



Zementis Predictive Analytics Solutions Guide

10.11.0.1

Zementis Predictive Analytics

Solutions Guide

Software AG

Copyright © 2004 - 2016 Zementis Inc.

Copyright © 2016 - 2021 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

This document applies to Zementis Server 10.11.0.1 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.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Table of Contents

1. Introduction	1
1.1. Decision Solutions Overview	1
2. Predictive Models	4
2.1. Predictive Models using PMML	5
2.1.1. Predictive Model Building Process	8
2.1.2. Deploy and Test Predictive Models	11
2.1.2.1. Deploying Models	11
2.1.2.2. Testing Models	12
2.1.3. Data Scoring and Classification	13
2.1.4. Model Metrics	13
2.2. Predictive Models using ONNX	15
2.2.1. Predictive Model Building Process	15
2.2.2. Deploy ONNX Models	16
2.2.3. Data Scoring	17
3. Custom Resources	18
3.1. Custom Resources for PMML	18
3.1.1. Custom PMML Functions	18
3.1.1.1. Create Custom PMML Functions	18
3.1.1.2. Use Custom PMML Functions	20
3.1.1.3. Non-Deterministic Functions	21
3.1.2. External Lookup Tables	21
3.1.2.1. Create Lookup Tables in Excel	22
3.1.2.2. Use Lookup Tables in PMML	24
3.1.3. External Training Data Tables	25
3.2. Custom Resources for ONNX	25
3.2.1. Create a Pre-processor	26
3.2.2. Create a Post-processor	26
3.3. Deploy Resources	27
3.3.1. Deleting Resources	28
4. Pipelines for ONNX	30
4.1. Defining Pipeline File	30
4.2. Deploy Pipeline File	30
5. Extensions API	32
5.1. Using the Zementis Server Extensions API	32

5.1.1. Custom Function	32
5.1.2. Lookup Table	33
5.1.3. Asset Repository	34
5.1.4. Logging Store	35
5.2. Overview of code examples	36
5.3. Deployment of Zementis Server Extensions	37
6. REST API	38
6.1. General Notes	38
6.1.1. URI	38
6.1.2. Request	39
6.1.3. Response	40
6.1.4. Errors	40
6.1.5. Authorization	42
6.2. API	46
6.2.1. JSON Objects	46
6.2.2. Operations on PMML Models	58
6.2.2.1. List Available PMML Models	58
6.2.2.2. Get PMML Model Information	59
6.2.2.3. Get PMML Model Source	60
6.2.2.4. Get PMML Model Serialized Source	61
6.2.2.5. Get PMML Model Metrics Information	61
6.2.2.6. Upload New PMML Model	62
6.2.2.7. Activate an existing PMML Model	64
6.2.2.8. Deactivate an existing PMML Model	65
6.2.2.9. Remove PMML Model	66
6.2.2.10. Remove All PMML Models	66
6.2.3. Apply PMML model	67
6.2.3.1. Apply PMML Model to Single Record	67
6.2.3.2. Apply PMML Model to Single Record and Explain Result	68
6.2.3.3. Apply PMML Model to Multiple Records	70
6.2.3.4. Asynchronously Apply PMML Model to Multiple Records	71
6.2.4. Operations on PMML Resources	72
6.2.4.1. List Available PMML Resources	72
6.2.4.2. Get PMML Resource Information	73
6.2.4.3. Get PMML Resource File	74
6.2.4.4. Upload New PMML Resource File	75
6.2.4.5. Remove PMML Resource File	76

6.2.4.6. Remove All PMML Resource Files	77
6.2.5. Operations on ONNX Models	78
6.2.5.1. List Available ONNX Models	78
6.2.5.2. Get ONNX Model Information	79
6.2.5.3. Get ONNX Model Source	80
6.2.5.4. Upload New ONNX Model	80
6.2.5.5. Activate an existing ONNX Model	81
6.2.5.6. Deactivate an existing ONNX Model	82
6.2.5.7. Remove ONNX Model	83
6.2.5.8. Remove All ONNX Models	84
6.2.6. Apply ONNX model	84
6.2.6.1. Apply ONNX Model to Data File	84
6.2.6.2. Apply ONNX Pipeline to Data File	86
6.2.7. Operations on ONNX Resources	87
6.2.7.1. List Available ONNX Resources	87
6.2.7.2. Get ONNX Resource Information	88
6.2.7.3. Get ONNX Resource File	88
6.2.7.4. Upload New ONNX Resource File	89
6.2.7.5. Remove ONNX Resource File	90
6.2.7.6. Remove All ONNX Resource Files	91
6.2.8. Operations on ONNX Pipelines	92
6.2.8.1. List Available ONNX Pipelines	92
6.2.8.2. Get ONNX Pipeline Information	93
6.2.8.3. Create New ONNX Pipeline	94
6.2.8.4. Remove ONNX Pipeline	95
6.2.8.5. Remove All ONNX Pipelines	95
6.2.9. Operations on License	96
6.2.9.1. Get License Properties	96
6.2.9.2. Upload a new license file	97

List of Figures

1.1. Decision Solution	2
2.1. Interoperability through Open Standards	5
2.2. Model Ensemble	7
2.3. Predictive Modeling Process	8
2.4. Predictive Models in the Zementis Server UI	12
2.5. Memory Metrics and Prediction Metrics of Classification Model	14
2.6. Memory Metrics and Prediction Metrics of Regression Model	15
2.7. Predictive Models in the Zementis Server UI	17
3.1. Custom PMML Function Example	19
3.2. Example Using a Custom Function in PMML	20
3.3. Custom PMML Function Example	21
3.4. Lookup Table Example	22
3.5. Sample Excel Lookup Table	23
3.6. A LookupTable with two inputs and one output	24
3.7. PMML Resource Files in the Zementis Server UI	27
3.8. ONNX Resource Files in the Zementis Server UI	28
3.9. Resource dependency exception in the Zementis Server UI	29
4.1. Pipeline Workflow	30
4.2. ONNX Pipelines in the Zementis Server UI	31
5.1. Dependencies for Custom Functions	33
5.2. Dependencies for Lookup Table	34
5.3. Dependencies for Asset Repository	35
5.4. Dependencies for Logging Repository	36
6.1. Interactive REST API Documentation	39

List of Tables

2.1. Sample Predictive Models	9
2.2. Directory Structure of Sample Models	10
2.3. Directory Structure of Sample Files	16
3.1. PMML and Java types in Zementis Server	19
3.2. API definition for pre-processor	26
3.3. API definition for post-processor	26
5.1. Directory structure of code examples	36
6.1. Typical Zementis REST Error Responses	41
6.2. Zementis REST Permissions	43

List of Examples

6.1. Zementis REST Error Response	40
6.2. Zementis REST Errors Object	47
6.3. Zementis REST PMML Models Object	47
6.4. Zementis REST PMML ModelInfo Object	48
6.5. Zementis REST PMML Field Object	49
6.6. Zementis REST PMML MetricsInfo Object	49
6.7. Zementis REST PMML Record Object	50
6.8. Zementis REST PMML Records Object	50
6.9. Zementis REST PMML Result Object	51
6.10. Zementis REST ONNX Result Object	52
6.11. Zementis REST PMML ResourceInfo Object	52
6.12. Zementis REST PMML Resources Object	53
6.13. Zementis REST ONNX Records Object	53
6.14. Zementis REST ONNX Model Information List	54
6.15. Zementis REST ONNX ModelInfo Object	55
6.16. Zementis REST ONNX Field Object	55
6.17. Zementis REST ONNX Resources Object	56
6.18. Zementis REST ONNX ResourceInfo Object	56
6.19. Zementis REST ONNX Processor Information	57
6.20. Zementis REST ONNX Pipelines Object	57
6.21. Zementis REST ONNX PipelineInfo Object	58
6.22. Zementis REST List PMML Models	58
6.23. Zementis REST Get PMML Model Information	59
6.24. Zementis REST Get PMML Model Source	60
6.25. Zementis REST Get PMML Model Serialized	61
6.26. Zementis REST Get PMML Model Metrics	62
6.27. Zementis REST Upload New PMML Model	63
6.28. Zementis REST Activate an existing PMML Model	64
6.29. Zementis REST Deactivate an existing PMML Model	65
6.30. Zementis REST Remove PMML Model	66
6.31. Zementis REST Remove All PMML Models	67
6.32. Zementis REST Apply PMML Model to Single Record	68
6.33. Zementis REST Apply PMML Model to Single Record and Explain Result	69
6.34. Zementis REST Apply PMML Model to Multiple Records	70

6.35. Zementis REST Asynchronously Apply PMML Model to Multiple Records	72
6.36. Zementis REST List PMML Resources	73
6.37. Zementis REST Get PMML Resource Information	73
6.38. Zementis REST Get PMML Resource File	74
6.39. Zementis REST Upload New PMML Resource File	75
6.40. Zementis REST Remove PMML Resource File	76
6.41. Zementis REST Remove All PMML Resource Files	77
6.42. Zementis REST List ONNX Models	78
6.43. Zementis REST Get ONNX Model Information	79
6.44. Zementis REST Get ONNX Model Source	80
6.45. Zementis REST Upload New Model	81
6.46. Zementis REST Activate an existing ONNX Model	82
6.47. Zementis REST Deactivate an existing ONNX Model	82
6.48. Zementis REST Remove ONNX Model	83
6.49. Zementis REST Remove All ONNX Models	84
6.50. Zementis REST Apply ONNX Model to Data File	85
6.51. Zementis REST Apply ONNX Pipeline to Data File	86
6.52. Zementis REST List ONNX Resources	87
6.53. Zementis REST Get ONNX Resource Information	88
6.54. Zementis REST Get ONNX Resource File	89
6.55. Zementis REST Upload New ONNX Resource File	89
6.56. Zementis REST Remove ONNX Resource File	90
6.57. Zementis REST Remove All ONNX Resource Files	91
6.58. Zementis REST List ONNX Pipelines	92
6.59. Zementis REST Get ONNX Pipeline Information	93
6.60. Zementis REST Create New ONNX Pipeline	94
6.61. Zementis REST Remove ONNX Pipeline	95
6.62. Zementis REST Remove All ONNX Pipelines	96
6.63. Zementis REST Get License	96
6.64. Zementis REST Post License	97

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) and Open Neural Network Exchange (ONNX) standards. PMML and ONNX handle 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 or ONNX.

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

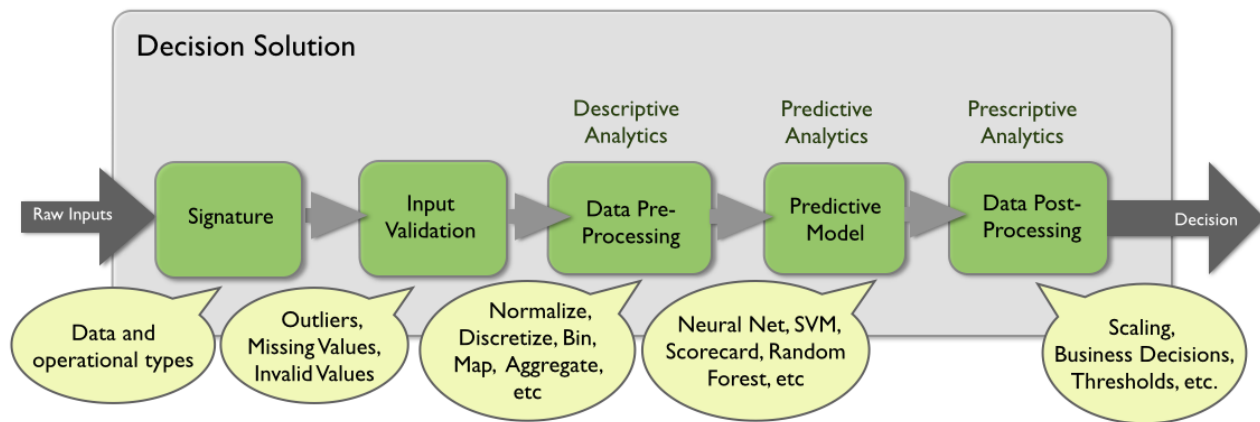
Zementis Server allows data-driven insight and expert knowledge to be combined into a single and powerful decision strategy through the use of PMML or ONNX. Whereas *expert knowledge* encapsulates the logic used by experts to solve problems, *data-driven knowledge* is based on the ability to automatically 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 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 and ONNX, 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, ONNX 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, PMML, and ONNX, both the human resource and technology requirement to deploy these powerful analytics is drastically reduced. At the same time, the pace of deployment is increased and model integrity and quality are 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 5](#) gives an overview of the Zementis Server Java Extensions API. Finally, an extensive description of the web service capabilities is provided in [Chapter 6](#).

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, PMML and ONNX provide a standards-based methodology and process through which value from Big Data initiatives can be gained, quantified and demonstrated.

