris classification problem is one of the most famous data mining problems and datasets. It involves determining the class of an Iris plant given the length and width of its sepal and petal. Possible classes are: `setosa`, `virginica`, and `versicolor`. The models built with the Iris dataset not only predict the class with the highest probability, but also output the probabilities for each of the three classes. For more on the Iris dataset and for further information on the Heart Disease dataset, please refer to Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

The Audit dataset is supplied as part of the R Rattle package - http://rattle.togaware.com (it is also available for download as a CSV file from http://rattle.togaware.com/audit.csv). The Audit data set is an artificial dataset consisting of fictional clients who have been audited, perhaps for tax refund compliance. For each case an outcome is recorded (whether the taxpayer's claims had to be adjusted or not) and any amount of adjustment that resulted is also recorded.

The Diabetes dataset consists of ten physiological variables (age, sex, weight, blood pressure ...) measure on 442 patients, and an indication of disease progression after one year. The goal is to predict disease progression from the given physiological variables. For more information on the Diabetes dataset, please refer to Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Table 2.1. Sample Predictive Models

File Name	Description
`HeartDisease_MS_Classification.pmml`	A multiple model implementing model segmentation and used to predict the likelihood of a person developing a heart disease. It contains three different models: a linear regression model, a decision tree and a neural network model. Each predictive model is executed depending on the value of an input field.
`TaxAudit_SVM.pmml`	A predictive model composed of a support vector machine used to predict compliance issues with tax returns and the consequent need for them to be audited.
`CustomerChurn_NN.pmml`	A predictive model composed of a neural network model used to predict the likelihood of churn, based on attrition, for a company's customer base. This model also defines thresholds and business rules as part of the model's post-processing for implementing a business strategy to mitigate the risk of churn.
`Diabetes_RF.pmml`	A predictive model composed of a random forest model used to predict diabetes progression for a group of patients. This predictive model is an example of multiple models being used to implement a random forest model (model ensemble).
`ECommerceFraud_NN.pmml`	A predictive model composed of a neural network model used to predict the likelihood of fraud for e-commerce transactions. This model requires the use of custom functions for some of its data pre-processing, which are made available through the file "custom.jar". It also requires a lookup table, which can be found in the "customerStateMappingTable.xls" file. Both files are available as custom resource files (see Table 2.2 for information on how to locate these files).
`Transformations.pmml`	This file contains a series of data pre-processing steps. It illustrates how PMML, in conjunction with Zementis Server, can be used solely for data manipulation. The results obtained from a PMML file containing transformations can then be used for training a predictive model.
`Iris_NN.pmml`	A neural network model used to predict the class of Iris flower. This model is used to illustrate the use of web services.
`Iris_MLR.pmml`	A multinomial logistic regression used to predict the class of Iris flower. This model is used to illustrate the use of web services.

File Name	Description
`Iris_CT.pmml`	A CART decision tree used to predict the class of Iris flower. This model is used to illustrate the use of web services.

All sample files described here are available to download from the Zementis Console Help page. In there you will find a link to a compressed file in ZIP format. When uncompressed, this file reveals a number of directories which contain the sample files. Table 2.2 describes how the sample files are organized.

Table 2.2. Directory Structure of Sample Models

Directory	Contents
`models`	Predictive models (PMML) files: contains the PMML files for all the sample solutions.
`resources`	Custom resource files: contains custom functions (JAR file) and a lookup table for model "ECommerceFraud_NN.pmml". Upload these resource files in Zementis Server before uploading the PMML model file.
`data`	Scored data files: contains the scored data files in CSV format for model execution for all the sample predictive models. Score a data file in Zementis Server against its respective model in order to perform the score matching test. Each data file is named according to its respective PMML file. In this case, if the PMML file is "Diabetes_RF.pmml", the data file is "Diabetes_RF.csv".
`rest-client`	Source and build files for sample Java client to Zementis REST API.

2.2. Deploy and Test Predictive Models

Once your models are built and expressed in PMML, it is extremely easy to deploy them in Zementis Server. Managing and deploying models can be accomplished through the use of the Zementis Console.

2.2.1. Deploying Models

Models are deployed in Zementis Server by uploading them directly in the Zementis Console. Although a data mining tool may export an older version of PMML, Zementis Server will automatically perform comprehensive syntactic and semantic checks, correct known issues and convert your PMML file to version 4.4 when the `Enable validation and correction on PMML file(s)` checkbox is checked. By default, the `Enable validation and correction on PMML file(s)` checkbox is checked. Unchecking the checkbox will improve upload time, but

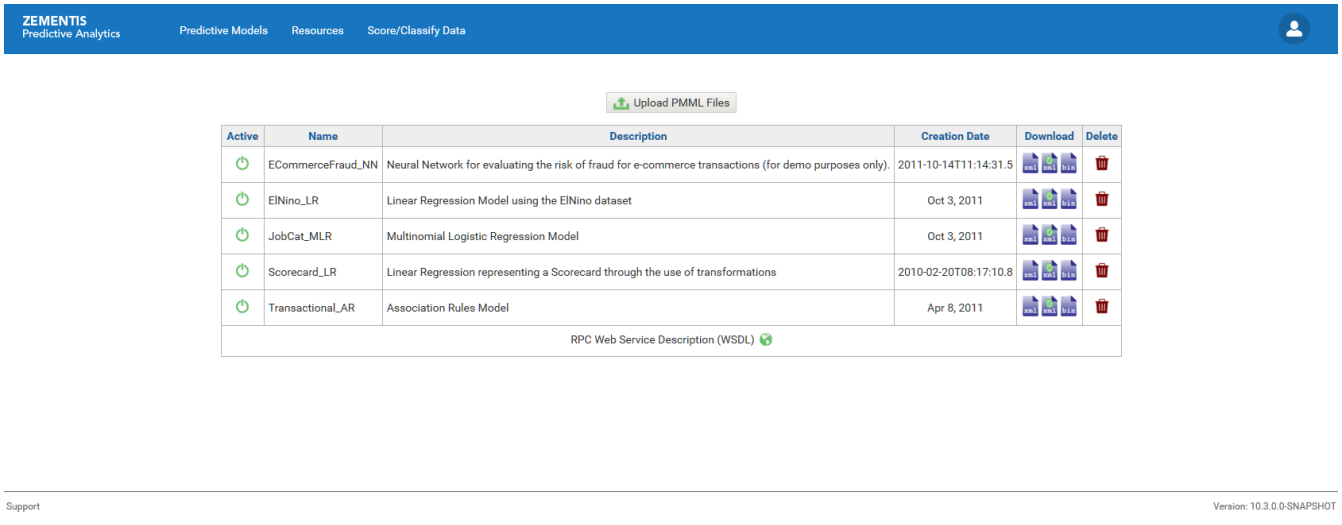
this is only recommended for annotated PMML files that are generated after being processed by Zementis Server. The annotated PMML file for a model can be downloaded by clicking the middle icon in the "Download" column of the corresponding model name. The yellow shield indicates potential issues with a PMML file that may need to be reviewed. The detailed warning messages are available in the annotated PMML file as comments at the top of the PMML file. The corresponding model is fully functional and more often than not, these warnings are not relevant to the scoring process. However, a review of these messages is highly recommended as, in some cases, they may have an impact on the scoring process. The green shield indicates that the PMML file was uploaded without any warnings or errors. For security vulnerability mitigation, the model name of uploaded model may be replaced with safe characters when the value of `modelName` attribute in PMML file(s) contains special or reserved characters.

Tip

If the PMML file is large, such as the Random Forest model, we recommend compressing the file using ZIP/ GZIP before uploading. This will reduce the upload time dramatically.

If you had previously uploaded models into Zementis Server, those models would be listed in the Zementis Console Predictive Models page. Figure 2.3 shows the Zementis Console after uploading the sample predictive models described in Table 2.1.

Figure 2.3. Predictive Models in the Zementis Console



For more information on how to upload your models through the Zementis Console, see the Help page.

2.2.2. Testing Models

Given that models are built with different tools, you need to make sure that both Zementis Server and the model development environment produce exactly the same results during scoring.

Zementis Server provides an integrated testing process to make sure your model was represented accurately, uploaded correctly, and works as expected. This is also done through the Zementis Console which allows for a model verification data file to be uploaded for score matching. This file should be in Comma Separated Values (CSV) format containing one record per line (for more information on how to format your CSV file for scoring, please refer to the Zementis support forum). Each record should have values for all the input variables along with at least one of the output variables. The values for the output variables serve as the expected predicted values. Zementis Server will compute new predicted values and compare them to the expected ones. If all the values match, the model is considered production-ready, i.e. ready for scoring. If not, Zementis Server offers execution trace details to facilitate trouble shooting.

The sample predictive models (Table 2.1) provide CSV files that can be used for testing their respective PMML files. For more information on how to test models, see the Zementis Console Help page or the Zementis support forum on model verification.

PMML also offers a "ModelVerification" element for similar testing purposes. In this way, verification records are part of the PMML file itself. Given that Zementis Server supports this element, there is more than one way to test models. For more information on this specific PMML element, please refer to the Data Mining Group (DMG) web site or to the book PMML in Action (2nd Edition): Unleashing the Power of Open Standards for Data Mining and Predictive Analytics by Alex Guazzelli, Wen-Ching Lin, and Tridivesh Jena, which is available for purchase on Amazon.com.

2.3. Data Scoring and Classification

Bulk scoring in batch mode can be easily performed through the Zementis Console, using the same process as for model testing. First, select the target model and then upload a data file in CSV format. The only difference between this process and the score-matching test is that in the present case, the predicted field and its expected scores are not part of the data file. Zementis Server will process the uploaded file and return a new file with your original data expanded with an extra column containing the predicted variable and the scores/results for each row. For more details on how to format your data file for batch scoring in Zementis Server, please refer to the Zementis support forum on data formatting.

Tip

If the data file is large, Software AG suggests compressing the file in ZIP format before uploading. This reduces the upload time dramatically. In this case, Zementis Server also returns a compressed file containing the results.

Real-time scoring allows other applications to get and use predictions on demand from anywhere in your enterprise. With Zementis Server this can be achieved through standard web service calls. Details on using web services can be found in Chapter 5.

2.4. Model Metrics

The Model Metrics feature describes Memory Metrics and Prediction Metrics for the selected model.

Clicking on the model name from the Zementis Console displays a pop-up window with model metrics for the respective model. Memory metrics information of a model is displayed only if the Zementis Java Agent is configured and the Prediction Metrics information will be displayed only when the scoring is applied at least once. Please refer to the Deployment Guide for configuring the Zementis Java Agent.

Memory Metrics provides the information about the memory footprint of the model on the server and its related attributes like used memory, free memory and total memory of the application. The same information is represented as a vertical Bar Chart.

Prediction Metrics provides a scoring result summary for the models. Prediction Metrics of a classification model displays the predicted categories and its respective counts as a Pie Chart. Prediction Metrics of a regression model displays the Five Point Summary of predicted values i.e., Minimum, FirstQuartile, Median, ThirdQuartile and Maximum values as a Box Plot.

With the fresh deployment of Zementis Server the Prediction Metrics is empty and it will be displayed only if scoring is applied. Prediction Metrics of a model will be reset when the model is deleted, deactivated or if the server is restarted. Also the Prediction Metrics information that shows up is always the cumulative result with the past scoring of the model.

Note

Currently the Prediction Metrics feature is supported only for classification and regression models. Please refer to the Deployment Guide for configuring Model Metrics.

