MASTER'S THESIS AT CSC, KTH

Software Design for Financial Risk Analysis

Mjukvarudesign för finansiell riskanalys

Master's thesis project by Sven Trebard

Email: wiz@wiz.se

Supervisor: Henrik Eriksson

Examiner: Yngve Sundblad

Commissioner: Nada Risk Europe AB

Software Design for Financial Risk Analysis

Abstract

This document is a compilation of ideas and reflections from building software for financial presentation and analysis. The target readers are financial risk analysts and developers of financial analysis software. Software of this nature is often developed in-house, often using Microsoft Excel, and the risk models are often proprietary. This document may be an entry point for developing a more structured and robust system.

More specifically, this document illuminates requirements and solutions developing and supporting PowerPilot, a program developed for and used by the risk group at Nord Pool Clearing ASA, a clearing house for the Nordic power market.

Mjukvarudesign för finansiell riskanalys

Sammanfattning

Denna rapport är en sammanfattning av idéer och erfarenheter från utveckling av mjukvara för presentation och analys av finansiella data. Den är skriven för finansiella riskanalytiker och utvecklare av mjukvara för finansiell analys. Mjukvara av denna typ är ofta utvecklad internt på företaget, ofta med Microsoft Excel, och riskmodellerna är ofta egenutvecklade. Detta dokument kan vara en utgångspunkt för utveckling av mer strukturerade och robusta system.

Mer specifikt belyser detta dokument behov och lösningar för att utveckla och underhålla PowerPilot, ett program som utvecklades för och används av riskgruppen på Nord Pool Clearing ASA, ett clearing-hus för den nordiska elmarknaden.

Foreword

I was employed by the quantitative analysis group at Swedbank Markets in 1996 to develop NetTrade, with my colleague Jörgen Blomberg. NetTrade was the first web application for online stock trading in Europe.

After this, I started developing software for technical analysis, i.e. the use of statistics and group trading behavior analysis as guidelines for equity-, currency- and fixed income trading.

In 2000, I cofounded Qitech AB, where my colleagues Richard Werneker and Josefin Bodell Werneker traded power contracts.

In 2002, Richard, Josefin and I founded Nada Risk Europe AB, and built a risk analysis tool for the power market, focusing on portfolio summaries and risk analysis. This software is called PowerPilot and the primary customer is Nord Pool Clearing, the main Nordic clearing house for power derivatives. PowerPilot is keeping track of several hundred member companies. The design is not limited to the power market, although much of the mathematics is custom-made for financial power analysis.

I would like to thank Richard Werneker and Josefin Bodell Werneker. I would also like to thank Trond Svensgaard and Håkan Sandebjer at Nord Pool for their invaluable opinions and demands.

Index of contents

1. Background

1.1 Nord Pool Clearing ASA

"Nord Pool Clearing ASA enters into financial contracts as a contractual counter-party. This means that Nord Pool assumes liability for covering the future clearing of financial contracts in order to reduce the risk of the contracts for buyers and sellers. Nord Pool Clearing provides clearing for financial, standardised electricity contracts traded on and off the exchange." (Nord Pool, 2009)

Nord Pool Clearing is a clearing house, acting as a middleman of trades to shield both parties of a transaction from the risk of the counterparty defaulting, in exchange for a commission. If one part of a commitment or contract defaults, i.e. is unable to pay its debts, Nord Pool Clearing assumes its liabilities. To cover for the potential losses of a defaulting member company, Nord Pool has to calculate an estimated risk of assuming the commitments of the company. Each member company has to have a surety or guarantor, e.g. a bank, for this amount.

1.2 PowerPilot

PowerPilot is a program for account presentation and financial risk analysis that was designed and implemented by me, mainly between 2002 and 2004, using the Java programming language.

In 2002, Nord Pool had a system for calculating the risk of its members, but it was executed once per day, after market closing, and was not designed for historical risk and trend analysis.

There was also an Excel application, Caesar, built by my colleagues Rickard and Josefin, which could calculate risk for a specified portfolio. The account view of PowerPilot is based on the user interface of Caesar.

The Nord Pool batch system is still in use today, but PowerPilot is a secondary system for realtime and history analysis of risk patterns and intraday risk detection, e.g. for finding members that are vulnerable to price changes. It is also used for account-, instrument- and trade searches and for experimentation.

PowerPilot has been used daily by the Nord Pool risk analysis group since 2003. During this time the original design has been kept intact, despite a host of new requirements, including many new instrument types and performance optimizations.

1.3 Requirements

The initial requirements were of a general nature, and the development followed a prototype – reaction – prototype pattern. These were the initial requests:

- A user should be able to view summaries of member accounts. Information about these accounts, their status and trades can be found in a Nord Pool database. A user should also be able to compose and view combinations of accounts.
- The Nord Pool database design is primarily based on transactions. PowerPilot, on the other hand is more like a book-keeping system, so the Nord Pool information had to be translated into account information.
- Once a day, PowerPilot must download and parse new financial instrument definitions.
- PowerPilot must be able to handle realtime price updates. The prices are downloaded from a web site and parsed. The update delay should not exceed ten minutes. Realtime updates should be reflected in a recalculated user interface.
- A user must be able to view the historical values of all time-dependent information, such as prices, quantities and portfolio risks as diagrams or tables.