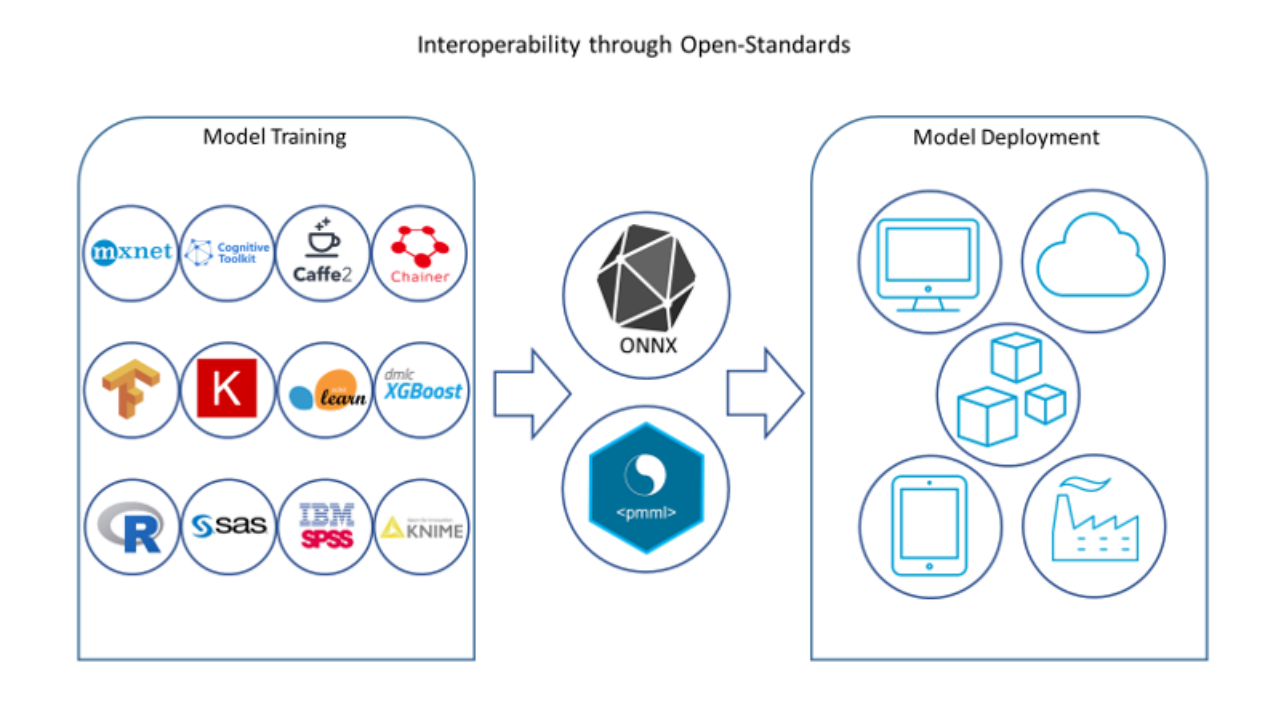
The predictive model building process begins by working with and developing a deep understanding of historical data which is mined for feature detectors. These are in turn used to build the predictive models. While a time-consuming and laborious process, this provides the foundation for creating value 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 or ONNX. 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. ONNX is a format for representing machine learning models, with a focus on Deep Learning models. ONNX defines a common set of operators – the building blocks of machine learning and deep learning models – and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers. Enabling interoperability between different frameworks and streamlining the path from research to production helps increase the speed of innovation in the AI community. Open-standards-based representation of machine learning models ensures interoperability between heterogenous model development and deployment environments, thereby shortening the model deployment cycle time.

Figure 2.1. Interoperability through Open Standards



2.1. Predictive Models using PMML

PMML is a very mature standard. Its latest version, PMML 4.4.1, was released in June 2020. 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, 4.3 and 4.4) into version 4.4.1. 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 4.4.1 incorporates many new elements into the standard, including elements for representing 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](#) website 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:

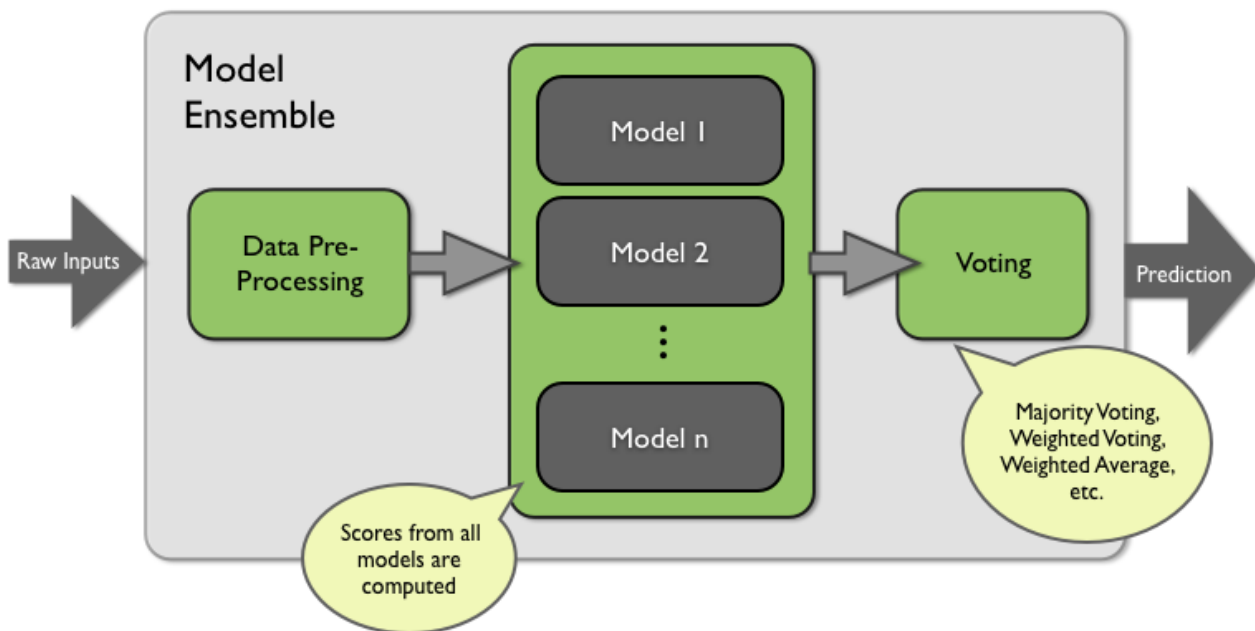
- Anomaly Detection Models (Isolation Forest and One-Class SVM)
- Association Rules Models (Rectangular or Transactional format)
- Clustering Models (Distribution-Based, Center-Based, and 2-Step Clustering)
- Decision Trees (for classification and regression) together with multiple missing value handling strategies (Default Child, Last Prediction, Null Prediction, Weighted Confidence, Aggregate Nodes)
- 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)
- 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 2.2](#), 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 2.2. Model Ensemble

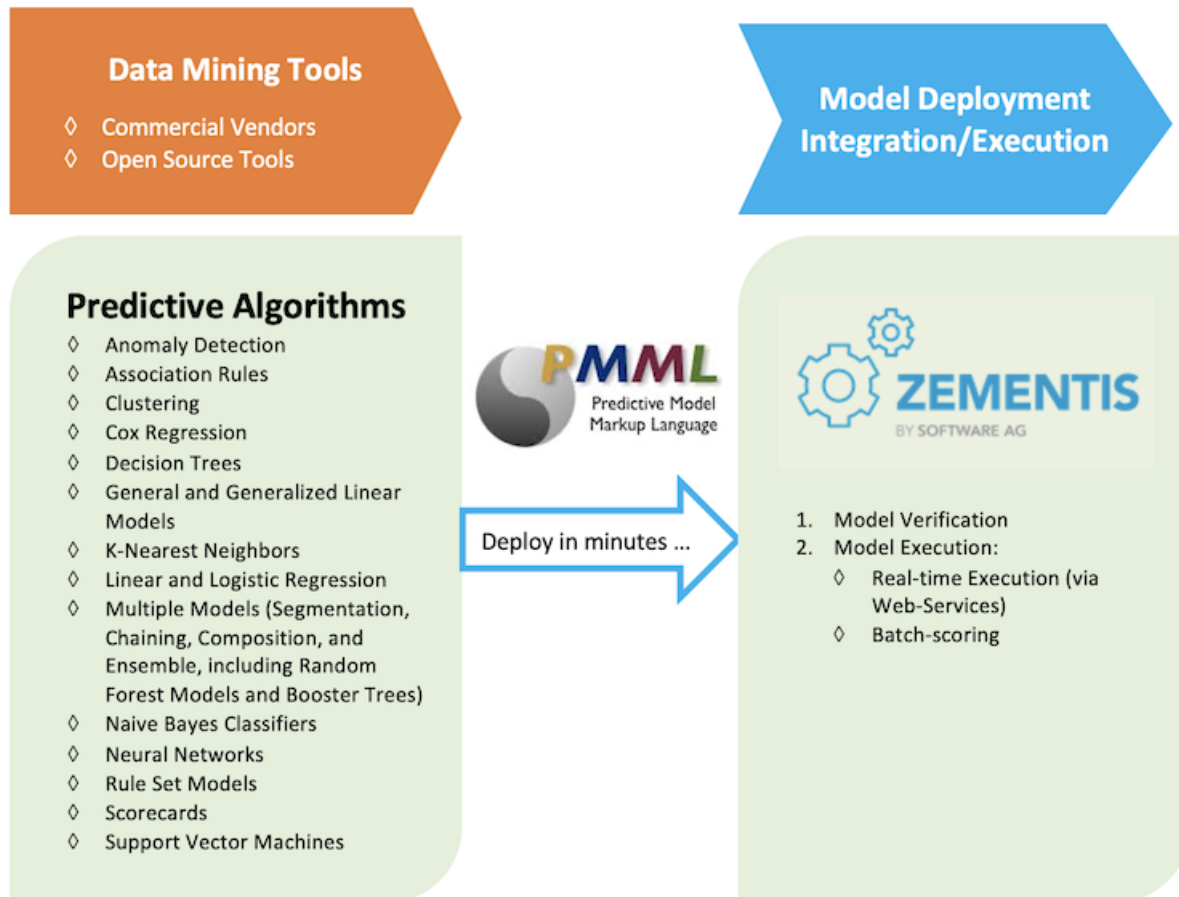


PMML is capable of representing not only a model ensemble but also model composition, segmentation, chaining and cascade. The same is true for Zementis Server, which consumes and executes PMML files containing multiple models. With Zementis Server and PMML, after a model is built, export it as (or convert 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 to 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 2.3](#). In the next sections, we elaborate on each phase of this process in more detail.

Figure 2.3. Predictive Modeling Process



2.1.1. 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 are 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 \(DMG\)](#) website.

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 6](#).

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.pmm1	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.pmm1	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.pmm1	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.pmm1	A predictive model composed of a random forest model used to predict diabetes progression for a group of patients. This predictive model is an ex-

File Name	Description
	ample 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.
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 Server UI 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

Directory	Contents
	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.

2.1.2. 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 Server UI.

2.1.2.1. Deploying Models

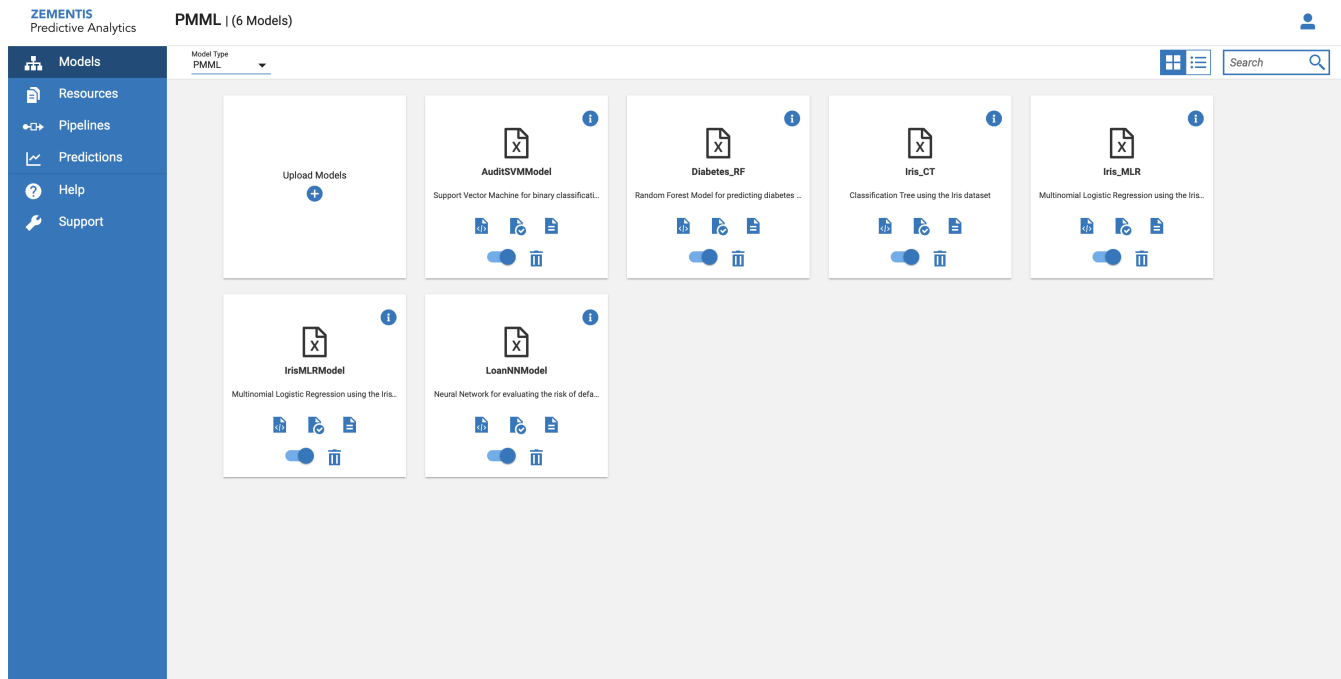
Models are deployed in Zementis Server by uploading them directly in the Zementis Server UI. Although a data mining tool may export an older version of PMML, Zementis Server will automatically perform comprehensive syntactic and semantic checks, correct known issues and convert your PMML file to version 4.4.1 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 potential issues with a PMML file may need to be reviewed by downloading the file in the Upload Model dialog box. 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 blue checkmark 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 Server UI Predictive Models page. [Figure 2.4](#) shows the Zementis Server UI after uploading the sample predictive models described in [Table 2.1](#).