Figure 2.4. Memory Metrics and Prediction Metrics of Classification Model

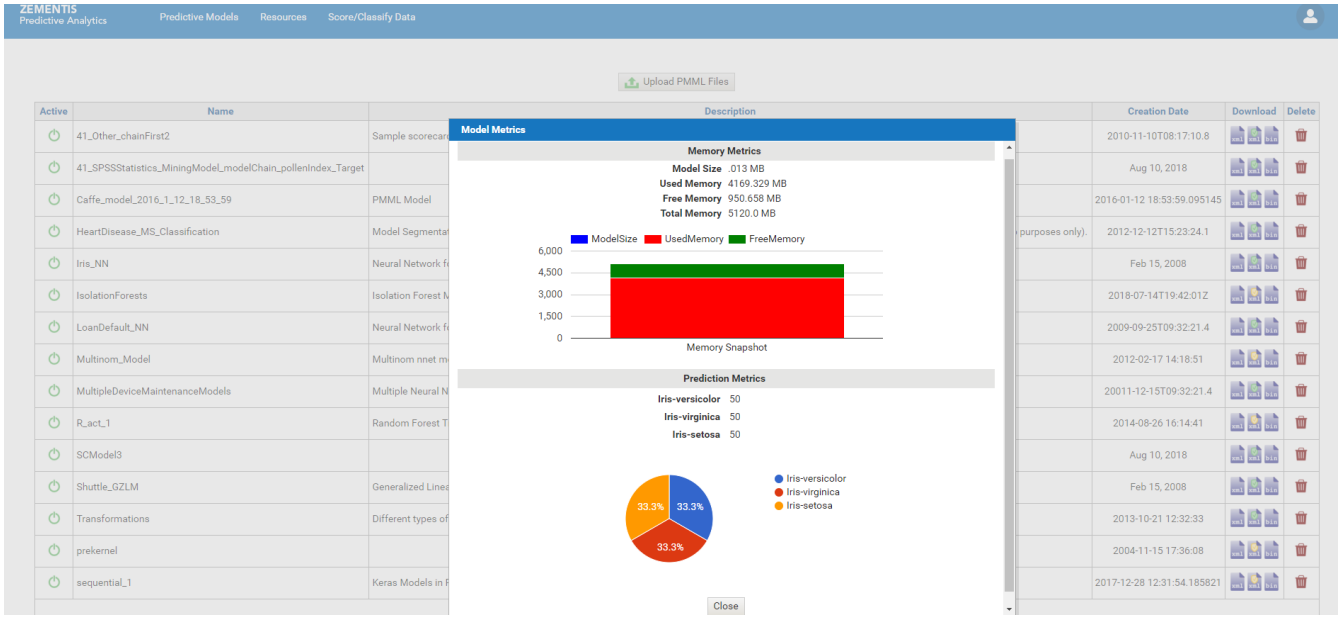
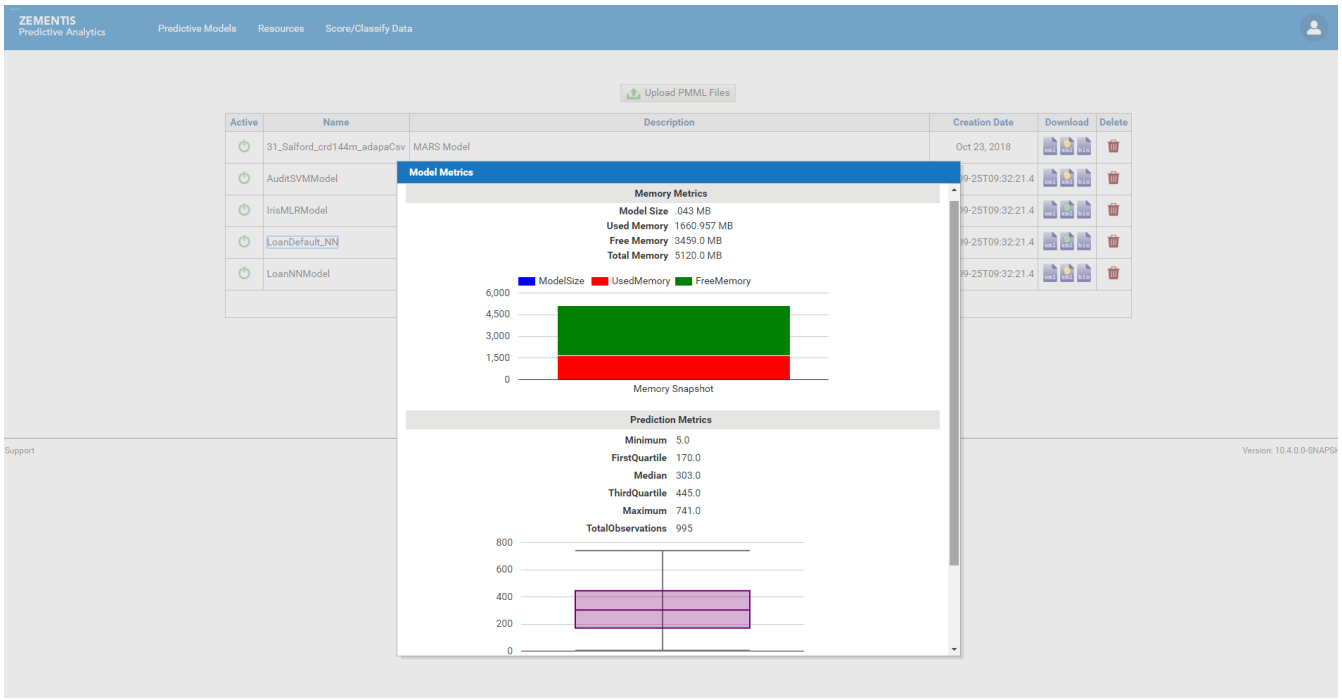


Figure 2.5. Memory Metrics and Prediction Metrics of Regression Model



2.5. Other Data Sources

Zementis Server also supports applying predictive models to a wide variety of data sources, such as images, audio files, videos, binary feeds or even text files as input data. In Zementis Server, with binary input definition and proper

custom functions which convert unstructured data into structured data, the data type of the input source in the deployed model could be in any format for analytics process. Details on how to apply models to binary data source can be found in Section 3.4.

Chapter 3. Custom Resources

Predictive models may require external resources such as custom functions, look-up tables or training data tables. Files containing such resources can be uploaded in Zementis Server using the Zementis Console. Note, such resources should be uploaded before any models that depend on them. Also, deleting a resource file will remove all the resources contained in it from Zementis Server. In this case, first the model that is dependent on the resource should be deleted.

3.1. Custom PMML Functions

Zementis Server provides a facility to create and use custom PMML functions. This capability enables, for example, the implementation of intricate calculations that cannot be easily described in PMML, functions that access external systems to retrieve necessary data, or even specialized algorithms not supported by PMML. One class of functions that can be easily implemented using custom functions which are aggregations over a period of time or window of transactions. Aggregations are used to obtain, for example, the count, average, maximum and minimum for a set of records. One example is to use custom functions to obtain the average transaction amount for a certain account for the last 30 days. The predictive model ECommerceFraud_NN.pmml, provided as part of the sample models, uses several custom functions to compute the average transaction amount as well as the transaction velocity for a period of time. This model is described in Table 2.1

Zementis Server currently supports custom functions written in Java. Once created and made available to Zementis Server, custom functions are used the same way as the built-in ones. The steps to achieve this are explained in the following sections.

3.1.1. Create Custom PMML Functions

Custom functions are implemented as static methods of Java classes. For a method to be recognized as a custom PMML function, the containing class needs to be annotated with the Zementis Server specific `@PMMLFunctions` annotation. In addition, the types of the method parameters as well as its return type must be compatible with the PMML data types. An example of such a function is shown in Figure 3.1.

Figure 3.1. Custom PMML Function Example

```
package com.company.udf;

import com.zementis.stereotype.PMMLFunctions;

@PMMLFunctions(namespace = "company")
class CustomFunctions {

    public static Long factorial(Long n) {
        if (n == null) {
            return null;
        } else if (n < 0) {
            throw new IllegalArgumentException();
        } else if (n == 0) {
            return 1;
        } else {
            return n * factorial(n-1);
        }
    }

}
```

In this example, the class `RecursiveFunctions` has been annotated with the `@PMMLFunctions` annotation. This annotation informs Zementis Server that the class contains methods which may be used as PMML functions. The parameter `namespace` defines a namespace for the functions defined in this class. Namespaces prevent conflicts between function names. Within PMML, the namespace is used as a prefix for the name of the custom function. For example, the PMML name of the function implemented by the Java method `factorial` in Figure 3.1 would be `company:factorial`.

The namespace does not have to be unique for each class. Multiple classes may specify the same namespace. This would allow, for example, creating the notion of a function library where functions spread across multiple class files are grouped under one namespace. In this scenario, extra care needs to be taken so that there are no ambiguities between function names located in different classes.

Within each annotated class, only methods that are declared as `public` and `static` can be used as PMML functions. In addition, a method should accept parameters and return values compatible with the PMML data types. Table 3.1 provides the Java primitive types and classes that correspond to the different PMML data types.

Table 3.1. PMML and Java types in Zementis Server

PMML Data Type	Java Primitive Type	Java Class
`boolean`	`boolean`	`java.lang.Boolean`
`date`		`org.joda.time.LocalDate`
`dateTime`		`org.joda.time.DateTime`
`double`	`double`	`java.lang.Double`

PMML Data Type	Java Primitive Type	Java Class
`float`	`float`	`java.lang.Float`
`integer`	`long`	`java.lang.Long`
`string`		`java.lang.String`
`time`		`org.joda.time.LocalTime`
`binary`		`java.io.InputStream`
`binary (buffered)`	`byte[]`	`byte[]`

The method return type must be one of the Java types listed in the table. Note that methods declared as `void` cannot be used as PMML functions. The types of the parameters must be either among those listed in the table or among one of their super-classes or super-interfaces (`java.lang.Object`, `java.lang.Comparable`, or `java.lang.Number`). Finally, methods can also declare variable number of parameters (`varargs`).

Important

Make sure these methods are thread-safe as Zementis Server may need to execute these methods concurrently in different threads.

Caution

The custom functions are packaged into a JAR file and loaded dynamically at runtime. This could result in serious security risks which can lead to system compromise. Even though uploading the custom function JARs require Administrative privileges (`adapa-admin` role), it is highly recommended that users upload only trusted JARs.

3.1.2. Use Custom PMML Functions

To make custom functions available to Zementis Server, compile the corresponding classes into a JAR file and upload it using the Zementis Console. To compile a class using the `@PMMLFunctions` annotation, include the `adapa-api-10.7.0.2.jar` file in the classpath. This file is included in the Zementis Server distribution package as well as the provided package of sample files.

Once deployed, custom functions can be used exactly like the built-in functions within `Apply` transformations. Please make sure you use the fully qualified name of the custom function, i.e. prefix the function name with the appropriate namespace. The PMML fragment in Figure 3.2 contains a simple example that uses the function defined in Figure 3.1.

Figure 3.2. Example Using a Custom Function in PMML

```
<DerivedField name="field2" optype="continuous" dataType="integer">
    <Apply function="company:factorial">
        <FieldRef field="field1"/>
    </Apply>
</DerivedField>
```

3.1.3. Non-Deterministic Functions

When processing PMML models, Zementis Server performs certain performance optimizations which assume that functions are deterministic, i.e. when presented with the same input values they always return the same result. However, this may not be the case for all functions. For example, the result of a function may depend on the current time and date. Another example might be a call to an external source that retrieves information that is being modified by other systems.

With Zementis Server, a custom function may be declared as non-deterministic by annotating the corresponding implementation Java method with the `@NonDeterministicFunction` annotation. Note that this annotation marks a method, and not the containing class. This means a class implementing multiple functions may contain a combination of deterministic and non-deterministic functions.

The following is an example of a non-deterministic function which provides the current time value for a specific a time zone.

Figure 3.3. Custom PMML Function Example

```
package com.company.udf;

import com.zementis.stereotype.PMMLFunctions;
import com.zementis.stereotype.NonDeterministicFunction;
import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;

@PMMLFunctions(namespace = "company")
class CustomFunctions {

    @NonDeterministicFunction
    public static DateTime dateTimeAtZome(String timeZone) {
        if (timeZone == null) {
            return null;
        }
        return new DateTime(DateTimeZone.forID(timeZone));
     }

}
```

3.2. External Lookup Tables

Predictive models can sometimes require the use of lookup tables. If relatively small and static, these tables can be easily embedded within the PMML file itself. However, if they are fairly large and/or they are modified frequently, it is more practical to create and manage them separately. Zementis Server supports external lookup tables and their seamless integration with predictive models.

As an example of a simple lookup table, suppose a model makes use of a country's GDP (Gross Domestic Product). That requires the ability to look up the GDP by country name. Such a simple lookup table is shown in Figure 3.4. Given an input country, say Taiwan, the row in the lookup table which has `Taiwan` in its first column maps it to a GDP of `576.20`. Being an example, we show only a few mappings; in reality, we can imagine similar cases with hundreds and even thousands of mappings.

Figure 3.4. Lookup Table Example

Find Country	Set GDP
Afghanistan	21.50
Brazil	1,492.00
Canada	1,023.00
China	7,262.00
Egypt	316.30
Germany	2,362.00
Greece	226.40
India	3,319.00
Iraq	54.40
Morocco	134.60
Switzerland	251.90
Taiwan	576.20
US	11,750.00

The predictive model ECommerceFraud_NN.pmml, provided as part of the sample models, uses a lookup table to retrieve the number of points for each US state. This model is described in Table 2.1

3.2.1. Create Lookup Tables in Excel

Zementis Server supports lookup tables implemented in Excel files. In this section, we describe the structure of such tables. In general, a lookup table has one or more input variables and an output variable. The intended functionality is that any set input values can be looked up to retrieve the corresponding output value, if one is found. Figure 3.5 shows a slightly expanded version of the previous example. Here, we have two input variables, `Country` and `State`. The output variable is `GDP`.

Figure 3.5. Sample Excel Lookup Table

LookupTable	*GDPTable*	
input	Country	String
input	State	String
output	GDP	Double
Country	**State**	**GDP**
Afghanistan		21.50
Brazil		1492.00
Canada		1023.00
China		7262.00
Egypt		316.30
Germany		2362.00
Greece		226.40
India		3319.00
US	California	557.37
US		11750.00

A single Excel file may contain one or more lookup tables. However, only one lookup table is allowed per worksheet. Multiple tables should be arranged in separate worksheets. Within a worksheet, the beginning of a lookup table is identified by the keyword `LookupTable`. The name of the table should appear in the cell right next to this keyword (`GDPTable` in this example). The definitions of the input and output variables start in the cell right below the `LookupTable` keyword. Variables must be listed one per row, with the output variable listed last. For each variable, provide the usage (`input` or `output`), the name and the data type. The variable names must be unique. The allowed types of data are `Integer`, `Long`, `Double`, `Float`, `Boolean` and `String`, corresponding to the Java primitive types. In this example, the first row defines an input variable called `Country` which is of type `String`. The next row defines an input variable `State`, again of type `String`. Finally, the output variable is called `GDP` which is of type `Double`.

The data area of the lookup table starts right below the output variable definition. In the simple form shown here, this area consists of one column per variable. The first is the header row, where the name of the corresponding variable is listed. All the following rows contain combinations of input and output values. Each row represents a mapping from the input values to the output value. Note that empty cells are allowed. For an input variable, an empty cell represents `any value`. For an output variable, an empty cell represents no value (or a `null` value). A fully empty row, i.e., a row with empty cells for all the variables marks the end of the table. Anything below a fully empty row is ignored.

Duplicate mappings are not allowed. However, with empty cells representing `any value`, overlapping mappings are possible (and allowed). To illustrate this, please consider the overlapping mapping in last two rows of the example in Figure 3.5. The second to last row implies that if the country is USA and the state is CA then the GDP is 557.37. However, the last row implies that if the country is USA, the GDP is 11750.00 no matter what the state is. In the presence of overlapping mappings, the tighter mapping, i.e. the mapping with more matching input values, prevails. In the current example, this means that the a GDP lookup for CA will result in 557.37 and a GDP lookup for any other state will be 11,750.00

In some cases, it is desirable to arrange some mapping as a cross tab. Such an example is shown in Figure 3.6 where the probability of child obesity can be looked up by child age and group. The probabilities for all the combinations of four child groups (Rural Girls, Urban Girls, Rural Boys, and Urban Boys) and six different ages (10 through 15) are presented.

Figure 3.6. A LookupTable with two inputs and one output

LookupTable	*ChildObesity*					
input	Group	String				
input	Age	Integer				
output	ObesityProbability	Double				
Age	10	11	12	13	14	15
Group	**ObesityProbability**					
Rural Girls	0.0058	0.0116	0.0566	0.0309	0.0174	0.0000
Urban Girls	0.0550	0.0570	0.0467	0.0650	0.0420	0.0526
Rural Boys	0.0222	0.0333	0.0294	0.0411	0.0118	0.0384
Urban Boys	0.0730	0.0730	0.0745	0.0627	0.0668	0.0117

The structure of a cross tab lookup table is similar to the previous one. The only difference is that the values for one or more of the input variables are listed horizontally above the header of the data area, as opposed to vertically. Note that not all input variables can be listed horizontally. At least one must be listed vertically. In addition, the header cell containing the name of the output variable must span all the data columns. Similarly to the previous case, the boundaries of the lookup table are identified by the first fully empty row and the first fully empty column.