- PowerPilot must be usable both for market surveillance and experimentation, i.e. you should be able to easily switch from real values to altered values and back again.
- The server should continuously calculate a number of key indicators for the hundreds of member accounts. These key indicators can be compared or used as general warnings.
- The risk parameters are updated approximately once per month, and PowerPilot should keep track of the historical values. The mathematics may be altered, e.g. by the introduction of a new type of instrument. PowerPilot should keep track of all historical definitions as well as the current definitions.

1.4 This document

The contents of the remaining chapters are as follows:

Chapter 2 describes the fundamentals of Uninode®, a Java framework that has been used.

Chapter 3 describes the object model for financial instruments that has been used. This model is not limited to the power market.

Chapter 4 describes some risk analysis terminology.

Chapter 5 describes the PowerPilot software, with an overview of server tasks and the client user interface.

Chapter 6 contains general thoughts and reflections.

The actual content of the Nord Pool databases is not disclosed in this document. All data in this document is generated by the PowerPilot Simulation Tool, described in the PowerPilot chapter.

2. Uninode®

Uninode is a framework for handling a number of implementation details, such as data persistence and dependencies. It also implements a number of ideas and concepts that are crucial for understanding how PowerPilot is designed. This chapter is an overview of Uninode with as little implementation detail as possible.

Uninode was developed by me 2000-2002, with continuous improvement since then.

It is based on my experiences developing a Smalltalk system for technical analysis at Swedbank Markets. I used my own framework as the basis for PowerPilot since I wrote it for this kind of development environment, i.e. unknown future requirements, both regarding functionality and interface design, extensive use of mathematics, numerous types of diagrams, client/server communication and persistence, runtime logging selection and more.

One strategy for building a robust system is to understand and model reality and actual relations, instead of modeling the minimum set of required objects of algorithms, even though it may limit optimization. If you expect many future changes, my experience is that reality is less prone to change than any direct requirements of the specification.

The support and maintenance effort is reduced by a consistent framework with clear interfaces, almost regardless its complexity. Once the framework is debugged, the effort of implementing new features tends not to increase for each new feature.

The ideas and patterns listed below are original, in the sense that I have not seen them anywhere else, except for the semantic net (graph theory) and the evaluation environment (Lisp). The patterns are designed to interlock, but they can be implemented individually, if necessary.

Uninode® is a registered US trademark.

2.1 The Uninode Net

The Uninode framework implements the functionality of a typed semantic net, in which the types are themselves part of the semantic net. Each node in the net has an edge pointing at its type. Uninode supports multiple type inheritance and package structures. This type hierarchy is like a runtime layer on top of the Java class definitions. When I refer to ‘the Uninode net’ in this document, I mean this typed semantic net.

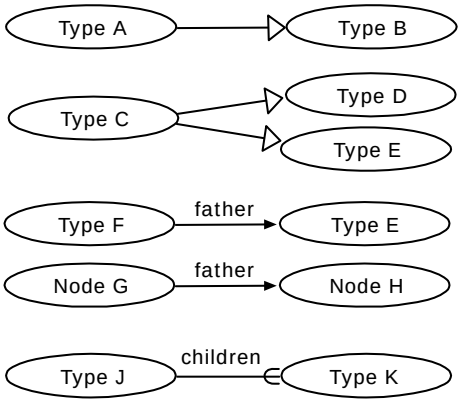
This may look like a standard object/attribute system, but the edges are more versatile than ordinary attributes. They may require arguments, in effect making them methods. Attributes can be considered to be methods without parameters. They can also use information from a thread-based environment, e.g. a historical date, to find a value. You can also specify *validity* and *estimation* requirements of a calculation or external access in the environment to suspend unnecessary communication and calculations. For instance, you can tell a data access object to load a value in the background instead of blocking, so that several accesses can be combined to speed up database communication. The Uninode edge also supports authorization and authentication at edge level. In addition, I have defined a scripting language called Edgescript, which can be used to define additional attributes and methods at runtime. Although complete with definitions and a compiler, Edgescript has never been used in PowerPilot.

Each node in the Uninode net can be accessed either using a Uninode id, which consists of a domain name and an identifier, or by following an edge from another node. Each domain has one or several Uninode servers handling node accesses and updates. A Uninode id is uniquely identifying at most one node or concept, just as most symbols have a unique Unicode id. The Uninode id is, of course, a Uniform Resource Identifier (URI).

There are hundreds of types already defined in the Uninode system. The basic type documentation can be found at http://www.uninode.org/.

The node and type diagram notation used in this document is very simple. It follows the normal rules of object orientation and inheritance.

A black arrow means ‘has a’. A white arrow means ‘is a’. A fork means ‘has a number of’.