Figure 2.4. Predictive Models in the Zementis Server UI



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

2.1.2.2. 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 Server UI 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 troubleshooting.

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 Server UI 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 \(DMG\)](#) website 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.

2.1.3. Data Scoring and Classification

Bulk scoring in batch mode can be easily performed through the Zementis Server UI, 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 6](#).

2.1.4. Model Metrics

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

Clicking on the model name from the Zementis Server UI 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 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 their 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 including 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 2.5. Memory Metrics and Prediction Metrics of Classification Model

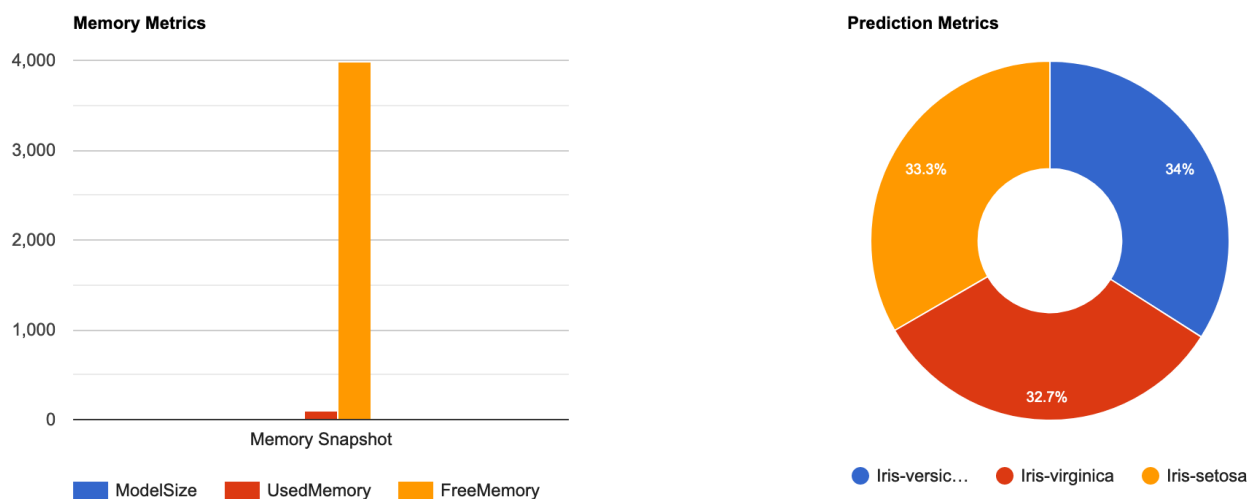
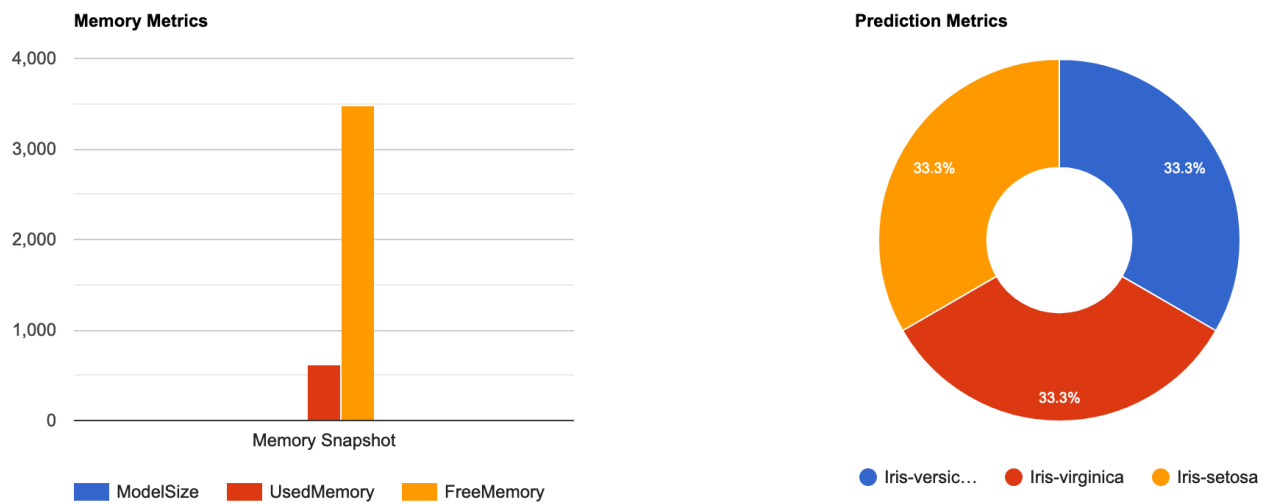


Figure 2.6. Memory Metrics and Prediction Metrics of Regression Model



2.2. Predictive Models using ONNX

ONNX file is a Protobuf encoded tensor graph built from ONNX operations which are versioned as operation sets. The operations include most of the typical deep learning primitives, linear operations, convolutions and activation functions. Through these operation sets, ONNX tackles the problem of interoperability between training and deployment environments by providing the guarantee that a model will always remain deterministic in how it behaves when presented with a given data set, with an outcome invariant from the runtime environment. ONNX is a community project and has received significant backing from Microsoft, Facebook, Nvidia, Amazon and other organizations that see AI as a core focus of their business.

2.2.1. 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 are usually performed within the scientist's desktop through the use of an array of tools and scripts. Once the model is ready, it can be exported to ONNX using an appropriate library.

Exporters exist for many popular deep learning and machine learning libraries. For example, in the Python ecosystem, [tf2onnx](#) exports models created with [TensorFlow](#) and [Keras](#), and [sklearn-onnx](#) exports models created with [scikit-learn](#).

All sample files are available to download from the Zementis Server UI 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 that contain the sample files. [Table 2.3](#) describes how the sample files are organized.

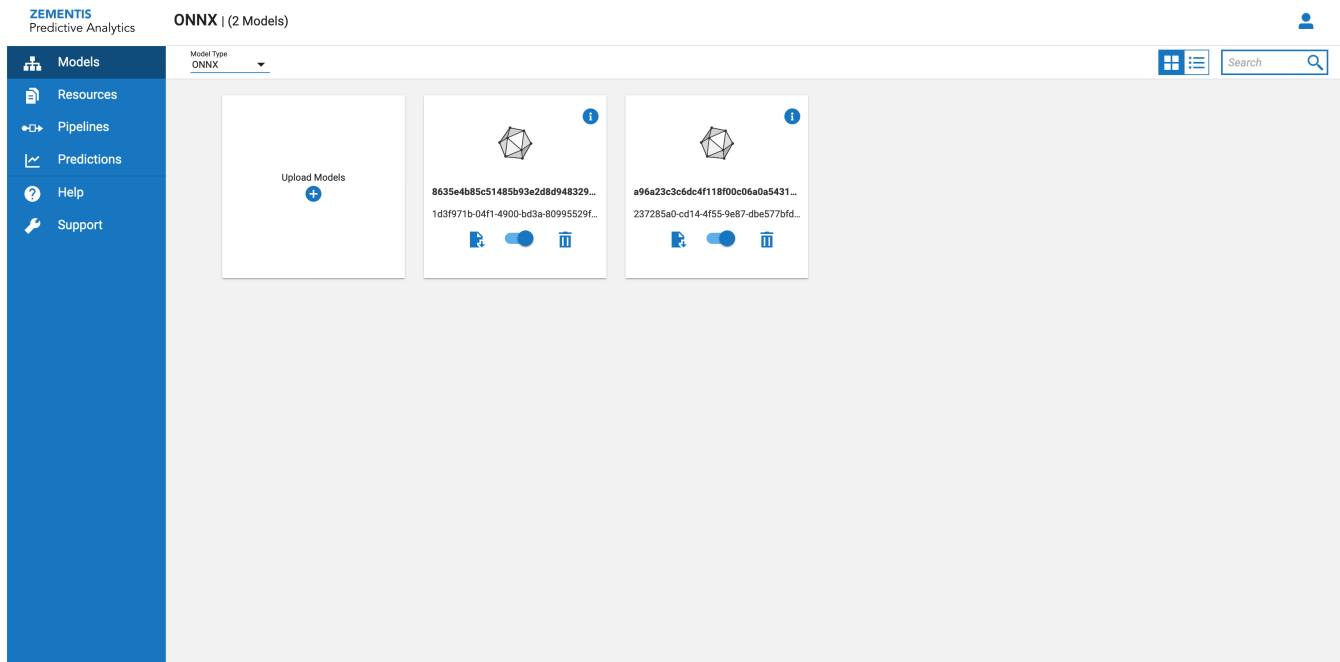
Table 2.3. Directory Structure of Sample Files

Directory	Contents
<code>rest-client</code>	Source and build files for sample Java client to Zementis REST API.
<code>rest-client/onnx</code>	Predictive model (ONNX) file: contains the "mobilenet.onnx" file.
<code>rest-client/onnx-resources</code>	Resource file: contains custom resource "mobilenet.jar". for model "mobilenet.onnx". Upload the resource files in Zementis Server before uploading the ONNX model file.
<code>rest-client/onnx-pipelines</code>	Pipeline definition file: "mobilenet.json". A pipeline represents an end-to-end workflow that encapsulates any pre-processing or post-processing steps that need to be performed in addition to invoking a machine learning model.
<code>rest-client/onnx-data</code>	Scored data files: contains the scored data files in PNG. Score a data file in Zementis Server against its respective model in order to perform the prediction. Each data file is named according to its respective ONNX file.

2.2.2. Deploy ONNX Models

Models are deployed in Zementis Server by uploading them directly in the Zementis Server UI. During upload, the blue check mark indicates that the ONNX file was uploaded without any errors. For more information on how to upload your models through the Zementis Server UI, see the Help page.

Figure 2.7. Predictive Models in the Zementis Server UI



2.2.3. Data Scoring

Data scoring can be performed through the Zementis Server UI. First, select the target model or pipeline and then upload a data file. Zementis Server will process the uploaded file and return a scored file.

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.

Note

For more information on how to manage your models through the Zementis Server UI, see the Help page.

Chapter 3. Custom Resources

Predictive models may require external resources. Files containing such resources can be uploaded to Zementis Server using the Zementis Server UI. Also, deleting a resource file will remove all the resources contained in it from Zementis Server.

3.1. Custom Resources for PMML

PMML models may require external resources such as custom functions, look-up tables or training data tables. Note, such resources should be uploaded before any models that depend on them. To delete a resource, the model that is dependent on the resource should be deleted first.

3.1.1. 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.

3.1.1.1. 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
float	float	java.lang.Float

PMML Data Type	Java Primitive Type	Java Class
integer	long	java.lang.Long
string		java.lang.String
time		org.joda.time.LocalDateTime

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.

3.1.1.2. 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 Server UI. To compile a class using the `@PMMLFunctions` annotation, include the `adapa-api-10.11.0.1.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>
```

3.1.1.3. 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 that provides the current time value for a specific 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));
    }
}
```

3.1.2. 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](#)

3.1.2.1. 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

<i>LookupTable</i>	<i>ChildObesity</i>					
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. Similar 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.

3.1.2.2. 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`, `childGroup`, and `childObesity`) correspond to what columns (`Age`, `Group`, and `Probability`) of the lookup table.

3.1.3. 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.

3.2. Custom Resources for ONNX

ONNX models may require external resources containing pre-processor and/or post-processor classes.

3.2.1. Create a Pre-processor

To create a pre-processor implement a class based on the API:
`com.zementis.adapa.onnx.api.PreProcessor<I>`

Table 3.2. API definition for pre-processor

Modifier and Type	Method and Description
<code>Class<I></code>	<code>getInputClass()</code> Return the Class information for the input type I processed by this Pre-Processor.
<code>Map<String, ai.onnxruntime.OnnxValue></code>	<code>processInput(I input)</code> Process the input and convert it to OnnxValue.