Our sample solution provides a lookup table in the Excel file `borrowerStateMappingTable.xls`. This table is used by the demo PMML model for fixed rate loans.

3.2.2. Use Lookup Tables in PMML

In PMML, lookup tables can be used within `MapValues` transformations and the `TableLocator` mechanism. In the following sample PMML snippet, the lookup table `ChildObesity` is used to retrieve the appropriate child obesity probability.

```
<LocalTransformations>
 <DerivedField name="obesityProbability" dataType="double" optype="continuous">
     <MapValues outputColumn="Probability" defaultValue="0.5" mapMissingTo="0">
         <FieldColumnPair column="Age" field="childAge" />
         <FieldColumnPair column="Group" field="childGroup" />
            <TableLocator>
             <Extension extender="ADAPA" name="TABLE_NAME" value="ChildObesity" />
            </TableLocator>
     </MapValues>
 </DerivedField>
</LocalTransformations>
```

The table used in the mapping is identified in the `Extension` element. The value attribute of this element contains the name of the lookup table to use. The rest of the structure details what fields of the model (`childAge`, `child-Group`, and `childObesity`) correspond to what columns (`Age`, `Group`, and `Probability`) of the lookup table.

3.3. External Training Data Tables

Some algorithms (e.g. K Nearest Neighbor) expect a table of training data as part of the model. This table can be included in the PMML document, or loaded as an external resource in CSV format. The format of the external table is identical to the one of the test data offered in the samples directory. This file should be in CSV format containing one record per line (for more information on how to format your CSV file, please refer to the Zementis support forum). Each record should have values for all the input variables along with the predicted values.

```
  <TrainingInstances>
   <InstanceFields>
    <InstanceField field="Sepal.Length" column="Sepal.Length"/>
    <InstanceField field="Sepal.Width" column="Sepal.Width"/>
    <InstanceField field="Petal.Length" column="Petal.Length"/>
    <InstanceField field="Petal.Width" column="Petal.Width"/>
    <InstanceField field="Species" column="Species"/>
   </InstanceFields>
            <TableLocator>
                <Extension extender="ADAPA" name="TRAINING_INSTANCES_NAME" value="Iris_KNN.csv" />
            </TableLocator>
  </TrainingInstances>
```

The table is identified in the `Extension` element. The value attribute of this element contains the name of the training data table to use including the file ending. The `InstanceFields` element details one to one correspondence between the field of the model and the column of the table.

3.4. Using Binary Data Type

Some predictive models use `binary` data as input for generating predictions. Zementis Server supports applying models to `binary` data by utilizing an external custom function which transforms unstructured data into the format expected by the model. Given proper `binary` input definition and appropriate custom function deployed in Zementis Server, the `binary` input data can be seamlessly integrated into the prediction process. This section shows how to

define a `binary` input in PMML using `DeepNetwork` model as an example and how to create the corresponding custom function that processes the provided `binary` input data. This sample is also packaged with the distribution: `adapa-app-10.7.0.2.zip/adapa-sampels/resources/custom-functions-deepnetwork`.

3.4.1. Using Default `binary` Type

Figure 3.7 shows an example of how to define `binary` input type. This can be done by setting the data type as `binary` in the `<DataField>` element. The `binary` input data, `input_image` in this case, can be sent to Zementis Server by providng the contents of the corresponding file. Zementis Server will process the contents of the file as a single `binary` input record. It is recommended to provide MIME type in `mimeType` attribute, for example `mimeType="image/png"`. Zementis Server will do the data format verification before starting the prediction process in order to avoid data type mismatch.

Figure 3.7. Binary DataType Example

```
<DataDictionary numberOfFields="2">
  <DataField name="input_image" optype="categorical" dataType="binary" mimeType="image/png"/>
  <DataField name="predictions" optype="categorical" dataType="string">
    <Value value="predicted_category_a"/>
    <Value value="predicted_category_b"/>
    <Value value="predicted_category_c"/>
  </DataField>
</DataDictionary>
```

Here are the steps to create a corresponding custom function:

- Implement a custom function as static method of Java class.
- Annotate it with Zementis Server specific `@PMMLFunctions` annotation.
- Specify the type of the method parameter as `java.io.InputStream`.

The custom function shown in Figure 3.8 processes the incoming `input_image` and returns the Base64 encoded String representing the pixel values of the image.

Figure 3.8. Custom Function of Binary Data Example

```
package com.zementis.udf;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import javax.imageio.ImageIO;
import org.apache.commons.codec.binary.Base64;
import com.zementis.stereotype.PMMLFunctions;

@PMMLFunctions(namespace = "myCustomFunction")
public class CustomFunctions {
    public static String getBase64String(InputStream inputStream) throws IOException {
        // read image from inputstream
        BufferedImage bufferedImage = ImageIO.read(inputStream);
        float[] array = preprocess(bufferedImage);
        return encodeFloatArrayToBase64String(array);
    }

    private static String encodeFloatArrayToBase64String(float[] array) {
        int float32Length = 4;
        byte[] floatByteArray = new byte[array.length * float32Length];
        for (int i = 0; i < array.length; i++) {
            byte[] thisFloatArray =

 ByteBuffer.allocate(float32Length).order(ByteOrder.LITTLE_ENDIAN).putFloat(array[i]).array();
            for (int j = 0; j < thisFloatArray.length; j++) {
                floatByteArray[float32Length * i + j] = thisFloatArray[j];
            }
        }
        return "data:float32;base64," + new String(Base64.encodeBase64(floatByteArray));
    }

    private static float[] preprocess(BufferedImage bufferedImage) {
        int height = bufferedImage.getHeight();
        int width = bufferedImage.getWidth();
        int channel = 3;
        float[] floatRGBArray = new float[height * width * channel];
        int counter = 0;
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                // read
                int pixel = bufferedImage.getRGB(j, i);
                int red = (pixel >> 16) & 0xFF;
                int green = (pixel >> 8) & 0xFF;
                int blue = pixel & 0xFF;

                // pre-process and write
                float r = (red / 127.5f) - 1.0f;
                floatRGBArray[counter++] = r;

                float g = (green / 127.5f) - 1.0f;
                floatRGBArray[counter++] = g;

                float b = (blue / 127.5f) - 1.0f;
                floatRGBArray[counter++] = b;
            }
        }
        return floatRGBArray;
    }
}
```

Once the custom function in Figure 3.8 is compiled and deployed, `myCustomFunction:getBase64String` can be used exactly like a built-in function within `Apply` transformations. The PMML fragment in Figure 3.9 contains a simple example that uses the function defined in Figure 3.8.

Figure 3.9. Example Using Custom Function of Binary Data in PMML

```
<DeepNetwork modelName="DeepNetwork Sample" functionName="classification" numberOfLayers="2">
  ...
  <LocalTransformations>
    <DerivedField name="input_base64String" optype="categorical" dataType="string">
      <Apply function="myCustomFunction:getBase64String">
        <FieldRef field="input_image"/>
      </Apply>
    </DerivedField>
  </LocalTransformations>
  <NetworkLayer layerType="Input" layerId="layer_1" connectionLayerId="na"
 inputFieldName="input_base64String">
    <LayerParameters inputDimension="(2, 2, 3)" outputDimension="(2, 2, 3)"/>
  </NetworkLayer>
  ...
</DeepNetwork>
```

3.4.2. Using Buffered `binary` Type

Zementis Server provides two ways to manage `binary` input data. The provided `binary` data can be processed either as a `java.io.InputStream` object or a `byte[]`. The data types are listed in Table 3.1. By default Zementis Server processes `binary` input data as `java.io.InputStream`. This means the provided input will be read only once. If `binary` input data needs to be read more than once, set `BINARY_BUFFERED` as `true` in `<Extension>` element as shown in PMML fragment in Figure 3.10.

Figure 3.10. Binary (Buffered) DataType Example

```
<DataDictionary numberOfFields="2">
  <DataField name="input_image" optype="categorical" dataType="binary" mimeType="image/png">
    <Extension extender="ADAPA" name="BINARY_BUFFERED" value="true" />
  </DataField>
  <DataField name="predictions" optype="categorical" dataType="string">
    <Value value="predicted_category_a"/>
    <Value value="predicted_category_b"/>
    <Value value="predicted_category_c"/>
  </DataField>
</DataDictionary>
```

Here are the steps to create a corresponding custom function:

- Implement a custom function as static method of Java class.
- Annotate it with Zementis Server specific `@PMMLFunctions` annotation.
- Specify the type of the method parameter as `byte[]`.

Figure 3.11. Custom Function of Buffered Binary Data Example

```
package com.zementis.udf;

import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.imageio.ImageIO;
import com.zementis.stereotype.PMMLFunctions;

@PMMLFunctions(namespace = "myCustomFunction")
public class CustomFunctions {

    public static String getBase64String(byte[] byteArray) {
        BufferedImage bufferedImage = ImageIO.read(new ByteArrayInputStream(byteArray));
        float[] array = preprocess(bufferedImage);
        return encodeFloatArrayToBase64String(array);
    }

    private static float[] preprocess(BufferedImage bufferedImage) {
        ...
        return floatRGBArray;
    }

    private static String encodeFloatArrayToBase64String(float[] array) {
        ...
        return base64encodedString;
    }
}
```

Once the custom function in Figure 3.11 is compiled and deployed, `myCustomFunction:getBase64String` can be used exactly like a built-in function within `Apply` transformations. The PMML fragment in Figure 3.12 contains a simple example that uses the function defined in Figure 3.11.

Figure 3.12. Example Using Custom Function of Buffered Binary Data in PMML

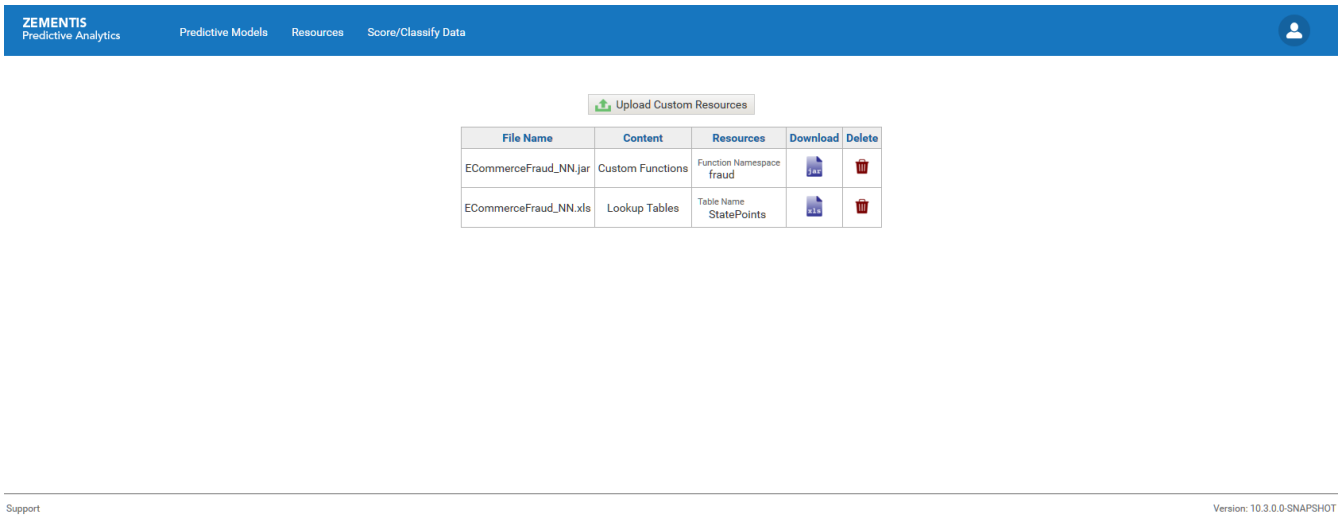
```
<DeepNetwork modelName="DeepNetwork Sample" functionName="classification" numberOfLayers="2">
  ...
  <LocalTransformations>
    <DerivedField name="input_base64String" optype="categorical" dataType="string">
      <Apply function="myCustomFunction:getBase64String">
        <FieldRef field="input_image"/>
      </Apply>
    </DerivedField>
  </LocalTransformations>
  <NetworkLayer layerType="Input" layerId="layer_1" connectionLayerId="na"
 inputFieldName="input_base64String">
    <LayerParameters inputDimension="(2, 2, 3)" outputDimension="(2, 2, 3)"/>
  </NetworkLayer>
  ...
</DeepNetwork>
```

3.5. Deploy Resources

Custom PMML functions or lookup tables are deployed in Zementis Server by simply uploading them directly in the Zementis Console.

If you have previously uploaded any resource files into Zementis Server, these are shown in the Zementis Console as a list. Figure 3.13 shows the Zementis Console after the uploading of the lookup table and custom functions (JAR file) used by predictive model ECommerceFraud_NN.pmml (for more details on this sample mode, see Table 2.1).