Put your customized pre-processing step in the method called “processInput”. So the pre-processed input will be ready for ONNX model inference. The return values in the method “processInput” are in key-value structure. The key indicates the model input field. The values should be in one of the final implemented classes extended from `ai.onnxruntime.OnnxValue`. Currently, `ai.onnxruntime.OnnxTensor` is the only datatype used. Regarding how to construct a correct `ai.onnxruntime.OnnxValue` or `ai.onnxruntime.OnnxTensor`, please use

```
OnnxTensor.createTensor(OrtEnvironment.getEnvironment(), yourData);
```

For more samples, please check the provided samples under release package.

3.2.2. Create a Post-processor

To create a post-processor implement a class based on the API:
`com.zementis.adapa.onnx.api.PostProcessor<O>`

Table 3.3. API definition for post-processor

Modifier and Type	Method and Description
<code>Class<O></code>	<code>getOutputClass()</code> Return the Class information for the output type O produced by this Post-Processor.
<code>O</code>	<code>processOutput(ai.onnxruntime.OrtSession.Result outputValue)</code> Process the onnxValue and convert it to desired output type O.

Put your customized post-processing step in the method called “processOutput”. So the post-processed output will perform the customized steps from the ONNX model inference result.

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 requires administrative privileges (`adapa-admin` role), it is highly recommended that users upload only trusted JARs.

3.3. Deploy Resources

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

If you have previously uploaded any resource files into Zementis Server, these are shown in the Zementis Server UI as a list. [Figure 3.7](#) shows the Zementis Server UI 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 3.7. PMML Resource Files in the Zementis Server UI

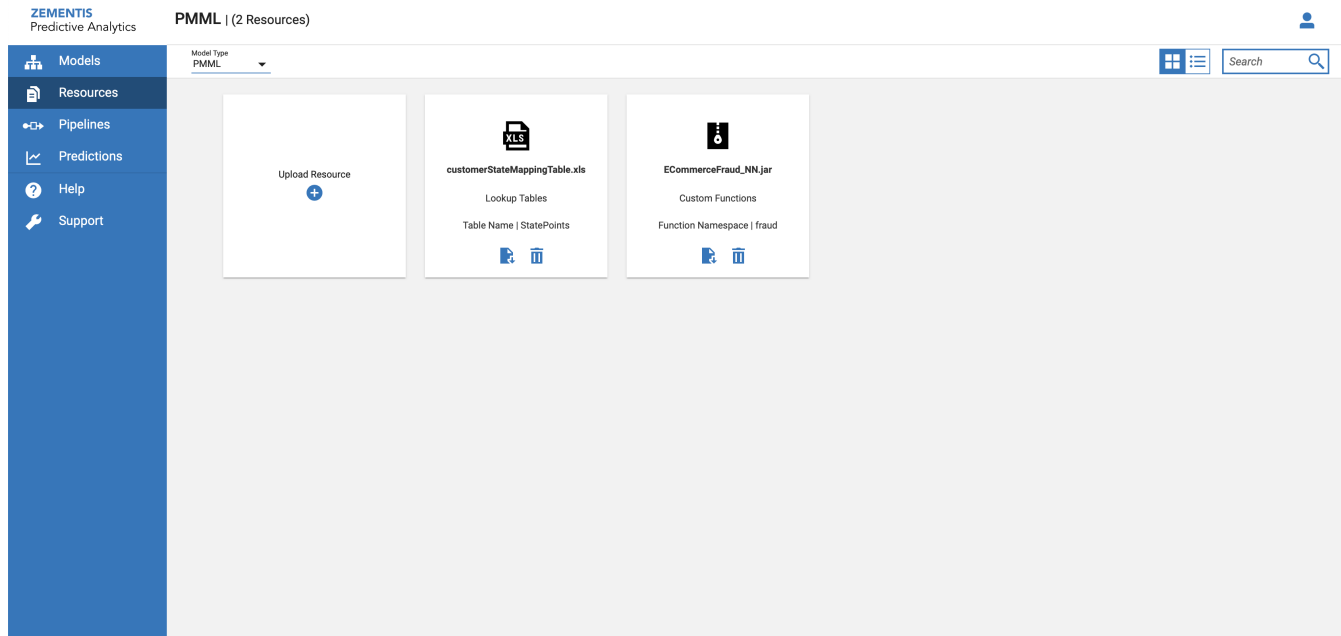
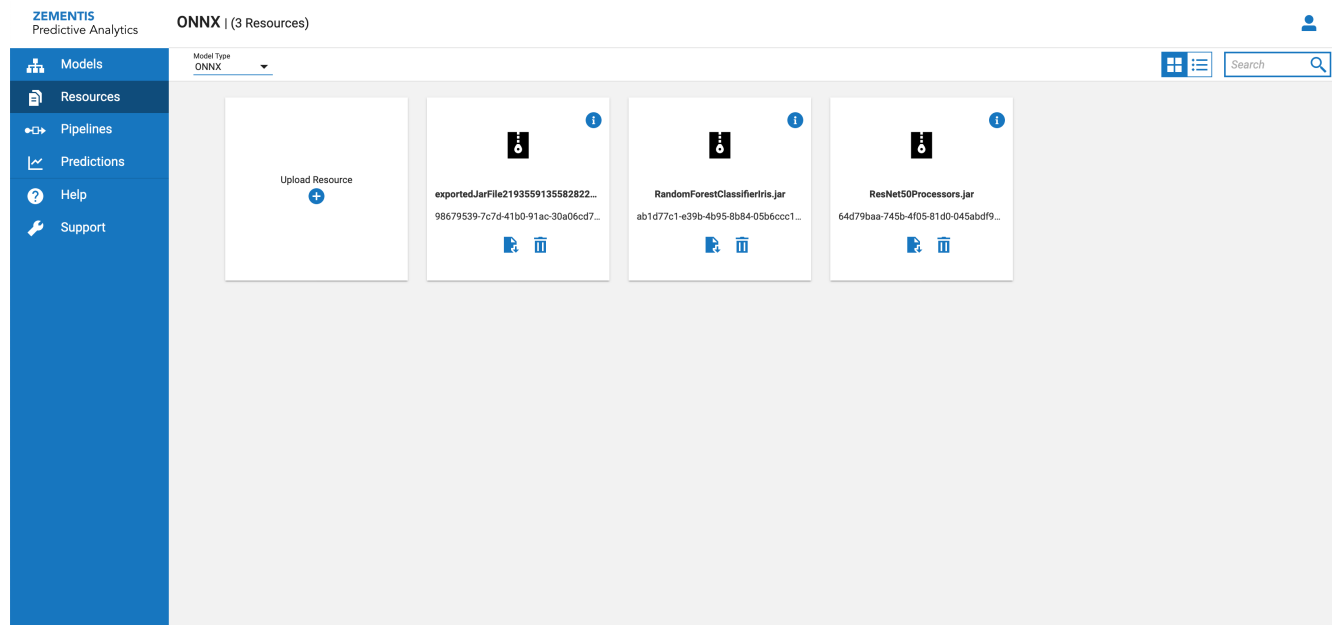


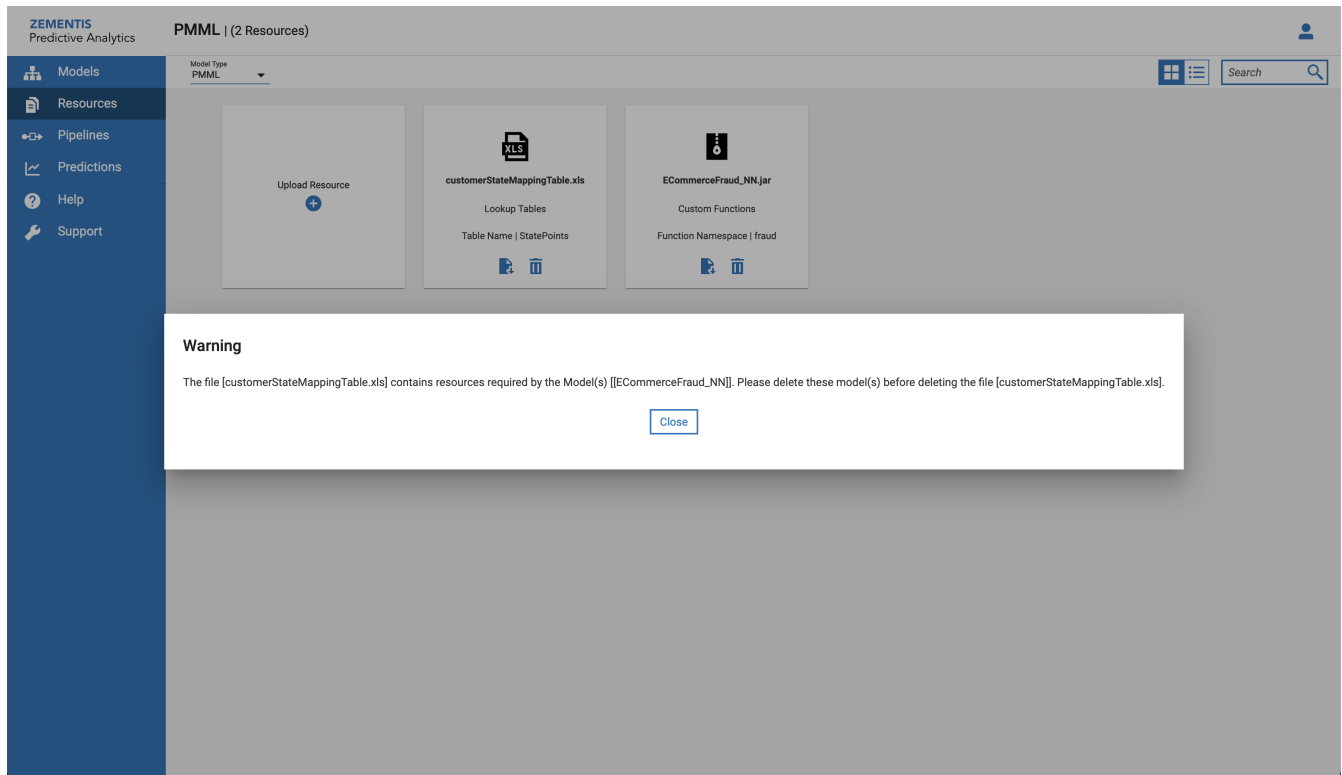
Figure 3.8. ONNX Resource Files in the Zementis Server UI



3.3.1. Deleting Resources

When deleting a resource file which is a downstream dependency of one of the models from the list, you must first delete the model and then delete the resource. [Figure 3.9](#) shows the Zementis Server UI when an exception is thrown.

Figure 3.9. Resource dependency exception in the Zementis Server UI



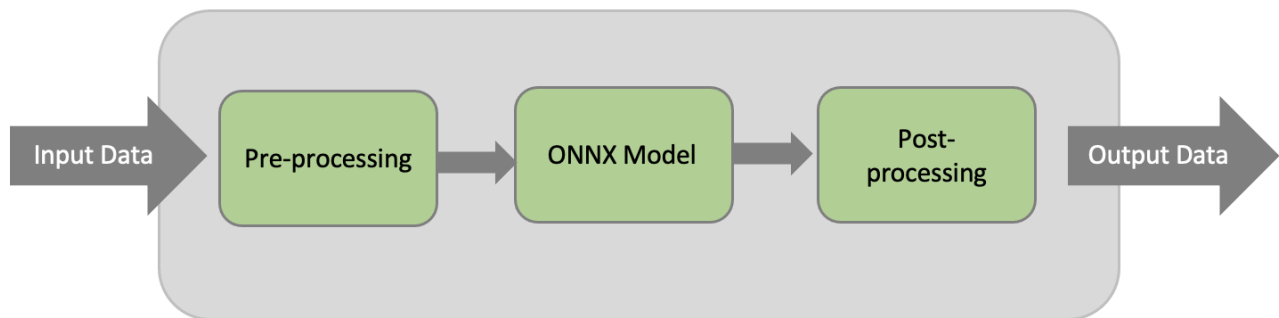
Note

For more information on how to manage your resources through the Zementis Server UI, see the Help page.

Chapter 4. Pipelines for ONNX

A pipeline represents an end-to-end workflow that encapsulates any pre-processing or post-processing steps that need to be performed in addition to invoking a machine learning model. If an ONNX model requires pre-processing or post-processing steps, the corresponding pre-processing/post-processing resources must be uploaded to the Zementis Server. A pipeline's end-to-end workflow can be depicted using the following sequence:

Figure 4.1. Pipeline Workflow



When a pre-processing step is part of a pipeline, the input data is first processed by the pre-processor. The pre-processed data is then passed onto the model and finally, the outputs from the model are processed by the post-processor. The output generated by the post-processor is returned as the final output.

4.1. Defining Pipeline File

A pipeline definition file is a JSON format file. This definition file must include a custom pipeline name, a machine learning model, and the inclusion of a pre-processor or post-processor is optional. A sample definition file content is as follows:

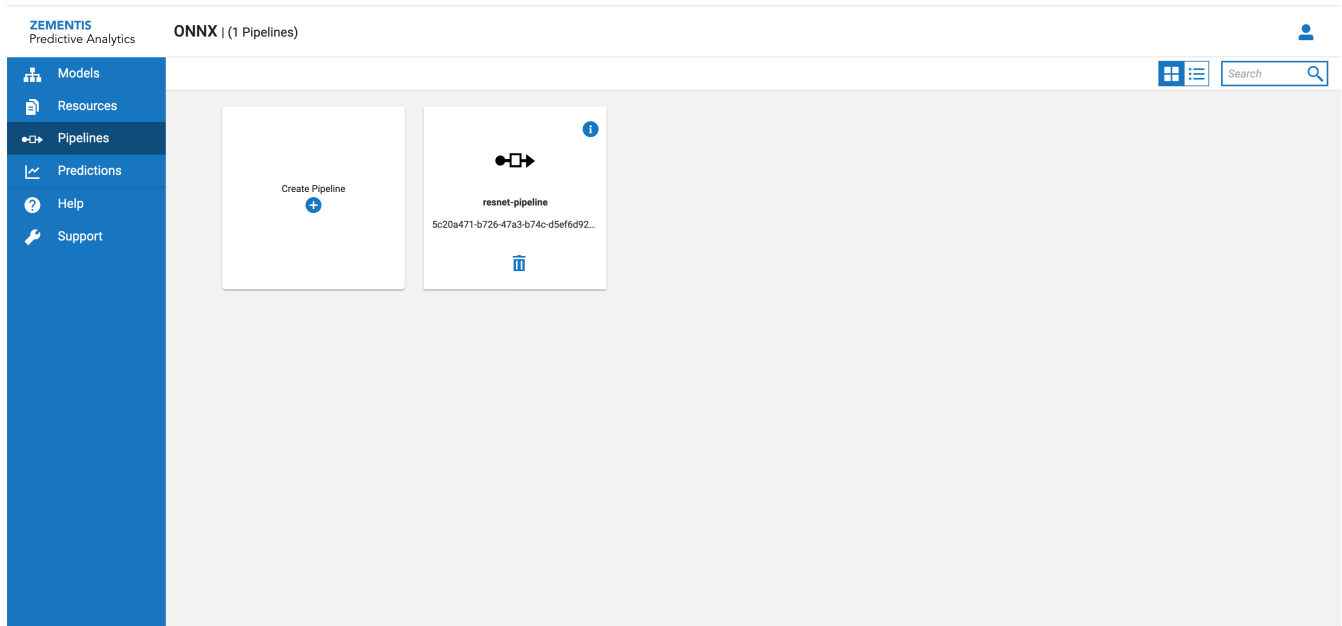
```

{
  "pipelineName": "mobilenet",
  "preProcessor": "pkg.Mobilenetv2Processors",
  "modelName": "main",
  "postProcessor": "pkg.Mobilenetv2Processors"
}
  
```

4.2. Deploy Pipeline File

Custom pipeline definition files can be deployed in Zementis Server by uploading them directly in Zementis Server UI.

Figure 4.2. ONNX Pipelines in the Zementis Server UI



Note

For more information on how to manage your pipelines through the Zementis Server UI, see the Help page.

Chapter 5. 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.

5.1. 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 5.2](#) will provide details about how the Extensions API and sample implementations are packaged with the `adapa-app-10.11.0.1.zip` distribution.

5.1.1. 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 5.1](#) when packaging the project as a JAR. Make sure \${project.version} resolves to 10.11.0.1.

Figure 5.1. 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>
```

5.1.2. 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 5.2](#) when packaging the project as a JAR. Make sure \${project.version} resolves to 10.11.0.1.

Figure 5.2. 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>
```

5.1.3. 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 5.3](#) when packaging the project as a JAR. Make sure `${project.version}` resolves to 10.11.0.1.

Figure 5.3. 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>
```

5.1.4. 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 `Model-LogHandler` 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 5.4](#) when packaging the project as a JAR. Make sure `${project.version}` resolves to `10.11.0.1`.

Figure 5.4. 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>
```

5.2. Overview of code examples

The files under directory `adapa-extensions/samples` offer Java code examples for each use case. [Table 5.1](#) describes all the sample files in detail.

Table 5.1. Directory structure of code examples

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

Directory	Files	Description
	<code>CalcSomethingFactory.java</code>	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	<code>applicationContext.xml</code>	The application context XML file to be included.
	<code>FileLogHandler.java</code>	Contains methods for custom record logging.
	<code>FileLogHandlerFactory.java</code>	Factory for custom record logging.
lookuptable	<code>applicationContext.xml</code>	The application context XML file to be included.
	<code>GDPLookupTable.java</code>	The lookup table returns a GDP number corresponding to Country and State. Country, State and GDP are columns in the database table <code>GDP_Table</code> .
repository	<code>applicationContext.xml</code>	The application context XML file to be included.
	<code>MongoAssetRepository.java</code>	A sample <code>AssetRepository</code> for MongoDB.

5.3. 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-10.11.0.1.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 6. 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.

6.1. General Notes

6.1.1. 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.

Note

Starting from 10.11 (Oct 2021) release, Zementis REST supports additional static path in Request URI: `onnx` and `pmml` that denote different standards of machine learning algorithms. With backward compatibility support in Zementis REST, clients are still able to access all operations of machine learning algorithms in PMML standard without additional static path `pmml` in Request URI. For example: both `http://domain:port/adapars/models` and `http://domain:port/adapars/pmml/models` operations are available for listing all available PMML models in Zementis REST.

See [Figure 6.1](#) for a snapshot of the interactive REST API Documentation.

Figure 6.1. Interactive REST API Documentation

Zementis REST API



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 `/pmml/model/Iris_NN/source?annotated=true` contains static path definitions such as `pmml`, `model` or `source`, path parameter `Iris_NN` for a dynamically allocated resource, and a query parameter `annotated=true`.

6.1.2. 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 an otherwise static URI that denotes a set of possible resource names on the server and is denoted with curly braces. For example, our `/pmml/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 `/pmml/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 `/pmml/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.

6.1.3. 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`
- `application/octet-stream`

6.1.4. 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 6.1. Zementis REST Error Response

Request

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

Request Header

```
GET /adapars/pmml/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
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 59
Date: Wed, 01 Sep 2021 20:29:18 GMT
```

Response Body

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

Table 6.1. Typical Zementis REST Error Responses

Code	Error Message
400	One of below messages: <ul style="list-style-type: none"> • Empty input stream. • File name missing. • Invalid XML format. • Invalid JSON format. • Failed to parse JSON input. • Invalid CSV File. • Data for input field 'input_field_name' is unavailable. • Input field 'input_field_name' with type OnnxTensor expects an array.
401	This request requires HTTP authentication.
403	You are not authorized to access this resource.
404	One of below messages: <ul style="list-style-type: none"> • Model with name 'model_name' not found. • Model with id 'model_id' not found. • Resource file with name 'file_name' not found. • Resource file with id 'resource_id' not found. • Pipeline with name 'pipeline_name' not found. • Pipeline with id 'pipeline_id' not found.
409	One of below messages:

Code	Error Message
	<ul style="list-style-type: none"> • A model with the name 'model_name' already exists. • A resource with the name 'resource_name' already exists. • A class with name 'class_name' inside resource file 'file_name' already exists. • Resource file with id 'resource_id' not found. • Pipeline with name 'pipeline_name' not found. • Pipeline with id 'pipeline_id' not found.
415	'some_specific_data_type' is not supported yet.
500	<p>One of below messages:</p> <ul style="list-style-type: none"> • Invalid License. • Invalid resource file. • Source for 'model_name_or_id' unavailable. • Resource file with name 'file_name' is unavailable. • Resource file with id 'resource_id' is unavailable. • Error occurred while serializing JSON output. • Error occurred while creating tensor as ONNX model input. • Error occurred while reading value from ONNX model output.

6.1.5. 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. In the interactive REST API Documentation, the `Authorize` section provides our available authentication methods. Given proper authentication information in the `Authorize` section, you can start to communicate with Zementis Server via all the listed REST APIs. The table below lists the authorized role(s) for each operation, and the detailed description of each operation can be found in the following sections.

Table 6.2. Zementis REST Permissions

Group		Operation	Definition	Authorized Role(s)
PMML	Models	List Available PMML Models	GET /pmml/models	adapa-admin, adapa-ws-user
		Get PMML Model Information	GET /pmml/model/{model_name}	adapa-admin, adapa-ws-user
		Get PMML Model Source	GET /pmml/model/{model_name}/source	adapa-admin
		Get PMML Model Serialized Source	GET /pmml/model/{model_name}/serialized	adapa-admin
		Get PMML Model Metrics Information	GET /pmml/model/{model_name}/metrics	adapa-admin, adapa-ws-user
		Upload New PMML Model	POST /pmml/model	adapa-admin
		Activate an existing PMML Model	PUT /pmml/model/{model_name}/activate	adapa-admin
		Deactivate an existing PMML Model	PUT /pmml/model/{model_name}/deactivate	adapa-admin
		Remove PMML Model	DELETE /pmml/model/{model_name}	adapa-admin
		Remove All PMML Models	DELETE /pmml/models	adapa-admin

Group		Operation	Definition	Authorized Role(s)	
	Apply	Apply PMML Model to Single Record	GET /pmml/apply/{model_name}	adapa-admin, adapa-ws-user	
		Apply PMML Model to Single Record and Explain Result	GET /pmml/apply/{model_name}/explain	adapa-admin, adapa-ws-user	
		Apply PMML Model to Multiple Records	POST /pmml/apply/{model_name}	adapa-admin, adapa-ws-user	
		Asynchronously Apply PMML Model to Multiple Records	POST /pmml/apply/{model_name}/async	adapa-admin, adapa-ws-user	
	Resources	List Available PMML Resources	GET /pmml/resources	adapa-admin, adapa-ws-user	
		Get PMML Resource Information	GET /pmml/resource/{file_name}	adapa-admin, adapa-ws-user	
		Get PMML Resource File	GET /pmml/resource/{file_name}/source	adapa-admin	
		Upload New PMML Resource File	POST /pmml/resource	adapa-admin	
		Remove PMML Resource File	DELETE /pmml/resource/{file_name}	adapa-admin	
		Remove All PMML Resource Files	DELETE /pmml/resources	adapa-admin	
	ONNX	Models	List Available ONNX Models	GET /onnx/models	adapa-admin, adapa-ws-user
			Get ONNX Model Information	GET /onnx/models/{model_id}	adapa-admin, adapa-ws-user
			Get ONNX Model Source	GET /onnx/models/{model_id}/source	adapa-admin
Upload New ONNX Model			POST /onnx/models	adapa-admin	

Group		Operation	Definition	Authorized Role(s)
		Activate an existing ONNX Model	PUT /onnx/models/{model_id}/activate	adapa-admin
		Deactivate an existing ONNX Model	PUT /onnx/models/{model_id}/deactivate	adapa-admin
		Remove ONNX Model	DELETE /onnx/models/{model_id}	adapa-admin
		Remove All ONNX Models	DELETE /onnx/models	adapa-admin
	Apply	Apply ONNX Model to Input Data	POST /onnx/apply/{model_id}	adapa-admin, adapa-ws-user
		Apply ONNX Pipeline to Input Data	POST /onnx/apply-pipeline/{pipeline_id}	adapa-admin, adapa-ws-user
	Resources	List Available ONNX Resources	GET /onnx/resources	adapa-admin, adapa-ws-user
		Get ONNX Resource Information	GET /onnx/resources/{resource_id}	adapa-admin, adapa-ws-user
		Get ONNX Resource File	GET /onnx/resources/{resource_id}/source	adapa-admin
		Upload New ONNX Resource File	POST /onnx/resources	adapa-admin
		Remove ONNX Resource File	DELETE /onnx/resources/{resource_id}	adapa-admin
		Remove All ONNX Resource Files	DELETE /onnx/resources	adapa-admin
	Pipelines	List Available ONNX Pipelines	GET /onnx/pipelines	adapa-admin, adapa-ws-user
		Get ONNX Pipeline Information	GET /onnx/pipelines/{pipeline_id}	adapa-admin, adapa-ws-user

Group		Operation	Definition	Authorized Role(s)
		Create New ONNX Pipeline	POST /on-nx/pipelines	adapa-admin
		Remove ONNX Pipeline	DELETE /on-nx/pipelines/{pipeline_id}	adapa-admin
		Remove All ONNX Pipelines	DELETE /on-nx/pipelines	adapa-admin
License		Get License	GET /license	adapa-admin
		Post License	POST /license	adapa-admin

6.2. API

Zementis REST has two parts of APIs denoted by static path identifiers:

- `onnx` or `pmml` for accessing different standards of machine learning algorithms.
- `models`, `apply`, `apply-pipeline`, and `resources` for machine learning algorithm management and prediction/inferences.

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` as an example with user permissions to execute all REST API operations. All PMML examples also include `Iris_NN` PMML model which can be found in the executable samples package.

6.2.1. JSON Objects

This section describes the content of different responses from Zementis REST in JSON format:

Errors

Error messages container.

Properties

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

Example 6.2. Zementis REST Errors Object

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

PMML Models

An array of model names with key `models`.