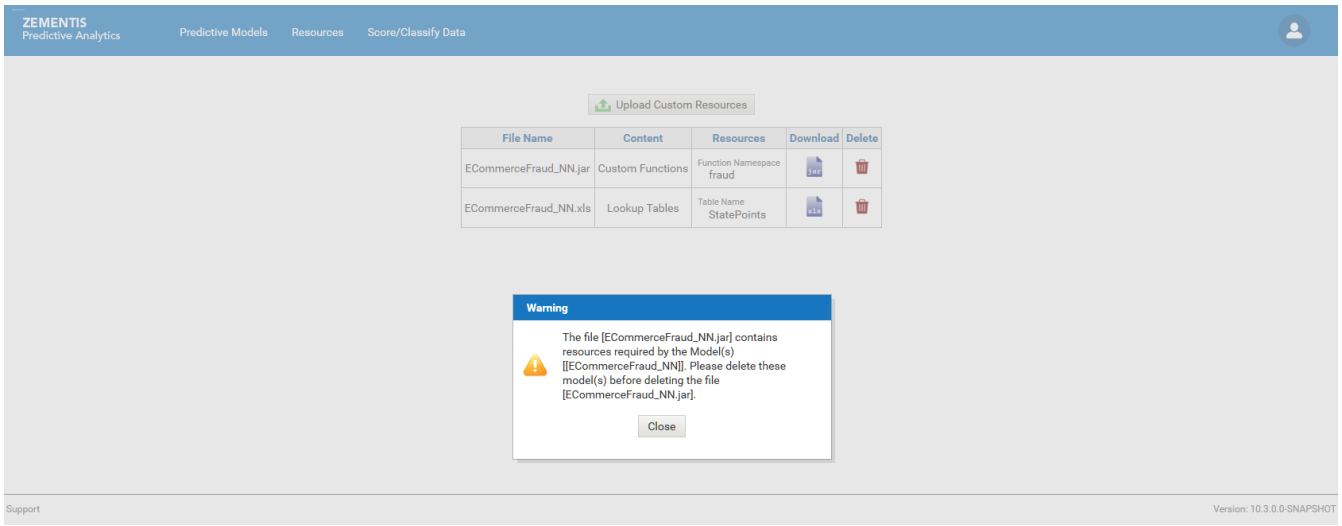
Figure 3.13. Resource Files in the Zementis Console



3.5.1. Deleting Resources

When deleting a resource file which is a downstream dependency of one of the models from the models list, you must first delete the model and then delete the resource. Figure 3.14 shows the Zementis Console when an exception is thrown.

Figure 3.14. Resource dependency exception in the Zementis Console



3.6. Supporting Python scripts in PMML

Zementis Server currently supports custom functions written in Java. Once created and made available to Zementis Server, custom functions are used the same way as the built-in ones. As an alternative, the same functionality can also be achieved by embedding python scripts in PMML. This approach eliminates the dependency on external resources by capturing the pre-processing transformation steps as python code embedded within the PMML file.

3.6.1. Define a Function with python script in PMML

Custom functions can be defined as a python script in PMML. For a Python script to be identified as a custom function script by zementis, it has to be defined as an `Extension` within the `Apply` element of `DefineFunction` element. An example of such a script is shown in Figure 3.16

The `name` attribute of the `Apply` element needs to be defined as `python` and the script has to be enclosed within the `Extension` element of `Apply` element by defining the `Extension` attributes appropriately.

Python script can be represented in clear text with proper indentation as shown in Figure 3.16 . It can also be represented in Base64 encoded text as shown in Figure 3.17 to avoid whitespace issues.

The python script has to be defined in the form of a python function by accepting required input as function parameters.

The function parameters are passed as string data types from zementis and they have to be converted into their respective python data types using python built-in functions. An example of type conversion is shown in Figure 3.15. Please refer Table 3.2 for the data type mapping.

Figure 3.15. Data Type Conversion Example

```
<DefineFunction name="customFunc" optype="continuous" dataType="double">
    <ParameterField name="transactionAmount" dataType="double" />
    <Apply function="python">
        <Extension extender="ADAPA" name="avgPerDay" value="double">
        def avgPerDay(transactionAmount):
            amount=float(transactionAmount)/30;
            return amount;
        </Extension>
        <FieldRef field="transactionAmount" />
    </Apply>
</DefineFunction>
```

Table 3.2. Data Type Mapping

PMML Type	Python Type
`integer`	`integer Example:int(paramValue)`

PMML Type	Python Type
`float, double`	`float Example:float(paramValue)`
`string, binary`	No conversion is required.

Note

The python script can be auto-generated into the PMML using Nyoka PMML exporter API. Please refer to Nyoka for details.

Table 3.3. Purpose of each attribute in Extension element

Attribute Name	Usage
`extender`	`Value of this attribute should be ADAPA.`
`name`	`The value of this attribute should be as same as the script function name. Refer Figure 3.16, getBase64EnocdedString is the main function and the same name is provided for the name attribute.`
`value`	`The value of this attribute is the return data type value from the script function. For example: string or integer or float or double.`

Figure 3.16. Custom Python Script Example

```
<DefineFunction name="pythonCustomFunc" optype="categorical" dataType="string">
    <ParameterField name="image" dataType="binary"/>
    <Apply function="python">
        <Extension extender="ADAPA" name="getBase64EncodedString" value="string">
def from_floatArray(floatArray, nlPos = 0):
 if sys.version_info >= (3,0):
  if nlPos > 0:
   result = ""
   nl = nlPos
   fArray = array('f')
   for i in range(0, len(floatArray)):
    fArray.append(floatArray[i])
    nl -= 1
    if le(n1,0):
     result += str(base64.standard_b64encode(fArray), 'utf-8') + "\n"
     nl = nlPos
     fArray = array('f')
   result += str(base64.standard_b64encode(fArray), 'utf-8')
   return result
  else:
   result = ""
   fArray = array('f')
   for i in range(0, len(floatArray)):
    fArray.append(floatArray[i])
   result += str(base64.standard_b64encode(fArray), 'utf-8')
   return result
 else:
  if nlPos > 0:
   result = ""
   nl = nlPos
   fArray = array('f')
   for i in range(0, len(floatArray)):
    fArray.append(floatArray[i])
    nl -= 1
    if le(n1,0):
     result += base64.standard_b64encode(fArray) + "\n"
     nl = nlPos
     fArray = array('f')
   result += base64.standard_b64encode(fArray)
   return result
  else:
   result = ""
   fArray = array('f')
   for i in range(0, len(floatArray)):
    fArray.append(floatArray[i])
   result += base64.standard_b64encode(fArray)
 return result
def getBase64EncodedString(input):
 from PIL import Image
 with Image.open(input) as img:
  width, height = img.size
  pix=img.load()
  x=list(img.getdata())
 pixels = list()
 for t in x:
  R,G,B=t
  for pix in [R, G, B]:
   pixels.append(pix / 127.5 - 1.0)

 myarray = np.asarray(pixels)
 return from_floatArray(myarray)
        </Extension>
 <FieldRef field="image"/>
    </Apply>
</DefineFunction>
```

Figure 3.17. Custom Python Script in Base64 Encoded Format

```
<DefineFunction name="customFunc" optype="categorical" dataType="string">
  <ParameterField name="image" dataType="binary"/>
 <Apply function="python">
  <Extension extender="ADAPA" name="getBase64EncodedString" value="string">
  ZGVmIGZyb21fZmxvYXRBcnJheShmbG9hdEFycmF5LCBubFBvcyA9IDApOg0KICAgICIiImNvbnZlcnRzIHRoZSBmbG9
  dEFycmF5IGludG8gYSBiYXNlNjQgc3RyaW5nOyBubFBvczogaW5zZXJ0cyBcbiBhZnRlciBubFBvcyBmbG9hdHMgaWYg
  Z2l2ZW4gIiIiDQogICAgaW1wb3J0IHN5cw0KICAgIGZyb20gYXJyYXkgaW1wb3J0IGFycmF5DQogICAgaW1wb3J0IGJh
  c2U2NA0KICAgIGlmIHN5cy52ZXJzaW9uX2luZm8gPj0gKDMsMCk6DQogICAgICAgIGlmIG5sUG9zID4gMDoNCiAgICAg
  ICAgICAgIHJlc3VsdCA9ICIiDQogICAgICAgICAgICBubCA9IG5sUG9zDQogICAgICAgICAgICBmQXJyYXkgPSBhcnJh
  eSgnZicpDQogICAgICAgICAgICBmb3IgaSBpbiByYW5nZSgwLCBsZW4oZmxvYXRBcnJheSkpOg0KICAgICAgICAgICAg
  ICAgIGZBcnJheS5hcHBlbmQoZmxvYXRBcnJheVtpXSkNCiAgICAgICAgICAgICAgICBubCAtPSAxDQogICAgICAgICAg
  ICAgICAgaWYgbGUobjEsMCk6DQogICAgICAgICAgICAgICAgICAgIHJlc3VsdCArPSBzdHIoYmFzZTY0LnN0YW5kYXJk
  X2I2NGVuY29kZShmQXJyYXkpLCAndXRmLTgnKSArICJcbiINCiAgICAgICAgICAgICAgICAgICAgbmwgPSBubFBvcw0K
  ICAgICAgICAgICAgICAgICBmQXJyYXkgPSBhcnJheSgnZicpDQogICAgICAgICAgICByZXN1bHQgKz0gc3RyKGJh
  c2U2NC5zdGFuZGFyZF9iNjRlbmNvZGUoZkFycmF5KSwgJ3V0Zi04JykNCiAgICAgICAgICAgIHJldHVybiByZXN1bHQN
  CiAgICAgICAgZWxzZToNCiAgICAgICAgICAgIHJlc3VsdCA9ICIiDQogICAgICAgICAgICBmQXJyYXkgPSBhcnJheSgn
  ZicpDQogICAgICAgICAgICBmb3IgaSBpbiByYW5nZSgwLCBsZW4oZmxvYXRBcnJheSkpOg0KICAgICAgICAgICAgICAg
  IGZBcnJheS5hcHBlbmQoZmxvYXRBcnJheVtpXSkNCiAgICAgICAgICAgIHJlc3VsdCArPSBzdHIoYmFzZTY0LnN0YW5k
  YXJkX2I2NGVuY29kZShmQXJyYXkpLCAndXRmLTgnKQ0KICAgICAgICAgICAgcmV0dXJuIHJlc3VsdA0KICAgIGVsc2U6
  DQogICAgICAgIGlmIG5sUG9zID4gMDoNCiAgICAgICAgICAgIHJlc3VsdCA9ICIiDQogICAgICAgICAgICBubCA9IG5s
  UG9zDQogICAgICAgICAgICBmQXJyYXkgPSBhcnJheSgnZicpDQogICAgICAgICAgICBmb3IgaSBpbiByYW5nZSgwLCBs
  ZW4oZmxvYXRBcnJheSkpOg0KICAgICAgICAgICAgICAgIGZBcnJheS5hcHBlbmQoZmxvYXRBcnJheVtpXSkNCiAgICAg
  ICAgICAgICBubCAtPSAxDQogICAgICAgICAgICAgICAgaWYgbGUobjEsMCk6DQogICAgICAgICAgICAgICAgICAg
  IHJlc3VsdCArPSBiYXNlNjQuc3RhbmRhcmRfYjY0ZW5jb2RlKGZBcnJheSkgKyAiXG4iDQogICAgICAgICAgICAgICAg
  ICAgIG5sID0gbmxQb3MNCiAgICAgICAgICAgICAgICAgICAgZkFycmF5ID0gYXJyYXkoJ2YnKQ0KICAgICAgICAgICAg
  cmVzdWx0ICs9IGJhc2U2NC5zdGFuZGFyZF9iNjRlbmNvZGUoZkFycmF5KQ0KICAgICAgICAgICAgcmV0dXJuIHJlc3Vs
  dA0KICAgICAgICBlbHNlOg0KICAgICAgICAgICAgcmVzdWx0ID0gIiINCiAgICAgICAgICAgIGZBcnJheSA9IGFycmF5
  KCdmJykNCiAgICAgICAgICAgIGZvciBpIGluIHJhbmdlKDAsIGxlbihmbG9hdEFycmF5KSk6DQogICAgICAgICAgICAg
  ICAgZkFycmF5LmFwcGVuZChmbG9hdEFycmF5W2ldKQ0KICAgICAgICAgICAgcmVzdWx0ICs9IGJhc2U2NC5zdGFuZGFy
  ZF9iNjRlbmNvZGUoZkFycmF5KQ0KICAgIHJldHVybiByZXN1bHQNCmRlZiBnZXRCYXNlNjRFbmNvZGVkU3RyaW5nKGlu
  cHV0KToNCiAgICBpbXBvcnQgc3lzDQogICAgaW1wb3J0IGFycmF5IGFzIGFycg0KICAgIGltcG9ydCBzdHJ1Y3QNCiAg
  ICBpbXBvcnQgYmFzZTY0DQogICAgZnJvbSBQSUwgaW1wb3J0IEltYWdlDQogICAgaW1wb3J0IG51bXB5IGFzIG5wDQog
  ICAgd2l0aCBJbWFnZS5vcGVuKGlucHV0KSBhcyBpbWc6DQogICAgICAgIHdpZHRoLCBoZWlnaHQgPSBpbWcuc2l6ZQ0K
  ICAgICAgICBwaXg9aW1nLmxvYWQoKQ0KICAgICAgICB4PWxpc3QoaW1nLmdldGRhdGEoKSkNCiAgICBwaXhlbHMgPSBs
  aXN0KCkNCiAgICBmb3IgdCBpbiB4Og0KICAgICAgICBSLEcsQj10DQogICAgICAgIGZvciBwaXggaW4gW1IsIEcsIEJd
  Og0KICAgICAgICAgICAgcGl4ZWxzLmFwcGVuZChwaXggLyAxMjcuNSAtIDEuMCkNCiAgICAgICAgDQogICAgbXlhcnJh
  eSA9IG5wLmFzYXJyYXkocGl4ZWxzKQ0KICAgIHJldHVybiBmcm9tX2Zsb2F0QXJyYXkobXlhcnJheSk=
  </Extension>
  <FieldRef field="image"/>
    </Apply>
</DefineFunction>
```

3.6.2. Use python script function from PMML

Once the python script is defined as in Figure 3.16, it can be used within the `Apply` transformations. The PMML fragment in Figure 3.18 contains an example that uses the python function described in Figure 3.16

Figure 3.18. Using python script function in PMML

```
<DeepNetwork modelName="DeepNetwork Sample" functionName="classification" numberOfLayers="2">
  ...
  <LocalTransformations>
    <DerivedField name="input_base64String" optype="categorical" dataType="string">
      <Apply function="pythonCustomFunc">
        <FieldRef field="input_image"/>
      </Apply>
    </DerivedField>
  </LocalTransformations>
  <NetworkLayer layerType="Input" layerId="layer_1" connectionLayerId="na"
 inputFieldName="input_base64String">
    <LayerParameters inputDimension="(2, 2, 3)" outputDimension="(2, 2, 3)"/>
  </NetworkLayer>
  ...
</DeepNetwork>
```

Important

This feature can be supported only if the host system contains python installation. Please refer **Section 4.5** in the Deployment Guide for configuring python support in PMML.

Caution

The embedded Python scripts in PMML are loaded and executed dynamically at runtime. This could result in serious security risks which can lead to system compromise. It is highly recommended that users upload such PMML files from a trusted source like Nyoka.