Properties

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

Example 6.3. Zementis REST PMML Models Object

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

PMML ModelInfo

PMML model information.

Properties

modelName (string): model name

description (string): model description

isActive (boolean): model currently loaded into memory

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

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

Example 6.4. Zementis REST PMML 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"
    }
  ]
}

```

PMML Field

PMML 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 6.5. Zementis REST PMML Field Object

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

PMML MetricsInfo

PMML model metrics.

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 6.6. Zementis REST PMML 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
  }
}
```

PMML Record

Object, record used in PMML standard machine learning algorithm, 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 6.7. Zementis REST PMML Record Object

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

PMML Records

Anonymous array of [PMML Record](#) objects used to represent multiple input or output records.

Example 6.8. Zementis REST PMML Records 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[[PMML Record](#)]) or one combined output ([ONNX Records](#)): array of output [PMML Record](#) objects or one output combination [ONNX Records](#) object that output values persisting separately in each output field.

Example 6.9. Zementis REST PMML 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
    }
  ]
}
```

Example 6.10. Zementis REST ONNX Result Object

```
{
  "model": "example_model",
  "outputs": [
    {
      "output_label(tensor(INT64))": [
        0,
        1,
        2
      ]
    },
    {
      "output_probability(sequence(map(INT64,FLOAT)))": [
        {
          "0": 0.99999934,
          "1": 0,
          "2": 0
        },
        {
          "0": 0,
          "1": 0.9499994,
          "2": 0.049999997
        },
        {
          "0": 0,
          "1": 0.03,
          "2": 0.9699994
        }
      ]
    }
  ]
}
```

PMML ResourceInfo

PMML resource file information.

Properties

fileName (string): file name

resourceType (string): resource type

resourceIdentifier (string): resource identifier

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

Example 6.11. Zementis REST PMML ResourceInfo Object

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

PMML Resources

An array of [PMML ResourceInfo](#) objects with key resources.

Example 6.12. Zementis REST PMML 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"
      ]
    }
  ]
}

```

ONNX Records

A combination output used to represent multiple input or output records from one or multiple output field(s).

Example 6.13. Zementis REST ONNX Records Object

```

[
  {
    "output_label(tensor(INT64))": [
      0,
      1,
      2
    ]
  },
  {
    "output_probability(sequence(map(INT64,FLOAT)))": [
      {
        "0": 0.99999934,
        "1": 0,
        "2": 0
      },
      {
        "0": 0,
        "1": 0.9499994,
        "2": 0.049999997
      },
      {
        "0": 0,
        "1": 0.03,
        "2": 0.9699994
      }
    ]
  }
]

```

ONNX Models

An array of [ONNX ModelInfo](#) objects with models.

Example 6.14. Zementis REST ONNX Model Information List

```

{
  "models": [
    {
      "id" : "abcd...",
      "modelName" : "example_model_1",
      ...
    },
    {
      "id" : "1234...",
      "modelName" : "example_model_2",
      ...
    }
  ]
}

```

ONNX ModelInfo

ONNX model information.

Properties

id (string): the identifier of this model

modelName (string): the name of this model

modelVersion (long): the version of the graph encoded

onnxVersion (long): the version of the ONNX intermediate representation (IR) this model targets

opsetVersion (string): the OperatorSets this model relies on

producerName (string): the name of the framework or tool used to generate this model

producerVersion (string): the version of the framework or tool used to generate this model

domain (string): domain name of the model

isActive (boolean): the inference session status of this model

inputs (array[ONNX Field]): array of input [ONNX Field](#) objects

outputs (array[ONNX Field]): array of output [ONNX Field](#) objects

Example 6.15. Zementis REST ONNX ModelInfo Object

```
{
  "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4",
  "modelName" : "example_model",
  "modelVersion": 0,
  "onnxVersion": 4,
  "opsetVersion": "ai.onnx.ml v1, v9, ai.onnx.training v1,
ai.onnx.preview.training v1, com.microsoft v1, com.microsoft.nchwv1,
com.microsoft.mlfeaturizers v1",
  "producerName": "example_producer",
  "producerVersion": "1.0",
  "domain": "ai.onnx",
  "isActive": true,
  "inputs": [
    {
      "name": "float_input",
      "type": "tensor(FLOAT)",
      "shape": [
        -1,
        4
      ]
    }
  ],
  "outputs": [
    {
      "name": "output_label",
      "type": "tensor(INT64)",
      "shape": [
        -1
      ]
    },
    {
      "name": "output_probability",
      "type": "sequence(map(INT64, FLOAT))"
    }
  ]
}
```

ONNX Field

ONNX field information.

Properties

name (string): field name

type (string): field data type with one of standard ONNX data types: Tensor, Sequence, or Map along with one of valid DataType values: FLOAT, UINT8, INT8, UINT16, INT16, INT32, INT64, STRING, or BOOL

shape (array): field data shape if in Tensor type

Example 6.16. Zementis REST ONNX Field Object

```
{
  "name": "example_onnx_field_name",
  "type": "tensor(FLOAT)",
  "shape": [ -1, 4 ]
}
```


ONNX Resources

An array of [ONNX ResourceInfo](#) objects with key `resources`.

Example 6.17. Zementis REST ONNX Resources Object

```
{
  "resources":
  [
    {
      "id": "aca2a253...",
      "resourceFileName": "example_resource_1.jar",
      "resourcePreProcessors": [ ... ],
      "resourcePostProcessors": [ ... ]
    },
    {
      "id": "eeadev78...",
      "resourceFileName": "example_resource_2.jar",
      "resourcePreProcessors": [ ... ],
      "resourcePostProcessors": [ ... ]
    }
  ]
}
```

ONNX ResourceInfo

ONNX resource file.

Properties

id (string): resource identifier

resourceFileName (string): resource file name

resourcePreProcessors (string): array of Pre [ONNX Processor](#) information

resourcePostProcessors (string): array of Post [ONNX Processor](#) information

Example 6.18. Zementis REST ONNX ResourceInfo Object

```
{
  "id": "aca2a253-009b-4dc0-b2a0-5fcf8b815676",
  "resourceFileName": "RandomForestClassifierIris.jar",
  "resourcePreProcessors": [
    {
      "className": "com.zementis.processor.RandomForestClassifierIris",
      "inputDataType": "InputStream",
      "outputDataType": "Map<String, OnnxValue>"
    }
  ],
  "resourcePostProcessors": [
    {
      "className": "com.zementis.processor.RandomForestClassifierIris",
      "inputDataType": "Result",
      "outputDataType": "InputStream"
    }
  ]
}
```

ONNX ProcessorInfo

ONNX processor information.

Properties

className (string): class name

inputDataType (string): input data type. In post-processor, the input data type should be Result.

outputDataType (string): output data type. In pre-processor, the output data type should be Map<String, OnnxValue>.

Example 6.19. Zementis REST ONNX Processor Information

```
{
  "className": "com.zementis.processor.RandomForestClassifierIris",
  "inputDataType": "Result",
  "outputDataType": "InputStream"
}
```

ONNX Pipelines

An array of [ONNX PipelineInfo](#) objects with key pipelines.

Example 6.20. Zementis REST ONNX Pipelines Object

```
{
  "pipelines": [
    {
      "id": "7e931f38...",
      "pipelineName": "example_pipeline_1",
      "preProcessor": "com.zementis.processor.RandomForestClassifierIris",
      "modelName": "example_model_1",
      "postProcessor": "com.zementis.processor.RandomForestClassifierIris"
    },
    {
      "id": "eeadev78...",
      "pipelineName": "example_pipeline_2",
      "preProcessor": "com.zementis.processor.RandomForestClassifierIris",
      "modelName": "example_model_2"
    }
  ]
}
```

ONNX PipelineInfo

ONNX Pipeline information.

Properties

id (string): pipeline identifier

pipelineName (string): pipeline name

preProcessor (string): the referred pre-processor class name

modelName (string): the referred model name

postProcessor (string): the referred post-processor class name

Example 6.21. Zementis REST ONNX PipelineInfo Object

```
{
  "id": "7e931f38-3cce-4573-ab8a-48f7b7fba8c8",
  "pipelineName": "example_pipeline",
  "preProcessor": "com.zementis.processor.RandomForestClassifierIris",
  "modelName": "example_model",
  "postProcessor": "com.zementis.processor.RandomForestClassifierIris"
}
```

6.2.2. Operations on PMML Models

6.2.2.1. List Available PMML Models

Definition **GET /models**

GET /pmml/models

Get 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 [PMML Models](#) object if successful, [Errors](#) otherwise.

Example 6.22. Zementis REST List PMML Models

Request

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

Request Header

```
GET /adapars/pmml/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" ]
}
```

6.2.2.2. Get PMML Model Information

Definition	GET /model/{model_name} GET /pmml/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 PMML ModelInfo object if successful, Errors otherwise.

Example 6.23. Zementis REST Get PMML Model Information

Request

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

Request Header

```
GET /adapars/pmml/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"
  } ],
  ...
}
```

6.2.2.3. Get PMML Model Source

Definition	<p>GET /model/{model_name}/source</p> <p>GET /pmml/model/{model_name}/source</p> <p>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.</p>
Request Parameters	<p>model_name (string): required path variable for existing model name</p> <p>annotated (boolean): optional query parameter used to request the annotated version of the PMML file</p>
Returns	Returns the PMML source code if successful, Errors otherwise.

Example 6.24. Zementis REST Get PMML Model Source

Request

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

Request Header

```
GET /adapars/pmml/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"/>
...

```

6.2.2.4. Get PMML Model Serialized Source

Definition	GET /model/{model_name}/serialized GET /pmml/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 6.25. Zementis REST Get PMML Model Serialized

Request

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

Request Header

```

GET /adapars/pmml/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
...

```

6.2.2.5. Get PMML Model Metrics Information

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

GET /pmml/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 [PMML MetricsInfo](#) object if successful, [Errors](#) otherwise.

Example 6.26. Zementis REST Get PMML Model Metrics

Request

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

Request Header

```
GET /adapars/pmml/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
  }
}
```

6.2.2.6. Upload New PMML Model

Definition **POST /model**

POST /pmml/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 4.4.1 (default is `true`)

Returns

Returns a [PMML 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 6.27. Zementis REST Upload New PMML Model

Request

```
curl -u adapa:adapa -k https://localhost/adapars/pmml/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/pmml/model -X POST -F file=@Iris_NN.pmml
curl -u adapa:adapa -k https://localhost/adapars/pmml/model?applyCleanser=true -X POST -F
file=@Iris_NN.pmml
```

Request Header

```
POST /adapars/pmml/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/pmml/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",
```



```



```

6.2.2.7. Activate an existing PMML Model

Definition **PUT /model/{model_name}/activate**

PUT /pmml/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 [PMML ModelInfo](#) object and 200 HTTP status code.

Example 6.28. Zementis REST Activate an existing PMML Model

Request

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

Request Header

```

PUT /adapars/pmml/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,
  ...
}
```

6.2.2.8. Deactivate an existing PMML Model

Definition **PUT /model/{model_name}/deactivate**

PUT /pmml/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 [PMML ModelInfo](#) object and 200 HTTP status code.

Example 6.29. Zementis REST Deactivate an existing PMML Model

Request

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

Request Header

```
PUT /adapars/pmml/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,
  ...
}
```

6.2.2.9. Remove PMML Model

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

Example 6.30. Zementis REST Remove PMML Model

Request

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

Request Header

```
DELETE /adapars/pmml/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" ]
}
```

6.2.2.10. Remove All PMML Models

Definition	<p>DELETE /models</p> <p>DELETE /pmml/models</p> <p>Remove all available models and list the remaining models.</p>
------------	--

Request Parameters	None
Returns	Returns a PMML Models object with an empty <code>models</code> array if successful, an Errors object otherwise.

Example 6.31. Zementis REST Remove All PMML Models

Request

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

Request Header

```
DELETE /adapars/pmml/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" : [ ]
}
```

6.2.3. Apply PMML model

6.2.3.1. Apply PMML Model to Single Record

Definition	GET /apply/{model_name} GET /pmml/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 (PMML Record) : optional query parameter for input PMML Record
Returns	Returns Result object if successful, Errors otherwise.

Example 6.32. Zementis REST Apply PMML Model to Single Record

Request

```
curl -u adapa:adapa -k https://localhost/adapars/pmml/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/pmml/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
  } ]
}
```

6.2.3.2. Apply PMML Model to Single Record and Explain Result

Definition

GET /apply/{model_name}/explain

GET /pmml/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 ([PMML Record](#)) : optional query parameter for input [PMML Record](#)

Returns

Returns a result in plain text if successful, [Errors](#) otherwise.

Example 6.33. Zementis REST Apply PMML Model to Single Record and Explain Result

Request

```
curl -u adapa:adapa -k https://localhost/adapars/pmml/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/pmml/apply/Iris_NN/explain?record=%7B%22sepal_length%22%3A5.1%2C%22sepal_width%22%3A3.5%2C%22petal_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] := 0.2222222222222213 (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)
```

6.2.3.3. Apply PMML Model to Multiple Records

Definition **POST /apply/{model_name}**

POST /pmml/apply/{model_name}

If a predictive model 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 [PMML 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.

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)

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 6.34. Zementis REST Apply PMML Model to Multiple Records

Request

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

Request Header

```
POST /adapars/pmml/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
...
```

6.2.3.4. Asynchronously Apply PMML Model to Multiple Records

Definition

POST /apply/{model_name}/async

POST /pmml/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 [PMML 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 6.35. Zementis REST Asynchronously Apply PMML Model to Multiple Records

Request

```
curl -u adapa:adapa -k https://localhost/adapars/pmml/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/pmml/apply/Iris_NN/async?maxThreads=8 -X POST \
-F file=@Iris_NN_CSV.properties
```

Request Header

```
POST /adapars/pmml/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'"
}
```

6.2.4. Operations on PMML Resources

6.2.4.1. List Available PMML Resources

Definition	<p>GET /resources</p> <p>GET /pmml/resources</p> <p>Get information on all available resource files uploaded on Zementis Server. Use file names as identifiers for all operations requiring a <code>file_name</code> path variable.</p>
Request Parameters	None
Returns	Returns a PMML Resources object if successful, an Errors object otherwise.

Example 6.36. Zementis REST List PMML Resources

Request

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

Request Header

```
GET /adapars/pmml/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" ]
  } ]
}
```

6.2.4.2. Get PMML Resource Information

Definition

GET /resource/{file_name}

GET /pmml/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 [PMML ResourceInfo](#) object if successful, an [Errors](#) object otherwise.

Example 6.37. Zementis REST Get PMML Resource Information

Request

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

Request Header

```
GET /adapars/pmml/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" ]
}
```

6.2.4.3. Get PMML Resource File

Definition **GET /resource/{file_name}/source**

GET /pmml/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 6.38. Zementis REST Get PMML Resource File

Request

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

Request Header

```
GET /adapars/pmml/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??
...
```

6.2.4.4. Upload New PMML Resource File

Definition **POST /resource**

POST /pmml/resource

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

Content-Type (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

Returns Returns [PMML 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 6.39. Zementis REST Upload New PMML Resource File

Request

```
curl -u adapa:adapa -k https://localhost/adapars/pmml/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/pmml/resource -X POST -F file=@ECommerceFraud_NN.xls
```

Request Header

```
POST /adapars/pmml/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/pmml/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" ]
}
```

6.2.4.5. Remove PMML Resource File

Definition **DELETE /resource/{file_name}**

DELETE /pmml/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 [PMML Resources](#) object with a list of all remaining resource files if successful, an [Errors](#) object otherwise.

Example 6.40. Zementis REST Remove PMML Resource File

Request

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

Request Header

```
DELETE /adapars/pmml/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" ]
  } ]
}

```

6.2.4.6. Remove All PMML Resource Files

Definition **DELETE /resources**

DELETE /pmml/resources

Remove all available resources and list the remaining resources.

Request Parameters None

Returns Returns a [PMML Resources](#) object with an empty `resources` array if successful, an [Errors](#) object otherwise.

Example 6.41. Zementis REST Remove All PMML Resource Files

Request

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

Request Header

```

DELETE /adapars/pmml/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" : [ ]
}
```

6.2.5. Operations on ONNX Models

6.2.5.1. List Available ONNX Models

Definition

GET /onnx/models

GET /onnx/models?model_name={model_name}

Get the ONNX model information of all the available ONNX models in Zementis Server. Use `model_name` as a query string to filter specified ONNX model from the model information list. Use model identifiers for all operations requiring the `model_id` path variable.

Request Parameters

model_name (string): optional query variable for filtering model by `model_name`

Returns

Returns [ONNX Models](#) object if successful, [Errors](#) object otherwise.

Example 6.42. Zementis REST List ONNX Models

Request

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

Request Header

```
GET /adapars/onnx/models HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 783
Date: Tue, 07 Sep 2021 20:20:24 GMT
```

Response Body

```
{
  "models" : [ {
    "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4",
```

```

"modelName" : "example_model_1",
"modelVersion" : 0,
"onnxVersion" : 4,
"opsetVersion" : "ai.onnx.ml v1...",
"producerName" : "producer_1",
"producerVersion" : "1.9.0",
"domain" : "ai.onnx",
"isActive" : true,
"inputs" : [ {
  "name" : "float_input",
  "type" : "tensor(FLOAT)",
  "shape" : [ -1, 4 ]
} ],
"outputs" : [ {
  "name" : "output_label",
  "type" : "tensor(INT64)",
  "shape" : [ -1 ]
}, {
  "name" : "output_probability",
  "type" : "sequence(map(INT64,FLOAT))"
} ]
} ]
}

```

6.2.5.2. Get ONNX Model Information

Definition	GET /onnx/models/{model_id} Get information of the specified ONNX model.
Request Parameters	model_id (string): required path variable for an existing model identifier
Returns	Returns ONNX ModelInfo object if successful, Errors object otherwise.

Example 6.43. Zementis REST Get ONNX Model Information

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4
```

Request Header

```

GET /adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*

```

Response Header

```

HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 712
Date: Tue, 07 Sep 2021 20:24:52 GMT

```

Response Body

```
{
```



```

"id" : "ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4",
"modelName" : "example_model_1",
"modelVersion" : 0,
"onnxVersion" : 4,
"opsetVersion" : "ai.onnx.ml v1...",
"producerName" : "producer_1",
"producerVersion" : "1.9.0",
"domain" : "ai.onnx",
"isActive" : true,
"inputs" : [ ... ],
"outputs" : [ ... ]
}

```

6.2.5.3. Get ONNX Model Source

Definition	GET /onnx/models/{model_id}/source Download original ONNX file.
Request Parameters	model_id (string): required path variable for an existing model identifier
Returns	Returns the ONNX file if successful, Errors object otherwise.

Example 6.44. Zementis REST Get ONNX Model Source

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4/source --output model_source.onnx
```

Request Header

```

GET /adapars/onnx/models/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4/source HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*

```

Response Header

```

HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/octet-stream
Transfer-Encoding: chunked
Date: Tue, 07 Sep 2021 21:45:28 GMT

```

Response Body

```
8006 bytes data ...
```

6.2.5.4. Upload New ONNX Model

Definition	POST /onnx/models Upload a new ONNX model.
------------	--

Request Parameters	Content-Type (string): required header parameter with one of the listed values: application/octet-stream OR multipart/form-data
Returns	Returns ONNX ModelInfo object with 201 HTTP response status code, and a response header entry <code>Location</code> with URI of the created ONNX model if the upload was successful. Errors object otherwise.

Example 6.45. Zementis REST Upload New Model

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models -X POST -T
RandomForestClassifierIris.onnx -H "Content-Type:application/octet-stream"
curl -u adapa:adapa -k https://localhost/adapars/onnx/models -X POST -F
file=@RandomForestClassifierIris.onnx
```

Request Header

```
POST /adapars/onnx/models HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
Content-Length: 57646
Expect: 100-continue
Content-Type: multipart/form-data; boundary=-----c117b742fdbd390f
```

Response Header

```
HTTP/1.1 201 Created
Location: https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4
Content-Type: application/json
Content-Length: 588
Date: Tue, 07 Sep 2021 21:56:25 GMT
```

Response Body

```
{
  "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4",
  "modelName" : "RandomForestClassifierIris",
  ...
  "inputs" : [ ... ],
  "outputs" : [ ... ]
}
```

6.2.5.5. Activate an existing ONNX Model

Definition	PUT /onnx/model/{model_id}/activate
	Activates the model with <code>model_id</code> 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.
Request Parameters	model_id (string): required path variable for existing model identifier
Returns	Returns an ONNX ModelInfo object and 200 HTTP status code.

Example 6.46. Zementis REST Activate an existing ONNX Model

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4/activate -X PUT
```

Request Header

```
PUT /adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4/activate HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 588
Date: Tue, 07 Sep 2021 22:21:12 GMT
```

Response Body

```
{
  "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4",
  "modelName" : "example_model_1",
  ...
  "isActive" : true,
  "inputs" : [ ... ],
  "outputs" : [ ...]
}
```

6.2.5.6. Deactivate an existing ONNX Model

Definition `PUT /onnx/model/{model_id}/deactivate`

De-activates the model with `model_id` 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_id (string)`: required path variable for existing model identifier

Returns Returns an [ONNX ModelInfo](#) object and 200 HTTP status code.

Example 6.47. Zementis REST Deactivate an existing ONNX Model

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4/deactivate -X PUT
```

Request Header

```
PUT /adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4/deactivate HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
```

```
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 588
Date: Tue, 07 Sep 2021 22:23:56 GMT
```

Response Body

```
{
  "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4",
  "modelName" : "example_model_1",
  ...
  "isActive" : false,
  "inputs" : [ ... ],
  "outputs" : [ ... ]
}
```

6.2.5.7. Remove ONNX Model

Definition	DELETE /onnx/models/{model_id} Remove the specified ONNX model and list the remaining ONNX models.
Request Parameters	model_id (string): required path variable for an existing model identifier
Returns	Returns ONNX Models object containing remaining ONNX models if successful, Errors object otherwise.

Example 6.48. Zementis REST Remove ONNX Model

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4 -X DELETE
```

Request Header

```
DELETE /adapars/onnx/models/ef1858e5-b7a0-49e4-a1e2-cb30b8cbafb4 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 59
Date: Tue, 07 Sep 2021 22:50:22 GMT
```

Response Body

```
{
  "models" : [ {
    "id" : "ef1858e5-b7a0-49e4-a1e2-cb30b8cbaf4",
    "modelName" : "example_remaining_model",
    ...
    "inputs" : [ ... ],
    "outputs" : [ ... ]
  } ]
}
```

6.2.5.8. Remove All ONNX Models

Definition	DELETE /onnx/models Remove all available ONNX models and list the remaining ONNX models.
Request Parameters	None
Returns	Returns ONNX Models object with an empty <code>models</code> array if successful, Errors object otherwise.

Example 6.49. Zementis REST Remove All ONNX Models

Request

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

Request Header

```
DELETE /adapars/onnx/models HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 24
Date: Tue, 07 Sep 2021 22:52:53 GMT
```

Response Body

```
{
  "models" : [ ]
}
```

6.2.6. Apply ONNX model

6.2.6.1. Apply ONNX Model to Data File

Definition	POST /onnx/apply/{model_id}
------------	------------------------------------

Apply a model to a file or binary data containing the input record(s) in JSON format.

Request Parameters

Content-Type (string) : required header parameter with one of the listed values: application/octet-stream, multipart/form-data, or application/json.

model_id (string) : required path variable for the identifier of the ONNX model to be applied.

Returns Returns results as [Result](#) object if successful, [Errors](#) otherwise.

Example 6.50. Zementis REST Apply ONNX Model to Data File

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/apply/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4 -X POST -H 'Content-Type: Multipart/form-data' -F file=@data.json
curl -u adapa:adapa -k https://localhost/adapars/onnx/apply/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4 -X POST -H 'Content-Type: application/octet-stream' -T data.json
curl -u adapa:adapa -k https://localhost/adapars/onnx/apply/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4 -X POST -H 'Content-Type: application/json' -d @data.json
curl -u adapa:adapa -k https://localhost/adapars/onnx/apply/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4 -X POST -H 'Content-Type: application/json' --data-binary '{"float_input": [[5.1, 3.5, 1.4, 0.2],[5.9, 3.0, 5.1, 1.8]]}'
```

Request Header

```
POST /adapars/onnx/apply/ef1858e5-b7a0-49e4-ale2-cb30b8cbafb4 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
Content-Length: 3852
Expect: 100-continue
Content-Type: multipart/form-data; boundary=-----79756ab37ee4122e
```

Response Header

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Date: Wed, 08 Sep 2021 03:20:57 GMT
```

Response Body

```
{
  "model": "example_model_1",
  "outputs": [
    {
      "output_label": [
        0,
        2
      ]
    },
    {
      "output_probability": [
        {
          "0": 0.99999934,
          "1": 0,
          "2": 0
        }
      ]
    }
  ]
}
```

```

    {
      "0": 0,
      "1": 0.03,
      "2": 0.9699994
    }
  ]
}

```

6.2.6.2. Apply ONNX Pipeline to Data File

Definition	<p>POST /onnx/apply-pipeline/{pipeline_id}</p> <p>Apply a pipeline to a file containing the input record(s) in JSON format or raw file that could be converted as proper ONNX model input record(s) via pre-processor.</p>
Request Parameters	<p>Content-Type (string) : required header parameter with one of the listed values: application/octet-stream or multipart/form-data.</p> <p>pipeline_id (string) : required path variable for the identifier of the ONNX pipeline to be applied.</p>
Returns	<p>Returns results as Result object if successful, or output data that is converted from ONNX model inferences via post-processor. Errors otherwise.</p>

Example 6.51. Zementis REST Apply ONNX Pipeline to Data File

Request

```

curl -u adapa:adapa -k https://localhost/adapars/onnx/apply-pipeline/
e6a83f70-2f92-4e11-9f1f-798d40331f54 -X POST -H 'Content-Type: Multipart/form-data' -F
file=@elephant.png
curl -u adapa:adapa -k https://localhost/adapars/onnx/apply-pipeline/
e6a83f70-2f92-4e11-9f1f-798d40331f54 -X POST -H 'Content-Type: application/octet-stream' -T elephant.png

```

Request Header

```

POST /adapars/onnx/apply-pipeline/e6a83f70-2f92-4e11-9f1f-798d40331f54 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
Content-Length: 96839
Expect: 100-continue
Content-Type: multipart/form-data; boundary=-----79756ab37ee4122e

```

Response Header

```

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 84
Date: Thu, 09 Sep 2021 21:51:00 GMT

```

Response Body

The identified object in the picture is [African_elephant] with confidence 0.8857361

6.2.7. Operations on ONNX Resources

6.2.7.1. List Available ONNX Resources

Definition **GET /onnx/resources**

GET /onnx/resources?file_name={file_name}

This operation retrieves the ONNX resource information of all the available ONNX resource files in Zementis Server. Use `file_name` as a query string to filter specified ONNX resource file from the resource information list. Use resource identifiers for all operations requiring the `resource_id` path variable.