Chapter 4. Extensions API

Zementis Server has been designed to easily support customizations and/or extensions needed to meet the requirements imposed by the target environment. Using the popular Spring Framework, it allows injecting external resources either as configuration modifications or as extensions. This means that Zementis Server can be customized by providing an appropriate Spring context file along with the necessary custom implementations and required libraries. In the following sections, the Zementis Server Java Extensions API is described, which can be implemented to provide custom resources (Custom Functions and Lookup Tables), custom asset repository and a custom logging store for Zementis Server.

4.1. Using the Zementis Server Extensions API

Using Zementis Server Extensions API, you can provide a custom implementation for the following:

- Custom Function
- Lookup Table
- Asset Repository
- Logging Store

The following sections will describe each of these items in detail. Section 4.2 will provide details about how the Extensions API and sample implementations are packaged with the `adapa-app-10.7.0.2.zip` distribution.

4.1.1. Custom Function

Zementis Server provides a facility to create and use custom PMML functions. This capability enables, for example, the implementation of intricate calculations that cannot be easily described in PMML, functions that access external systems to retrieve necessary data, or even specialized algorithms not supported by PMML.

The `Function<T>` interface represents a custom function which can be called from PMML. This function can be referenced by the name returned by the `getName()` method and it operates on the arguments provided in the `evaluate(Object...)` method. It returns a value of the specified type `T`. A sample implementation of this interface is contained in `CalcSomething.java` which demonstrates a custom function that can operate on several (at least 2) numeric arguments and returns a value of type Double.

The `Function.Factory` interface provides a factory method for creating `Function` instances with the method `createFunction(String functionName, Class<?> ... argumentTypes)`. The `Function.getName()`

method must match parameter `functionName` and `Function.evaluate(Object...)` must be able to operate on parameter `argumentTypes`. A sample implementation of this interface is contained in `CalcSomethingFactory.java` which creates functions that can operate over a variable number (but at least two) of numeric arguments.

Please add the following dependencies as listed under Figure 4.1 when packaging the project as a JAR. Make sure `${project.version}` resolves to `10.7.0.2`.

Figure 4.1. Dependencies for Custom Functions

```
<dependencies>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-extensions</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-api</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-bundle</artifactId>
  <version>${project.version}</version>
 </dependency>
</dependencies>
```

4.1.2. Lookup Table

Predictive models can sometimes require the use of lookup tables. If relatively small and static, these tables can be easily embedded within the PMML file itself. However, if they are fairly large and/or they are modified frequently, it is more practical to create and manage them separately.

The `LookupTable` interface represents a lookup table that can be called from PMML. This lookup table can be referenced by the name returned by the `getName()` method. The lookup table implementation can be used to retrieve an output value identified by column name with `getOutputColumnName()`. This can be done by looking up provided input values which are identified by column names with `getInputColumnNames()`. The order of input values for the `lookup(Object...)` method must match the order of column names returned by the `getInputColumnNames()` method. A sample implementation of this interface is contained in `GDPLookupTable.java` which returns a GDP number corresponding to two inputs, Country and State by querying a database table.

Please add the following dependencies as listed under Figure 4.2 when packaging the project as a JAR. Make sure `${project.version}` resolves to `10.7.0.2`.

Figure 4.2. Dependencies for Lookup Table

```
<dependencies>
 <dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.6</version>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-extensions</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-api</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-bundle</artifactId>
  <version>${project.version}</version>
 </dependency>
</dependencies>
```

4.1.3. Asset Repository

The `AssetRepository` interface provides methods for managing Zementis Server assets on a back-end storage. By default, Zementis Server uses a file-based repository to store the uploaded artifacts (models and resources). Zementis Server also provides support for a database-based repository by using the Java Persistence API (JPA) in conjunction with using Hibernate as the JPA provider. A traditional Database can be plugged-in as a repository store for Zementis Server by providing an appropriate configuration file.

On top of this, Zementis Server also allows users to provide a custom back-end store (e.g. MongoDB) by implementing this interface. A sample implementation is contained in `MongoAssetRepository.java`. As shown in the sample implementation, the `addAsset(Serializable, InputStream)` method requires assignment of a unique identifier to the provided Zementis Server asset. The choice of unique identifier is left to the implementor. The implementation of this interface needs to be in the classpath of Zementis Server library along with any required JDBC drivers.

Please add the following dependencies as listed under Figure 4.3 when packaging the project as a JAR. Make sure `${project.version}` resolves to `10.7.0.2`.

Figure 4.3. Dependencies for Asset Repository

```
<dependencies>
 <dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongodb-driver</artifactId>
  <version>3.3.0</version>
 </dependency>
 <dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.5</version>
 </dependency>
 <dependency>
  <groupId>commons-lang</groupId>
  <artifactId>commons-lang</artifactId>
  <version>2.6</version>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-extensions</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-api</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-bundle</artifactId>
  <version>${project.version}</version>
 </dependency>
</dependencies>
```

4.1.4. Logging Store

Information about records processed by Zementis Server can be logged in a file system or database. The captured data includes input and output values as well as information regarding invalid and missing values presented to the model for execution. The logging mechanism can be enabled and configured for file-based or database store by providing an appropriate Spring configuration file as described in the Zementis Server Deployment Guide.

On top of this, Zementis Server also allows users to provide a custom logging store by implementing the `ModelLogHandler` interface. This interface represents a handler for logging records that a model processes. This interface can be implemented to log entire records, invalid values and missing values. A sample implementation of this interface is contained in `FileLogHandler.java`. This implementation logs every record to a file as soon as the record is processed. The implementation also logs a counter for missing and invalid values for a given field. The logging of missing and invalid values is done when method `flush()` is invoked.

Note

The implementor is responsible for the invocation of `flush()` and for ensuring the thread safety of any state which is maintained before `flush()` is invoked. The code samples are for illustration purposes only.

The `ModelLogHandler.Factory` interface provides a factory method for creating ModelLogHandler instances. A sample implementation of this interface is contained in FileLogHandlerFactory.java.

Please add the following dependencies as listed under Figure 4.4 when packaging the project as a JAR. Make sure `${project.version}` resolves to `10.7.0.2`.

Figure 4.4. Dependencies for Logging Repository

```
<dependencies>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-extensions</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-api</artifactId>
  <version>${project.version}</version>
  <scope>provided</scope>
 </dependency>
 <dependency>
  <groupId>com.zementis.adapa</groupId>
  <artifactId>adapa-bundle</artifactId>
  <version>${project.version}</version>
 </dependency>
</dependencies>
```

4.2. Overview of code examples

The files under directory `adapa-extensions/samples` offer Java code examples for each use case. Table 4.1 describes all the sample files in detail.

Table 4.1. Directory structure of code examples

Directory	Files	Description
customfunction	`applicationContext.xml`	The application context XML file to be included.
	`CalcSomething.java`	The CalcSomething function calculates something over several numeric arguments. In order to support Double, Float, and Long arguments, it uses reflection to enable invocation of the appropriate "doubleValue" method at runtime.

Directory	Files	Description
	`CalcSomethingFactory.java`	Example of a custom function factory which creates functions that can compute something over a variable number (but at least two) of numeric arguments.
loghandler	`applicationContext.xml`	The application context XML file to be included.
	`FileLogHandler.java`	Contains methods for custom record logging.
	`FileLogHandlerFactory.java`	Factory for custom record logging.
lookuptable	`applicationContext.xml`	The application context XML file to be included.
	`GDPLookupTable.java`	The lookup table returns a GDP number corresponding to Country and State. Country, State and GDP are columns in the database table GDP_Table.
repository	`applicationContext.xml`	The application context XML file to be included.
	`MongoAssetRepository.java`	A sample AssetRepository for MongoDB.

4.3. Deployment of Zementis Server Extensions

Once the new Zementis Server extension is created, the Java code needs to be packaged as a JAR together with all depending libraries. Once the JAR file is created, copy it in the directory `ADAPA_HOME/adapa-lib`. This directory must also contain `adapa-extensions-10.7.0.2.jar` file. The new code can then be integrated into Zementis Server by using a Spring configuration file as described in the respective `applicationContext.xml`. This context file needs to be copied to the working directory of the server. One or more context files may be used. In case there are multiple context files, rename them as per the extension it configures (For example, adapaContextLogging.xml or adapaContextRepository.xml). For configuration purposes and upon start-up, Zementis Server will examine any files in the server's working directory following the name pattern `adapaContext*.xml`. Please note that configuration changes through context files require a server restart before they can take effect.

Chapter 5. REST API

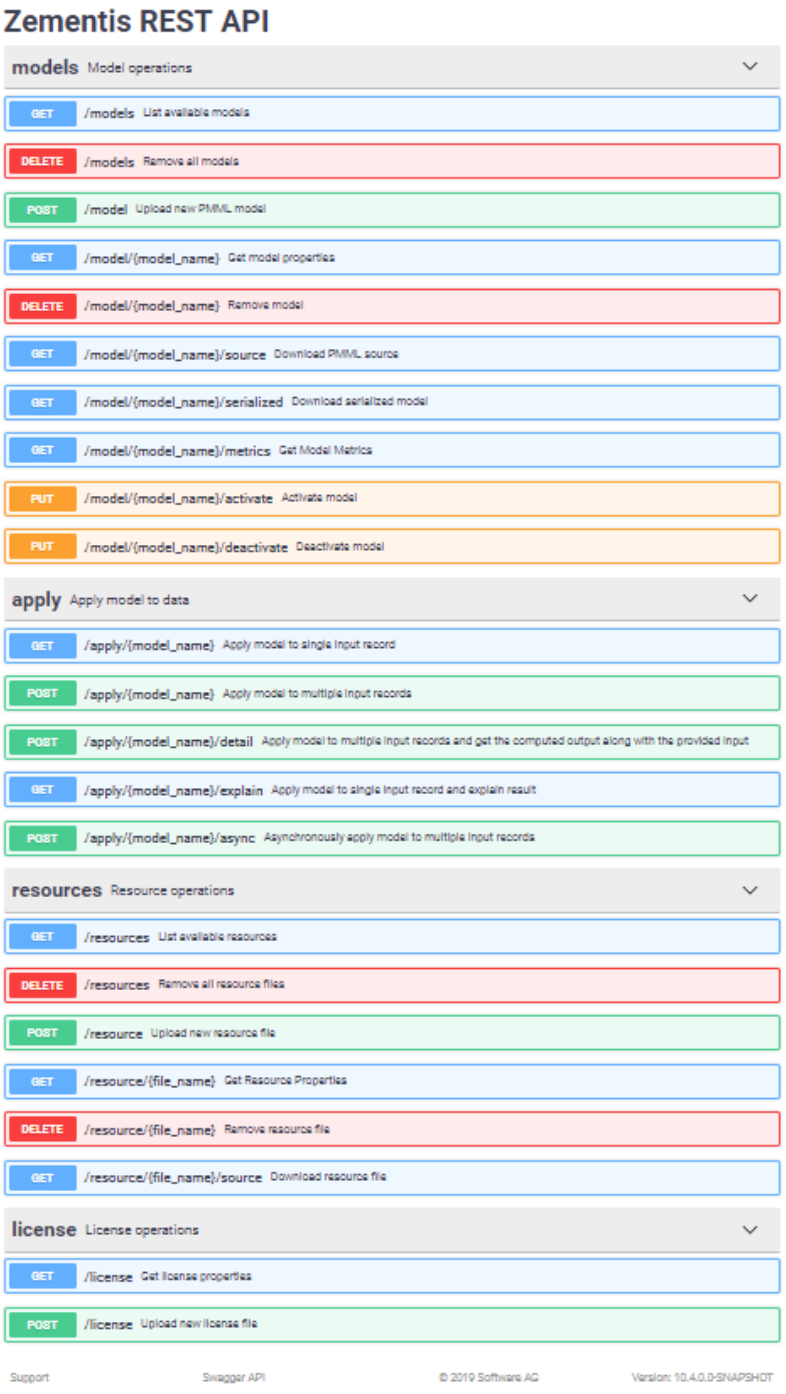
This Application Programming Interface (API) provides users with a comprehensive set of defined interfaces to interact with Zementis Server using Representational State Transfer (REST) over Hypertext Transfer Protocol (HTTP). Zementis REST API allows users to perform operations on models and custom resources, and process data by issuing a simple request using any `HTTP` client such as a web browser.

5.1. General Notes

5.1.1. URI

A full path to the Zementis REST API resource consists of a base path and a resource path. The base path Uniform Resource Identifier (URI) for the Zementis REST API is `http://domain:port/adapars`, where `http` or `https` is the protocol name, `domain` is the internet domain or network address, `port` is a non-negative integer representing the port number, and `adapars` represents the application context path. The base path is static and does not change between requests; it merely identifies the server with an application on the network. Connecting with your favorite web browser to the base path `URI` will load Zementis REST interactive API documentation that describes all available resources, enables request execution and displays received responses from the Zementis REST service. See Figure 5.1.

Figure 5.1. Interactive REST API Documentation



Following the base path is the resource path. It may contain path or query parameters depending on the type of the request and available resources on the server. For example, a resource path `/model/Iris_NN/source?annotated=true` contains static path definitions such as `model` or `source`, path parameter `Iris_NN` for a dynamically allocated resource, and a query parameter `annotated=true`.

5.1.2. Request

The HTTP request is a combination of a simple Uniform Resource Identifier (URI), HTTP verb GET, POST, PUT, or DELETE, request parameters, which can be in the form of a path variable, query, body, or header parameters, and message body (content). The path variable is a variable part of otherwise static URI that denotes a set of possible resource names on the server and is denoted with curly braces. For example, our `/model/{model_name}/source` resource path specifies the PMML file for an arbitrary model denoted as `{model_name}`. Thus, the request path for the PMML file of model `Iris_NN` should be constructed as `/model/Iris_NN/source`. Query parameters are appended to the URI with a question mark followed by a list of key/value pairs. A query variable `annotated` with the value `true` in the `/model/Iris_NN?annotated=true` resource path specifies that the returned PMML file should contain annotations as placed by Zementis Server, in case of errors or warnings. Header parameters are HTTP message metadata in the form of key/value pairs containing information about the message such as content type, message encoding type, authorization, etc. Body parameters appear only in POST or PUT requests and need to be encoded by the HTTP client.

Please, refer to HTTP 1.1 specification for details.

5.1.3. Response

The HTTP response message is composed of a message header and a message body. All Zementis REST response content types implement standard UTF-8 character set encoding. The header contains response status code and header fields represented as list of key/value pairs, i.e. `Content-Type:application/json`.

Every response from Zementis REST contains a `Content-Type` header entry with one of following internet media types (aka MIME) as value.

- `application/json`
- `application/xml`
- `text/plain`
- `application/zip`

5.1.4. Errors

Zementis REST maps error responses to appropriate HTTP status codes and returns a Javascript Object Notation (JSON) Errors object in the response body containing an array of error messages. For example, if the requested model, e.g. `Iris_NN`, has not been uploaded into Zementis Server yet, a response header with status code `404` and its following response body with Errors are returned.

Example 5.1. Zementis REST Error Response

```
Request

   curl -u adapa:adapa  -k https://localhost/adapars/model/Iris_NN

Request Header

   GET /adapars/model/Iris_NN HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 404 Not Found
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 49
   Date: Thu, 27 Mar 2014 20:22:14 GMT

Response Body

   {
     "errors" : [ "Model 'Iris_NN' not found." ]
   }
```

Table 5.1. Typical Zementis REST Error Responses

Code	Error Message
400	Empty input stream.
400	File name missing.
400	Invalid XML format.
400	Failed to parse JSON input.
400	Invalid CSV File.
401	This request requires HTTP authentication.
403	You are not authorized to access this resource.
404	Model 'model_name' not found.
404	Resource 'file_name' not found.
409	A model with the name 'model_name' already exists.
409	A resource file with the name 'file_name' already exists.

Code	Error Message
409	A resource with the name 'resource_name' already exists.
500	Invalid License.
500	Internal server error.

5.1.5. Authorization

All requests are authorized by the basic access authentication method. For example, `HTTP` header entry `Authorization: Basic YWRhcGE6YWRhcGE=` is created for credentials with user name and password `adapa`. If the provided credentials fail to authenticate, the `HTTP 401` response code is returned, and `403` if the user is not authorized to perform the requested operation. The below table lists the authorized role(s) for each operation, and the detailed description of each operation can be found in the following sections.

Table 5.2. Zementis REST Permissions

Operation	Definition	Authorized Role(s)
List Available Models	GET /models	`adapa-admin`, `adapa-ws-user`
Get Model Information	GET /model/{model_name}	`adapa-admin`, `adapa-ws-user`
Get Model Source	GET /model/{model_name}/source	`adapa-admin`
Get Model Serialized Source	GET /model/{model_name}/serialized	`adapa-admin`
Get Model Metrics Information	GET /model/{model_name}/metrics	`adapa-admin`, `adapa-ws-user`
Upload New Model with POST	POST /model	`adapa-admin`
Upload New Model with PUT	PUT /model	`adapa-admin`
Activate an existing Model with PUT	PUT /model/{model_name}/activate	`adapa-admin`
Deactivate an existing Model with PUT	PUT /model/{model_name}/deactivate	`adapa-admin`
Remove Model	DELETE /model/{model_name}	`adapa-admin`
Remove All Models	DELETE /models	`adapa-admin`
Apply Model to Single Record	GET /apply/{model_name}	`adapa-admin`, `adapa-ws-user`
Apply Model to Single Record and Explain Result	GET /apply/{model_name}/explain	`adapa-admin`, `adapa-ws-user`

Operation	Definition	Authorized Role(s)
Apply Model to Multiple Records or Apply Model to Single Binary Data	POST /apply/{model_name}	`adapa-admin`, `adapa-ws-user`
Apply Model to Multiple Records or Apply Model to Single Binary Data with PUT	PUT /apply/{model_name}	`adapa-admin`, `adapa-ws-user`
Asynchronously Apply Model to Multiple Records	POST /apply/{model_name}/async	`adapa-admin`, `adapa-ws-user`
Asynchronously Apply Model to Multiple Records with PUT	PUT /apply/{model_name}/async	`adapa-admin`, `adapa-ws-user`
List Available Resources	GET /resources	`adapa-admin`, `adapa-ws-user`
Get Resource Information	GET /resource/{file_name}	`adapa-admin`, `adapa-ws-user`
Get Resource File	GET /resource/{file_name}/source	`adapa-admin`
Upload New Resource File with POST	POST /resource	`adapa-admin`
Upload New Resource File with PUT	PUT /resource	`adapa-admin`
Remove Resource File	DELETE /resource/{file_name}	`adapa-admin`
Remove All Resource Files	DELETE /resources	`adapa-admin`
Get License	GET /license	`adapa-admin`
Post License	POST /license	`adapa-admin`

5.2. API

Zementis REST has three APIs denoted by static path identifiers: `models`, `apply`, `and resources`. Requests in the following examples employ syntax for `cURL`, a popular command line data transfer tool for Unix-like systems, and use username/password credentials `adapa/adapa` with user permissions to execute all REST API operations. All examples also include `Iris_NN` PMML model which can be found in the executable samples package.

5.2.1. JSON Objects

Errors Error messages container

Properties

errors (array[string]): array of strings containing error messages

Example 5.2. Zementis REST Errors Object

```
{
   "errors": [
      "Model 'Iris_NN' not found."
   ]
}
```

Models

Model names container

Properties

models (array[string]): array of strings containing model names

Example 5.3. Zementis REST Models Object

```
{
   "models": [
      "Iris_NN",
      "Iris_CT",
      "Iris_MLR"
   ]
}
```

ModelInfo

Model information

Properties

modelName (string): model name

description (string): model description

isActive (boolean): model currently loaded into memory

inputFields (array[Field]): array of input Field objects

outputFields (array[Field]): array of output Field objects

Example 5.4. Zementis REST ModelInfo Object

```
{
   "modelName": "Iris_NN",
   "description": "Neural Network for multi-class classification using the Iris
 dataset",
   "isActive": true,
   "inputFields": [
      {
         "name": "sepal_length",
         "type": "DOUBLE",
         "usage": "ACTIVE"
      },
      {
         "name": "sepal_width",
         "type": "DOUBLE",
         "usage": "ACTIVE"
      },
      {
         "name": "petal_length",
         "type": "DOUBLE",
         "usage": "ACTIVE"
      },
      {
         "name": "petal_width",
         "type": "DOUBLE",
         "usage": "ACTIVE"
      }
   ],
   "outputFields": [
      {
         "name": "class",
         "type": "STRING",
         "usage": "OUTPUT"
      },
      {
         "name": "Probability_setosa",
         "type": "DOUBLE",
         "usage": "OUTPUT"
      },
      {
         "name": "Probability_versicolor",
         "type": "DOUBLE",
         "usage": "OUTPUT"
      },
      {
         "name": "Probability_virginica",
         "type": "DOUBLE",
         "usage": "OUTPUT"
      }
   ]
}
```

Field

Field information

Properties

`name (string)`: field name

`type (string)`: field data type with one of string values: `BOOLEAN`, `INTEGER`, `FLOAT`, `DOUBLE`, `DATE`, `DATETIME`, `TIME`, or `STRING`

`usage (string):` field usage type with one of string values: `ACTIVE`, `SUPPLEMENTARY`, `TARGET`, `GROUP`, `DERIVED`, or `OUTPUT`

Example 5.5. Zementis REST Field Object

```
{
   "name": "petal_width",
   "type": "DOUBLE",
   "usage": "ACTIVE"
}
```

MetricsInfo

Model Metrics information

Properties

`modelSize (string):` model size

`usedMemory (string):` used memory

`freeMemory (string):` free memory

`totalMemory (string):` total memory

`predictionMetrics (object):` Object used to represent prediction metrics as a set of key/value pairs

Example 5.6. Zementis REST MetricsInfo Object

```
{
  "modelSize": ".006 MB",
  "usedMemory": "1126.692 MB",
  "freeMemory": "3993.302 MB",
  "totalMemory": "5120.0 MB",
  "predictionMetrics": {
    "Iris-setosa": 50,
    "Iris-versicolor": 50,
    "Iris-virginica": 50
  }
}
```

Record

Object used to represent input or output data record as a set of field/value pairs.

Properties

`field_name_1 (string):` optional field/value pair

`field_name_2 (number):` optional field/value pair

`field_name_3 (boolean):` optional field/value pair

`field_name... (date-time):` optional field/value pair

`field_name_n (array[string]):` optional field/value pair

Example 5.7. Zementis REST Record Object

```
{
   "probability": 0.99995417336,
   "days": 47,
   "class": "shirt",
   "time": "2010-07-14 09:00:02",
   "colors": [ "white", "red", "yellow" ]
}
```

Records

Anonymous array of Record objects used to represent multiple input or output records.

Example 5.8. Zementis REST Record Object

```
[
   {
      "Probability_virginica": 2.536692637033178E-13,
      "class": "Iris-setosa",
      "Probability_setosa": 0.9995535104664939,
      "Probability_versicolor": 4.464895332525406E-4
   },
   {
      "Probability_virginica": 1.0465677336558733E-12,
      "class": "Iris-setosa",
      "Probability_setosa": 0.9985890830740689,
      "Probability_versicolor": 0.0014109169248845744
   },
   {
      "Probability_virginica": 4.111504068226951E-13,
      "class": "Iris-setosa",
      "Probability_setosa": 0.9993451737365701,
      "Probability_versicolor": 6.54826263018726E-4
   }
]
```

Result

Object used to return the result of applying a model to data.

Properties

`modelName (string):` model name

`outputs (array[Record]):` array of output Record objects

Example 5.9. Zementis REST Result Object


```
{
   "model": "Iris_NN",
   "outputs": [
      {
         "Probability_virginica": 2.536692637033178E-13,
         "class": "Iris-setosa",
         "Probability_setosa": 0.9995535104664939,
         "Probability_versicolor": 4.464895332525406E-4
      },
      {
         "Probability_virginica": 1.0465677336558733E-12,
         "class": "Iris-setosa",
         "Probability_setosa": 0.9985890830740689,
         "Probability_versicolor": 0.0014109169248845744
      },
      {
         "Probability_virginica": 4.111504068226951E-13,
         "class": "Iris-setosa",
         "Probability_setosa": 0.9993451737365701,
         "Probability_versicolor": 6.54826263018726E-4
      },
      {
         "Probability_virginica": 6.620361333170605E-13,
         "class": "Iris-setosa",
         "Probability_setosa": 0.9990465573403722,
         "Probability_versicolor": 9.534426589658814E-4
      }
   ]
}
```


ResourceInfo — Resource file information

Properties

`fileName (string)`: file name

`resourceType (string)`: resource type

`resourceIdentifier (string)`: resource identifier

`resourceNames (array[string])`: array of resource names

Example 5.10. Zementis REST ResourceInfo Object


```
{
  "fileName": "ECommerceFraud_NN.xls",
  "resourceType": "Lookup Tables",
  "resourceIdentifier": "Table Name",
  "resourceNames": [
    "StatePoints"
  ]
}
```


Resources — Anonymous array of ResourceInfo objects.

Example 5.11. Zementis REST Resources Object

```
{
  "resources": [
    {
      "fileName": "ECommerceFraud_NN.xls",
      "resourceType": "Lookup Tables",
      "resourceIdentifier": "Table Name",
      "resourceNames": [
        "StatePoints"
      ]
    },
    {
      "fileName": "ECommerceFraud_NN.jar",
      "resourceType": "Custom Functions",
      "resourceIdentifier": "Function Namespace",
      "resourceNames": [
        "fraud"
      ]
    }
  ]
}
```

5.2.2. Operations on Models

5.2.2.1. List Available Models

Definition	**GET /models** This operation retrieves the model names of all the available PMML models in Zementis Server. Use these model names as identifiers for all operations requiring the `model_name` path variable.
Request Parameters	None
Returns	Returns Models object if successful, Errors otherwise.

Example 5.12. Zementis REST List Models

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/models

Request  Header

   GET /adapars/models HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
```

```
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 53
   Date: Wed, 26 Mar 2014 18:20:09 GMT

Response Body

   {
     "models" : [ "Iris_NN", "Iris_CT", "Iris_MLR" ]
   }
```

5.2.2.2. Get Model Information

Definition	**GET /model/{model_name}**
	Get model name, description, and information about input, output, or derived fields.
Request Parameters	**`model_name (string)`:** required path variable for existing model name
Returns	Returns ModelInfo object if successful, Errors otherwise.

Example 5.13. Zementis REST Get Model Information

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/model/Iris_NN

Request  Header

   GET /adapars/model/Iris_NN HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 1969
   Date: Wed, 26 Mar 2014 18:39:57 GMT

Response Body

   {
    "modelName" : "Iris_NN",
    "description" : "Neural Network for multi-class classification using the Iris dataset",
    "isActive": true,
    "inputFields" : [ {
      "name" : "sepal_length",
      "type" : "DOUBLE",
```

```
      "usage" : "ACTIVE"
    }, {
      "name" : "sepal_width",
      "type" : "DOUBLE",
      "usage" : "ACTIVE"
    }, {
   ...
```

5.2.2.3. Get Model Source

Definition	**GET /model/{model_name}/source** Get annotated or original PMML file. Annotated source may contain warning or error messages embedded in XML comments that are useful for verifying that the PMML code is correct.
Request Parameters	**`model_name (string):`** required path variable for existing model name **`annotated (boolean):`** optional query parameter used to request the annotated version of the PMML file.
Returns	Returns the PMML source code if successful, Errors otherwise.

Example 5.14. Zementis REST Get Model Source

```
Request

   curl -u adapa:adapa  -k https://localhost/adapars/model/Iris_NN/source?annotated=true

Request  Header

   GET /adapars/model/Iris_NN/source?annotated=true HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/xml
   Content-Length: 7983
   Date: Wed, 26 Mar 2014 20:44:04 GMT

Response Body

   <?xml version="1.0" encoding="UTF-8"?>
   <!--(Comment generated by ADAPA) PMML processed by ADAPA (Version : 4.2)-->
   <PMML version="4.2"
          xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2/pmml-4-2.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://www.dmg.org/PMML-4_2">
     <Header copyright="Copyright (c) 2008-2014 Zementis, Inc. (www.zementis.com)"
```

```
            description="Neural Network for multi-class classification using the Iris dataset">
      <Timestamp>Feb 15, 2008</Timestamp>
    </Header>
    <DataDictionary numberOfFields="5">
      <DataField dataType="string" name="target_class" optype="categorical">
        <Value value="Iris-setosa"/>
        <Value value="Iris-versicolor"/>
        <Value value="Iris-virginica"/>
      </DataField>
      <DataField dataType="double" name="sepal_length" optype="continuous"/>
      <DataField dataType="double" name="sepal_width" optype="continuous"/>
  ...
```

5.2.2.4. Get Model Serialized Source

Definition	**GET /model/{model_name}/serialized**
	Get binary file containing serialized representation of the model.
Request Parameters	**`model_name (string)`:** required path variable for existing model name
Returns	Returns the binary file if successful, Errors otherwise.