Request Parameters **file_name (string)**: optional query variable for filtering resource by `file_name`

Returns Returns [ONNX Resources](#) object if successful, [Errors](#) object otherwise.

Example 6.52. Zementis REST List ONNX Resources

Request

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

Request Header

```
GET /adapars/onnx/resources HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 992
Date: Thu, 09 Sep 2021 17:08:25 GMT
```

Response Body

```
{
  "resources" : [ {
    "id" : "86b2b844...",
    "resourceFileName" : "example_resource_1.jar",
    "resourcePreProcessors" : [ ... ],
    "resourcePostProcessors" : [ ... ]
  }, {
    "id" : "da5abbe57...",
    "resourceFileName" : "example_resource_2.jar",
    "resourcePreProcessors" : [ ... ],
    "resourcePostProcessors" : [ ... ]
  } ]
}
```


6.2.7.2. Get ONNX Resource Information

Definition	GET /onnx/resources/{resource_id} Get information of the specified ONNX resource file.
Request Parameters	resource_id (string): required path variable for an existing resource identifier
Returns	Returns a ONNX ResourceInfo object if successful, Errors object otherwise.

Example 6.53. Zementis REST Get ONNX Resource Information

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf
```

Request Header

```
GET /adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 460
Date: Thu, 09 Sep 2021 17:11:51 GMT
```

Response Body

```
{
  "id" : "86b2b844-6c58-4fa5-ba2e-da5abbe578cf",
  "resourceFileName" : "example_resource_1.jar",
  "resourcePreProcessors" : [ {
    "className" : "com.zementis.processor.RandomForestClassifierIris",
    "inputDataType" : "InputStream",
    "outputDataType" : "Map<String, OnnxValue>"
  } ],
  "resourcePostProcessors" : [ {
    "className" : "com.zementis.processor.RandomForestClassifierIris",
    "inputDataType" : "Result",
    "outputDataType" : "InputStream"
  } ]
}
```

6.2.7.3. Get ONNX Resource File

Definition	GET /onnx/resources/{resource_id}/source Download ONNX resource file.
Request Parameters	resource_id (string): required path variable for an existing resource identifier

Returns `ONNXResourceFile` if successful, `Errors` object otherwise.

Example 6.54. Zementis REST Get ONNX Resource File

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf/source --output resource.jar
```

Request Header

```
GET /adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf/source HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/octet-stream
Content-Length: 2990
Date: Thu, 09 Sep 2021 17:17:24 GMT
```

Response Body

```
2990 bytes data ...
```

6.2.7.4. Upload New ONNX Resource File

Definition

POST /onnx/resources

Upload a new ONNX resource file. The file name in 'file' body parameter will be used to identify this resource. if the request is with Content-Type `application/octet-stream`.

Request Parameters

Content-Type (string): required header parameter with one of the listed values: `application/octet-stream` or `multipart/form-data`

Content-Type (string): required query parameter for ONNX resource name, if Content-Type is `application/octet-stream`; optional query parameter if Content-Type is `multipart/form-data`

Returns

Returns `ONNXResourceInfo` object with 201 HTTP response status code, and a response header entry `Location` with URI of the created resource if the upload was successful. `Errors` object otherwise.

Example 6.55. Zementis REST Upload New ONNX Resource File

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/resources?file=example_resource_1.jar -X POST -T
example_resource_1.jar -H "Content-Type:application/octet-stream"
curl -u adapa:adapa -k https://localhost/adapars/onnx/resources -X POST -F
file=@example_resource_2.jar
```

Request Header

```
POST /adapars/onnx/resources?file=example_resource_1.jar HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
Content-Length: 24164
Expect: 100-continue
Content-Type: application/octet-stream
```

Response Header

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://localhost:8080/adapars/resource/19edee6a-1f74-40d1-9b6b-e4f077a637a5
Content-Type: application/json
Content-Length: 440
Date: Thu, 09 Sep 2021 20:53:41 GMT
```

Response Body

```
{
  "id" : "19edee6a-1f74-40d1-9b6b-e4f077a637a5",
  "resourceFileName" : "example_resource_1.jar",
  "resourcePreProcessors" : [ ... ],
  "resourcePostProcessors" : [ ... ]
}
```

6.2.7.5. Remove ONNX Resource File

Definition	DELETE /onnx/resources/{resource_id}
	Remove the specified ONNX resource file and list the remaining ONNX resources.
Request Parameters	resource_id (string): required path variable for an existing resource identifier
Returns	Returns ONNX Resources object containing remaining ONNX resources if successful, Errors object otherwise.

Example 6.56. Zementis REST Remove ONNX Resource File

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf -X DELETE
```

Request Header

```
DELETE /adapars/onnx/resources/86b2b844-6c58-4fa5-ba2e-da5abbe578cf HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
```

```
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 460
Date: Thu, 09 Sep 2021 17:23:46 GMT
```

Response Body

```
{
  "resources" : [ {
    "id" : "da5abbe57...",
    "resourceFileName" : "example_resource_2.jar",
    "resourcePreProcessors" : [ ... ],
    "resourcePostProcessors" : [ ... ]
  } ]
}
```

6.2.7.6. Remove All ONNX Resource Files

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

Example 6.57. Zementis REST Remove All ONNX Resource Files

Request

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

Request Header

```
DELETE /adapars/onnx/resources HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 23
Date: Thu, 09 Sep 2021 17:26:41 GMT
```

Response Body

```
{
```

```
"resources" : [ ]
}
```

6.2.8. Operations on ONNX Pipelines

6.2.8.1. List Available ONNX Pipelines

Definition

GET /onnx/pipelines

GET /onnx/pipelines?pipeline_name={pipeline_name}

This operation retrieves the ONNX pipeline information of all the available ONNX pipelines in Zementis Server. Use `pipeline_name` as a query string to filter specified ONNX pipeline from the pipeline information list. Use pipeline identifiers for all operations requiring the `pipeline_id` path variable.

Request Parameters

pipeline_name (**string**): optional query variable for filtering pipeline by `pipeline_name`

Returns

Returns [ONNX Pipelines](#) object if successful, [Errors](#) object otherwise.

Example 6.58. Zementis REST List ONNX Pipelines

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines
```

Request Header

```
GET /adapars/onnx/pipelines HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 992
Date: Thu, 09 Sep 2021 21:08:15 GMT
```

Response Body

```
{
  "pipelines" : [
    {
      "id" : "86b2b844...",
      "pipelineName" : "example_pipeline_1",
      "preProcessor" : "com.zementis.class1",
      "modelName" : "example_model_1",
```

```

    "postProcessor" : "com.zementis.class2"
  }, {
    "id" : "dfa3ec15...",
    "pipelineName" : "example_pipeline_2",
    "preProcessor" : "com.zementis.class1",
    "modelName" : "example_model_1"
  }, {
    "id" : "da5abbe57...",
    "pipelineName" : "example_pipeline_3",
    "modelName" : "example_model_2"
  }, {
    "id" : "4fa5b532...",
    "pipelineName" : "example_pipeline_4",
    "modelName" : "example_model_2",
    "postProcessor" : "com.zementis.class2"
  }
]
}

```

6.2.8.2. Get ONNX Pipeline Information

Definition `GET /onnx/pipelines/{pipeline_id}`

Get information on the specified ONNX pipeline.

Request Parameters `pipeline_id (string)`: required path variable for an existing pipeline identifier

Returns Returns a [ONNX PipelineInfo](#) object if successful, [Errors](#) object otherwise.

Example 6.59. Zementis REST Get ONNX Pipeline Information

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines/72ad114b-79c9-4dc8-b899-7b2ec16ee5c6
```

Request Header

```

GET /adapars/onnx/pipelines/72ad114b-79c9-4dc8-b899-7b2ec16ee5c6 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*

```

Response Header

```

HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 460
Date: Thu, 09 Sep 2021 21:13:46 GMT

```

Response Body

```

{
  "id" : "72ad114b-79c9-4dc8-b899-7b2ec16ee5c6",
  "pipelineName" : "example_pipeline_1",
  "preProcessor" : "com.zementis.class1",
  "modelName" : "example_model_1",
  "postProcessor" : "com.zementis.class2",

```

```
}
```

6.2.8.3. Create New ONNX Pipeline

Definition	<p>POST /onnx/pipelines</p> <p>Create a new ONNX pipeline definition.</p>
Request Parameters	<p>Content-Type (string): required header parameter with one of below values: application/octet-stream, multipart/form-data, or application/json</p>
Returns	<p>Returns ONNX PipelineInfo object with 201 HTTP response status code, and a response header entry <code>Location</code> with URI of the created pipeline if the upload was successful. Errors object otherwise.</p>

Example 6.60. Zementis REST Create New ONNX Pipeline

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines -X POST -T example_pipeline_1.json -H 'Content-Type:application/octet-stream'
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines -X POST -F file=@example_pipeline_2.json
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines -X POST -d @example_pipeline_3.json -H 'Content-Type: application/json'
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines -X POST --data-binary '{"pipelineName": "example_pipeline_4", "modelName": "example_model", "preProcessor": "className1", "postProcessor": "className2"}'
```

Request Header

```
POST /adapars/onnx/pipelines HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
Content-Length: 164
Expect: 100-continue
Content-Type:application/octet-stream
```

Response Header

```
HTTP/1.1 201 Created
Location: http://localhost:8080/adapars/onnx/pipelines/dfa3ec15-c515-49ee-a980-659549eba5b6
Content-Type: application/json
Content-Length: 218
Date: Thu, 09 Sep 2021 21:02:30 GMT
```

Response Body

```
{
  "id" : "dfa3ec15-c515-49ee-a980-659549eba5b6",
  "pipelineName" : "example_pipeline_1",
  "preProcessor" : "com.zementis.processor.example1",
  "modelName" : "example_model_1",
  "postProcessor" : "com.zementis.processor.example1"
```

```
}
```

6.2.8.4. Remove ONNX Pipeline

Definition	DELETE /onnx/pipelines/{pipeline_id}
	Remove the specified ONNX pipeline and list the remaining ONNX pipelines.
Request Parameters	pipeline_id (string) : required path variable for an existing pipeline identifier
Returns	Returns ONNX Pipelines object containing remaining ONNX pipelines if successful, Errors object otherwise.

Example 6.61. Zementis REST Remove ONNX Pipeline

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines/72ad114b-79c9-4dc8-b899-7b2ec16ee5c6 -X DELETE
```

Request Header

```
DELETE /adapars/onnx/pipelines/72ad114b-79c9-4dc8-b899-7b2ec16ee5c6 HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 460
Date: Thu, 09 Sep 2021 21:15:45 GMT
```

Response Body

```
{
  "pipelines" : [
    {
      "id" : "dfa3ec15...",
      "pipelineName" : "example_pipeline_2",
      "preProcessor" : "com.zementis.class1",
      "modelName" : "example_model_1"
    }, {
      "id" : "da5abbe57...",
      "pipelineName" : "example_pipeline_3",
      "modelName" : "example_model_2"
    }
  ]
}
```

6.2.8.5. Remove All ONNX Pipelines

Definition	DELETE /onnx/pipelines
------------	-------------------------------

Remove all available ONNX pipelines and list the remaining ONNX pipelines.

Request Parameters None

Returns Returns [ONNX Pipelines](#) object with an empty `pipelines` array if successful, [Errors](#) object otherwise.

Example 6.62. Zementis REST Remove All ONNX Pipelines

Request

```
curl -u adapa:adapa -k https://localhost/adapars/onnx/pipelines -X DELETE
```

Request Header

```
DELETE /adapars/onnx/pipelines HTTP/1.1
Authorization: Basic YWRhcGE6YWRhcGE=
User-Agent: curl/7.64.1
Host: localhost
Accept: */*
```

Response Header

```
HTTP/1.1 200 OK
Cache-Control: private
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json
Content-Length: 23
Date: Thu, 09 Sep 2021 21:17:04 GMT
```

Response Body

```
{
  "pipelines" : [ ]
}
```

6.2.9. Operations on License

6.2.9.1. 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 6.63. 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."
}
```

6.2.9.2. 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 6.64. 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."
}
```