Example 5.15. Zementis REST Get Model Serialized

```
Request

   curl -u adapa:adapa  -k https://localhost/adapars/model/Iris_NN/serialized

Request  Header

   GET /adapars/model/Iris_NN/serialized HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.54.0
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: private
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/octet-stream
   Transfer-Encoding: chunked
   Date: Wed, 09 Aug 2017 22:44:48 GMT

Response Body

   BINARY DATA
   ...
```

5.2.2.5. Get Model Metrics Information

Definition	**GET /model/{model_name}/metrics**

Get the memory metrics and prediction metrics of an uploaded model.

Request Parameters	**`model_name (string)`:** required path variable for existing model name
Returns	Returns MetricsInfo object if successful, Errors otherwise.

Example 5.16. Zementis REST Get Model Metrics

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/Iris_NN/metrics

Request  Header

   GET /adapars/Iris_NN/metrics HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 1969
   Date: Wed, 26 Mar 2014 18:39:57 GMT

Response Body

   {
   "modelSize": ".006 MB",
   "usedMemory": "1126.692 MB",
   "freeMemory": "3993.302 MB",
   "totalMemory": "5120.0 MB",
   "predictionMetrics": {
     "Iris-versicolor": 50,
     "Iris-virginica": 50,
     "Iris-setosa": 50
   }
   }
```

5.2.2.6. Upload New Model

Definition	**POST /model**

Upload new PMML model. Resulting identifier for this model is extracted from optional PMML attribute `modelName` if specified or `file` body parameter name otherwise. If the PMML file is large, such as Random Forest model, we recommend compressing the file using ZIP/GZIP before uploading. This will reduce the upload time dramatically.

Request Parameters	**`Content-Type (string):`** required header parameter with two accepted values: `application/octet-stream` or `multipart/form-data` **`file (string):`** required query parameter for PMML file name, if Content-Type is `application/octet-stream`, or a body parameter in `multipart/form-data` content encoding **`applyCleanser (boolean):`** optional parameter used to automatically perform comprehensive syntactic and semantic checks, correct known issues and convert your PMML file to version 4.4 (default is `true`)
Returns	Returns a ModelInfo object, `201` HTTP status code, and a response header entry `Location` with the URI of the created resource if the upload was successful. If the uploaded model was a valid XML but an invalid PMML, `200` HTTP status code and error annotated PMML source is returned, Errors otherwise.

Example 5.17. Zementis REST Upload New Model with POST

```
Request


   curl -u adapa:adapa -k https://localhost/adapars/model?file=Iris_NN.pmml -X POST -T Iris_NN.pmml \
        -H "Content-Type:application/octet-stream"
   curl -u adapa:adapa -k https://localhost/adapars/model -X POST -F file=@Iris_NN.pmml
   curl -u adapa:adapa -k https://localhost/adapars/model?applyCleanser=true -X POST -F
 file=@Iris_NN.pmml

Request  Header

   POST /adapars/model HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 9265
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=----------------------------1ff14caee8ae

Response Header

   HTTP/1.1 201 Created
   Server: Apache-Coyote/1.1
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Location: https://localhost/adapars/model/Iris_NN
   Content-Type: application/json
   Content-Length: 836
   Date: Wed, 26 Mar 2014 19:45:18 GMT

Response Body

   {
     "modelName" : "Iris_NN",
     "description" : "Neural Network for multi-class classification using the Iris dataset",
     "inputFields" : [ {
       "name" : "sepal_length",
       "type" : "DOUBLE",
       "usage" : "ACTIVE"
     }, {
       "name" : "sepal_width",
```

```
      "type" : "DOUBLE",
      "usage" : "ACTIVE"
    }, {
  ...
```

5.2.2.7. Upload New Model with PUT

Definition	**PUT /model** Upload new PMML model. Resulting identifier for this model is extracted from optional PMML attribute `modelName` if specified or `file` query parameter name otherwise. If the PMML file is large, such as the Random Forest model, we recommend compressing the file using ZIP/GZIP before uploading. This will reduce the upload time dramatically.
Request Parameters	**`file (string):`** required query parameter for PMML file name **`applyCleanser (boolean):`** optional parameter used to automatically perform comprehensive syntactic and semantic checks, correct known issues and convert your PMML file to version 4.4 (default is `true`)
Returns	Returns a ModelInfo object, `201` HTTP status code, and a response header entry `Location` with the URI of the created resource if the upload was successful. If the uploaded model was a valid XML but the PMML was invalid, a `200` HTTP status code and with errors annotated PMML file is returned, Errors otherwise.

Example 5.18. Zementis REST Upload New Model with PUT

```
Request

   curl -u adapa:adapa -k 'https://localhost/adapars/model?file=Iris_NN.pmml' -X PUT -T Iris_NN.pmml
   curl -u adapa:adapa -k 'https://localhost/adapars/model?file=Iris_NN.pmml&applyCleanser=true' -X PUT -
T Iris_NN.pmml

Request  Header

   PUT /adapars/model?file=Iris_NN.pmml HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 9061
   Expect: 100-continue

Response Header

   HTTP/1.1 201 Created
   Server: Apache-Coyote/1.1
```

```
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Location: https://localhost/adapars/model/Iris_NN
   Content-Type: application/json
   Content-Length: 836
   Date: Wed, 26 Mar 2014 19:56:26 GMT

Response Body

   {
     "modelName" : "Iris_NN",
     "description" : "Neural Network for multi-class classification using the Iris dataset",
     "inputFields" : [ {
       "name" : "sepal_length",
       "type" : "DOUBLE",
       "usage" : "ACTIVE"
     }, {
       "name" : "sepal_width",
       "type" : "DOUBLE",
       "usage" : "ACTIVE"
     }, {
   ...
```

5.2.2.8. Activate an existing Model with PUT

Definition	**PUT /model/{model_name}/activate** Activates the model with name `modelName` if it was inactive. Activating an active model has no effect. After activation, the model is immediately available for handling data processing requests. Please note an active model consumes runtime resources, especially Heap.
Request Parameters	**`model_name (string)`:** required path variable for existing model name
Returns	Returns a ModelInfo object and `200` HTTP status code.

Example 5.19. Zementis REST Activate an existing Model

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/model/Iris_NN/activate -X PUT

Request  Header

   PUT /adapars/model/Iris_NN/activate HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.54.0
   Host: localhost
   Accept: */*

Response Header
```

```
   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Cache-Control: private
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   Content-Type: application/json
   Content-Length: 7166
   Date: Wed, 09 Aug 2017 22:44:48 GMT

Response Body

   {
     "modelName" : "Iris_NN",
     "description" : "Neural Network for multi-class classification using the Iris dataset",
     "isActive" : true,
   ...
```

5.2.2.9. Deactivate an existing Model with PUT

Definition	**PUT /model/{model_name}/deactivate** De-activates the model with name `modelName` by making it inactive. After de-activation, the model is still available, but it no longer consumes runtime resources, especially Heap. Deactivating an inactive model has no effect.
Request Parameters	**`model_name (string)`:** required path variable for existing model name
Returns	Returns a ModelInfo object and `200` HTTP status code.

Example 5.20. Zementis REST Deactivate an existing Model

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/model/Iris_NN/deactivate -X PUT

Request  Header

   PUT /adapars/model/Iris_NN/deactivate HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.54.0
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Cache-Control: private
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   Content-Type: application/json
   Content-Length: 7166
   Date: Wed, 09 Aug 2017 22:44:48 GMT

Response Body

   {
```

```
    "modelName" : "Iris_NN",
    "description" : "Neural Network for multi-class classification using the Iris dataset",
    "isActive" : false,
  ...
```

5.2.2.10. Remove Model

Definition	**DELETE /model/{model_name}** Remove the specified model and list the remaining models.
Request Parameters	**`model_name (string):`** required path variable for existing model name
Returns	Returns a Models object with a list of remaining model names if successful, Errors object otherwise.

Example 5.21. Zementis REST Remove Model

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/model/Iris_NN -X DELETE

Request  Header

   DELETE /adapars/model/Iris_NN HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 42
   Date: Wed, 26 Mar 2014 19:53:50 GMT

Response Body

   {
     "models" : [ "Iris_CT", "Iris_MLR" ]
   }
```

5.2.2.11. Remove All Models

Definition	**DELETE /models**

	Remove all available models and list the remaining models.
Request Parameters	None
Returns	Returns a Models object with an empty `models` array if successful, an Errors object otherwise.

Example 5.22. Zementis REST Remove All Models

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/models -X DELETE

Request  Header

   DELETE /adapars/models HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 20
   Date: Wed, 26 Mar 2014 20:01:42 GMT

Response Body

   {
     "models" : [ ]
   }
```

5.2.3. Apply model

5.2.3.1. Apply Model to Single Record

Definition	**GET /apply/{model_name}** Apply a model to a single JSON input record.
Request Parameters	**`model_name (string)`:** required path variable for name of the model to be applied **`record ( Record ) :`** optional query parameter for input Record
Returns	Returns Result object if successful, Errors otherwise.

Example 5.23. Zementis REST Apply Model to Single Record

```
Request

  curl -u adapa:adapa -k  https://localhost/adapars/apply/Iris_NN -G --data-urlencode \
    'record={"sepal_length":5.1,"sepal_width":3.5,"petal_length":1.4,"petal_width":0.2}'

Request  Header

   GET /adapars/apply/Iris_NN?record=%7B%22sepal_length%22%3A5.1%2C%22sepal_width%22%3A3.5%2C%22
           petal_length%22%3A1.4%2C%22petal_width%22%3A0.2%7D HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 231
   Date: Wed, 26 Mar 2014 20:10:30 GMT

Response Body

   {
     "model" : "Iris_NN",
     "outputs" : [ {
       "Probability_virginica" : 2.536692637033178E-13,
       "class" : "Iris-setosa",
       "Probability_setosa" : 0.9995535104664939,
       "Probability_versicolor" : 4.464895332525406E-4
     } ]
   }
```

5.2.3.2. Apply Model to Single Record and Explain Result

Definition	**GET /apply/{model_name}/explain** Apply model to a single JSON input record and get the result with details of the performed computation in plain text. Useful for debugging PMML code.
Request Parameters	**`model_name (string)`:** required path variable for name of the model to be applied **`record ( Record ) :`** optional query parameter for input Record
Returns	Returns a result in plain text if successful, Errors otherwise.

Example 5.24. Zementis REST Apply Model to Single Record and Explain Result

```
Request
```

```
  curl -u adapa:adapa -k  https://localhost/adapars/apply/Iris_NN/explain -G --data-urlencode \
       'record={"sepal_length":5.1,"sepal_width":3.5,"petal_length":1.4,"petal_width":0.2}'

Request  Header

   GET /adapars/apply/Iris_NN/explain?record=%7B%22sepal_length%22%3A5.1%2C%22sepal_width%22%3A3.5%2C%22
          petal_length%22%3A1.4%2C%22petal_width%22%3A0.2%7D HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 19:00:00 EST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: text/plain
   Content-Length: 1361
   Date: Wed, 26 Mar 2014 20:13:34 GMT

Response Body

   [sepal_length] := 5.1 (DOUBLE)
   [sepal_width] := 3.5 (DOUBLE)
   [petal_length] := 1.4 (DOUBLE)
   [petal_width] := 0.2 (DOUBLE)

   [MiningSchema]
   [sepal_length] := 5.1 (DOUBLE)
   [sepal_width] := 3.5 (DOUBLE)
   [petal_length] := 1.4 (DOUBLE)
   [petal_width] := 0.2 (DOUBLE)

   [LocalTransformations]
   [derived_sepal_length] := 0.22222222222222213 (DOUBLE)
   [derived_sepal_width] := 0.6818181818181818 (DOUBLE)
   [derived_petal_length] := 0.07017543859649121 (DOUBLE)
   [derived_petal_width] := 0.04166666666666667 (DOUBLE)

   [BackPropagationNetwork]
   Value of neural input [3] is [0.042].
   Value of neural input [2] is [0.07].
   Value of neural input [1] is [0.682].
   Value of neural input [0] is [0.222].
   Value of output neuron [11] in the last neural layer is [1].
   Value of output neuron [12] in the last neural layer is [0].
   Value of output neuron [13] in the last neural layer is [0].

   [Output]
   The [predictedValue] is [Iris-setosa (STRING)]
   [class] := Iris-setosa (STRING)
   The [probability] of [Iris-setosa (STRING)] is [0.9995535104664939 (DOUBLE)]
   [Probability_setosa] := 0.9995535104664939 (DOUBLE)
   The [probability] of [Iris-versicolor (STRING)] is [4.464895332525406E-4 (DOUBLE)]
   [Probability_versicolor] := 4.464895332525406E-4 (DOUBLE)
   The [probability] of [Iris-virginica (STRING)] is [2.536692637033178E-13 (DOUBLE)]
   [Probability_virginica] := 2.536692637033178E-13 (DOUBLE)
```

5.2.3.3. Apply Model to Multiple Records or Apply Model to Single Binary Data

Definition	**POST /apply/{model_name}** This provides two kinds of operations. Generally, if a predictive model without `binary` type input is applied, this will be a batch 'apply' operation that streams multiple input records to Zementis Server. Zementis Server will automatically detect `Comma Separated Value` (CSV) or JSON Records formatted input and stream results back in the same format unless otherwise specified in the `Accept` request header parameter with `text/csv` or `application/json` values. Compressing input data with `zip` or `gzip` will result in the same compression method for the returned output stream. If a predictive model with a `binary` type input is applied, this will be a single 'apply' operation that processes a single binary source as input to Zementis Server.
Request Parameters	**`Content-Type (string):`** required header parameter with two accepted values: `application/octet-stream` or `multipart/form-data` **`model_name (string):`** required path variable for the name of the model to be applied **`maxThreads:`** optional query parameter for specifying the maximum number of concurrent threads (default value is twice the number of processor cores). No impact if a predictive model with a `binary` type input was applied. **`maxRecordsPerThread:`** optional query parameter for specifying the maximum number of records processed by a thread in batch (default value is 5000). No impact if a predictive model with a `binary` type input was applied. **`Accept:`** optional header parameter for explicitly specifying `text/csv` or `application/json` output format **`User-Agent:`** optional header parameter for full duplex HTTP streaming data if set to `AdapaStreaming` followed by any characters or a string containing value `curl` . Default data handling mode is copy-forward where response is rendered only after full request has been read by the server.
Returns	Returns results as CSV or as Result object if successful, Errors otherwise.

Example 5.25. Zementis REST Apply Model to Multiple Records

```
Request
```

```
  curl -u adapa:adapa -k https://localhost/adapars/apply/Iris_NN -X POST -T Iris_NN.csv \
       -H "Content-Type:application/octet-stream"
  curl -u adapa:adapa -k https://localhost/adapars/apply/Iris_NN?maxThreads=8 -X POST -F
 file=@Iris_NN.csv

Request  Header

   POST /adapars/apply/Iris_NN?maxThreads=8 HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 10148
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=----------------------------6da946996e0d

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: text/csv
   Transfer-Encoding: chunked
   Date: Wed, 26 Mar 2014 20:19:23 GMT

Response Body

   class,Probability_setosa,Probability_versicolor,Probability_virginica
   Iris-setosa,0.9995535104664939,4.464895332525406E-4,2.536692637033178E-13
   Iris-setosa,0.9985890830740689,0.0014109169248845744,1.0465677336558733E-12
   Iris-setosa,0.9993451737365701,6.54826263018726E-4,4.111504068226951E-13
   ...
```

Example 5.26. Zementis REST Apply Model to Single Binary Record

```
Request

  curl -u adapa:adapa -k https://localhost/adapars/apply/Caffe_NN -X POST -H 'Accept:application/json' -
F file=@0.jpg

Request  Header

   POST /adapars/apply/Caffe_NN HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.43.0
   Host: localhost
   Accept: application/json
   Content-Length: 5319
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=------------------------6099e489fd2da819

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 403
   Date: Fri, 27 May 2016 21:39:07 GMT

Response Body

   {
     "model" : "Caffe_NN",
```

```
    "outputs" : [ {
      "p_7" : 0.009013318755324183,
      "p_8" : 0.011660178735845163,
      "p_9" : 0.040489440800734404,
      "p_0" : 0.7602463077131643,
      "class" : "0",
      "p_1" : 0.006724422031736871,
      "p_2" : 0.052489690530517254,
      "p_3" : 0.004134235496422808,
      "p_4" : 0.027965981244545225,
      "p_5" : 0.014539398304602753,
      "p_6" : 0.07273702638710705
  } ]
```

5.2.3.4. Asynchronously Apply Model to Multiple Records

Definition	**POST /apply/{model_name}/async** This is an asynchronous batch 'apply' operation that streams multiple input records from remote location specified in uploaded properties file and writes the result back to the remote data target. The properties file describes the remote data source and target locations, connection properties, and access credentials. Zementis Server will automatically detect `Comma Separated Value` (CSV) or JSON Records formatted input and streams the result back in CSV format. Compressing input data with `zip` or `gzip` will result in the same compression method for the result.
Request Parameters	**`Content-Type (string)`:** required header parameter with two accepted values: `application/octet-stream` or `multipart/form-data` **`model_name (string)`:** required path variable for the name of the model to be applied **`maxThreads`:** optional query parameter for specifying the maximum number of concurrent threads (default value is twice the number of processor cores). **`maxRecordsPerThread`:** optional query parameter for specifying the maximum number of records processed by a thread in batch (default value is 5000).
Returns	Returns status information, job ID and description, output handle, and start timestamp of processing job in JSON format.

Example 5.27. Zementis REST Asynchronously Apply Model to Multiple Records

```
Request

  curl -u adapa:adapa -k https://localhost/adapars/apply/Iris_NN/async -X POST -T Iris_NN_CSV.properties
 \
       -H "Content-Type:application/octet-stream"
  curl -u adapa:adapa -k https://localhost/adapars/apply/Iris_NN/async?maxThreads=8 -X POST \
```

```
       -F file=@Iris_NN_CSV.properties

Request  Header

   POST /adapars/apply/Iris_NN/async?maxThreads=8 HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.54.0
   Host: localhost
   Accept: */*
   Content-Length: 376
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=----------------------------c6e69656a61898e9

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Content-Type: application/json
   Date: Thu, 10 Aug 2017 16:24:42 GMT

Response Body

   {
     "status" : "STARTED",
     "id" : 4,
     "output" : "Iris_NN_output_4_20170810_092441.csv",
     "startTime" : "2017-08-10 09:24:41.595 -0700",
     "description" : "Amazon S3 Connector: bucket='myBucket', input='Iris_NN.csv'"
   }
```

5.2.4. Operations on Resources

5.2.4.1. List Available Resources

Definition	**GET /resources** This operation retrieves information on all available resource files uploaded on Zementis Server. Use file names as identifiers for all operations requiring a `file_name` path variable.
Request Parameters	None
Returns	Returns a Resources object if successful, an Errors object otherwise.

Example 5.28. Zementis REST List Resources

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resources

Request Header

   GET /adapars/resources HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
```

```
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 363
   Date: Mon, 24 Nov 2014 22:56:50 GMT

Response Body

   {
     "resources" : [ {
       "fileName" : "ECommerceFraud_NN.jar",
       "resourceType" : "Custom Functions",
       "resourceIdentifier" : "Function Namespace",
       "resourceNames" : [ "fraud" ]
     }, {
       "fileName" : "ECommerceFraud_NN.xls",
       "resourceType" : "Lookup Tables",
       "resourceIdentifier" : "Table Name",
       "resourceNames" : [ "StatePoints" ]
     } ]
   }
```

5.2.4.2. Get Resource Information

Definition	**GET /resource/{file_name}**
	Get information on the specified resource file.
Request Parameters	**`file_name (string)`:** required path variable for an existing resource file name
Returns	Returns a ResourceInfo object if successful, an Errors object otherwise.

Example 5.29. Zementis REST Get Resource Information

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resource/ECommerceFraud_NN.jar

Request Header

   GET /adapars/resources/ECommerceFraud_NN.jar HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
```

```
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 161
   Date: Mon, 24 Nov 2014 23:05:51 GMT

Response Body

   {
     "fileName" : "ECommerceFraud_NN.jar",
     "resourceType" : "Custom Functions",
     "resourceIdentifier" : "Function Namespace",
     "resourceNames" : [ "fraud" ]
   }
```

5.2.4.3. Get Resource File

Definition	**GET /resource/{file_name}/source**
	Download a resource file.
Request Parameters	**`file_name (string)`:** required path variable for an existing resource file name
Returns	Returns a copy of the resource file if successful, an Errors object otherwise.

Example 5.30. Zementis REST Get Resource File

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resource/ECommerceFraud_NN.jar/source

Request Header

   GET /adapars/resources/ECommerceFraud_NN.jar/source HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/octet-stream
   Content-Length: 1675
   Date: Mon, 24 Nov 2014 23:15:35 GMT

Response Body

   PK
   ??uE META-INF/??PK
   ??uE?'!
   K-*??#R0?3??r?Cq,HL?HU?%-?x???RKRSt?*A???
```

```
...
```

5.2.4.4. Upload New Resource File

Definition	**POST /model** Upload a new resource file. The file name in 'file' body parameter will be used to identify this resource.
Request Parameters	**`Content-Type (string):`** required header parameter with two accepted values: `application/octet-stream` or `multipart/form-data` **`file (string):`** required query parameter for PMML file name, if Content-Type is `application/octet-stream`, or a body parameter in `multipart/form-data` content encoding **`Content-Type (string):`** required body parameter for resource a file name, and its content
Returns	Returns ResourceInfo object, `201` HTTP response status code, and response header entry `Location` with URI of created resource if upload was successful, an Errors object otherwise.

Example 5.31. Zementis REST Upload New Resource File with POST

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resource?file=ECommerceFraud_NN.xls -X POST \
        -T ECommerceFraud_NN.xls -H "Content-Type:application/octet-stream"
   curl -u adapa:adapa -k https://localhost/adapars/resource -X POST -F file=@ECommerceFraud_NN.xls

Request  Header

   POST /adapars/resource HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 30933
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=----------------------------d9c9597fd160

Response Header

   HTTP/1.1 201 Created
   Server: Apache-Coyote/1.1
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Location: http://localhost:8080/adapars/resource/ECommerceFraud_NN.xls
   Content-Type: application/json
   Content-Length: 156
   Date: Wed, 26 Mar 2014 19:45:18 GMT

Response Body
```

```
   {
     "fileName" : "ECommerceFraud_NN.xls",
     "resourceType" : "Lookup Tables",
     "resourceIdentifier" : "Table Name",
     "resourceNames" : [ "StatePoints" ]
   }
```

5.2.4.5. Upload New Resource File with PUT

Definition	**PUT /model** Upload a new resource file. The file name in 'file' query parameter will be used to identify this resource.
Request Parameters	**`file (string):`** required query parameter for resource file name
Returns	Returns a ResourceInfo object, `201` HTTP response status code, and a response header entry `Location` with URI of the created resource if the upload was successful, an Errors object otherwise.

Example 5.32. Zementis REST Upload New Resource File with PUT

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resource?file=ECommerceFraud_NN.xls -X PUT -T
 ECommerceFraud_NN.xls

Request  Header

   PUT /adapars/resource?file=ECommerceFraud_NN.xls HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 30720
   Expect: 100-continue

Response Header

   HTTP/1.1 201 Created
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Location: http://localhost:8080/adapars/resource/ECommerceFraud_NN.xls
   Content-Type: application/json
   Content-Length: 156
   Date: Mon, 24 Nov 2014 23:37:26 GMT

Response Body

   {
```

```
    "fileName" : "ECommerceFraud_NN.xls",
    "resourceType" : "Lookup Tables",
    "resourceIdentifier" : "Table Name",
    "resourceNames" : [ "StatePoints" ]
  }
```

5.2.4.6. Remove Resource File

Definition	**DELETE /resource/{file_name}** Remove the specified resource file and list all remaining resources.
Request Parameters	**`file_name (string):`** required path variable for existing resource file name
Returns	Returns a Resources object with a list of all remaining resource files if successful, an Errors object otherwise.

Example 5.33. Zementis REST Remove Resource File

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resource/ECommerceFraud_NN.jar -X DELETE

Request Header

   DELETE /adapars/resource/ECommerceFraud_NN.xls HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 195
   Date: Mon, 24 Nov 2014 23:50:13 GMT

Response Body

   {
     "resources" : [ {
       "fileName" : "ECommerceFraud_NN.jar",
       "resourceType" : "Custom Functions",
       "resourceIdentifier" : "Function Namespace",
       "resourceNames" : [ "fraud" ]
     } ]
   }
```

5.2.4.7. Remove All Resource Files

Definition	**DELETE /resources** Remove all available resources and list the remaining resources.
Request Parameters	None
Returns	Returns a Resources object with an empty `resources` array if successful, an Errors object otherwise.

Example 5.34. Zementis REST Remove All Resource Files

```
Request

   curl -u adapa:adapa -k https://localhost/adapars/resources -X DELETE

Request Header

   DELETE /adapars/resources HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   Pragma: No-cache
   Cache-Control: no-cache
   Expires: Wed, 31 Dec 1969 16:00:00 PST
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Content-Type: application/json
   Content-Length: 23
   Date: Mon, 24 Nov 2014 23:57:57 GMT

Response Body

   {
     "resources" : [ ]
   }
```

5.2.5. Operations on License

5.2.5.1. Get License Properties

Definition	**GET /license** This operation retrieves properties of license on Zementis Server.
Request Parameters	None

Returns	Returns license properties if successful, an Errors object otherwise.

Example 5.35. Zementis REST Get License

```
Request

   curl -X GET "http://localhost:8080/adapars/licenses" -H "accept: application/json"

Request Header

   GET /adapars/license HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*

Response Header

   cache-control: private
   content-length: 204
   content-length: 583
   content-type: application/json
   date: Sun, 27 Jan 2019 23:24:35 GMT

Response Body

{
  "properties" : {
    "Company" : "Zementis, Inc.",
    "Email" : "support@zementis.com",
    "Expires On" : "Jan 6, 2116 16:24 PST",
    "Product" : "ADAPA",
    "Edition" : "ADAPA Enterprise Server",
    "Number of Cores" : "128 (system reports 4 cores)",
    "Name" : "Engineering Team"
  },
  "status" : "VALID",
  "message" : "The license is VALID."
}
```

5.2.5.2. Upload a new license file

Definition	**POST /license**
	Upload a new license file.
Request Parameters	**`file_name (string):`** required path variable for an existing license file name
Returns	Returns license properties if successful, an Errors object otherwise.

Example 5.36. Zementis REST Post License

```
Request

   curl -X GET "http://localhost:8080/adapars/licenses" -H "accept: application/json"

Request Header
```

```
   POST /adapars/license HTTP/1.1
   Authorization: Basic YWRhcGE6YWRhcGE=
   User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
   Host: localhost
   Accept: */*
   Content-Length: 30933
   Expect: 100-continue
   Content-Type: multipart/form-data; boundary=----------------------------d9c9597fd160

Response Header

   HTTP/1.1 201 Created
   Server: Apache-Coyote/1.1
   X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
   Location: http://localhost:8080/adapars/resource/ECommerceFraud_NN.xls
   content-length: 583
   content-type: application/json
   date: Sun, 27 Jan 2019 23:24:35 GMT

Response Body

 {
  "properties" : {
    "Company" : "Zementis, Inc.",
    "Email" : "support@zementis.com",
    "Expires On" : "Jan 6, 2116 16:24 PST",
    "Product" : "ADAPA",
    "Edition" : "ADAPA Enterprise Server",
    "Number of Cores" : "128 (system reports 4 cores)",
    "Name" : "Engineering Team"
  },
  "status" : "VALID",
  "message" : "The license is VALID."
}
```