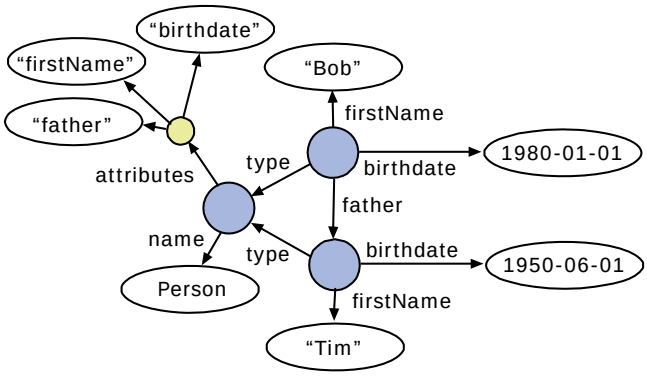
Uninode supports inheritance. A is a subtype of B, and B is a supertype of A.

Uninode supports multiple type inheritance. Type C is a subtype of types D and E.

A special notation is that instances of type F may have an edge called *father*, pointing at an instance of type E. This notation is also used for normal nodes. Node G has an edge called *father* pointing at Node H.

If node G is an instance of type F, node H must be an instance of type E or a subtype of E.

Each instance of type J has a list called *children* containing any number of instances of type K.



A net with two normal nodes and one type node.

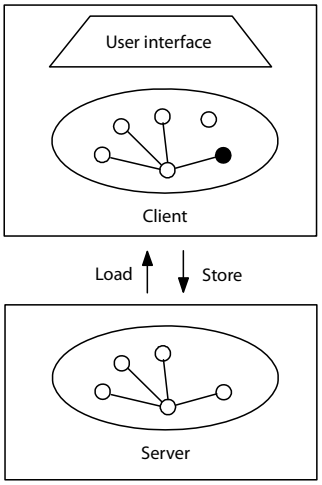
2.2 Dependencies

The Uninode net supports a registry of dependencies, i.e. the value of edge A of node B depends on the value of edge C of node D, and that edge A will be notified when edge C of node D is altered.

Individual updates do not necessarily trigger recalculations of the whole system, only the dependent values and fields. This is essential for a realtime system.

The dependency system makes it very easy to build Excel-like applications, especially when used with the special Uninode interface components described later in this chapter.

2.3 Local editing

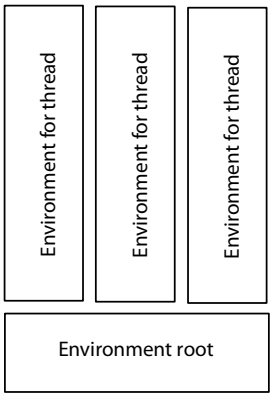


The Uninode net of each client is based on a global Uninode net, maintained by a set of servers. This net can be modified locally by each client, marked by the black dot in the diagram. For instance, the price of an instrument or the quantity of a possession can be changed in the local model to test the effects. This will not alter the server model, but it will affect all client calculations and views that depend on this value. For instance, if your user interface displays the value of an edge in two separate text fields, and you change the value in one text field, the other text field will be altered as well. You can also create new Uninode net nodes, marked by the solitary dot in the diagram, which can be evaluated in the Uninode environment.

The Uninode implementation allows you to subscribe to realtime updates. These updates, for instance the price of a contract, will change the client model and trigger update requests, but only if there are registered dependencies.

The updates trigger user interface components to repaint themselves, and it is the repainting that triggers new calculations. This means that hidden fields and tools do not trigger calculations until they are actually displayed.

2.4 The Uninode Environment



A Uninode environment contains information about the evaluation of the current thread, similar to the environment of Lisp. It keeps track of dimension values, the Edgescript evaluation stack, certain aspects of logging, the current user, the validity and estimation requirements of the calculation and much more.

A Uninode dimension is like an invisible parameter to any method evaluated in the thread. This is needed when you add methods as edges in the Uninode net, and the methods depend on the value of edges of other nodes. For instance, you can write a method that takes a number of edges as parameters, calculates some mathematical function with the values of these edges, and return the result. This method, e.g. a calculation of risk, is then added to some node in the net. At evaluation time, the method collects the values of the parameter edges, and returns the risk. The edge evaluations may depend on the dimension values of the environment. Thus, the risk method may depend on the historical date, the currency or some user settings that are not known when the risk method is written.

You can alter the dimension values to find changes in edge values as some dimension value changes. For instance, if you want the change of some edge value since yesterday, you just need to calculate the edge value with the current history dimension value, retrieve this current dimension value, subtract a day, push the new dimension value and ask for the edge value again, this time calculating the edge value the day before. Any pushed dimension value will automatically be popped when the method returns.

You can create a two-dimensional diagram by iterating over one dimension, keeping all other dimension values fixed, and plot the calculated edge value against the altered dimension value. You can create a three-dimensional diagram by iterating over two dimensions this way. If the edge value does not depend on the iterated dimension, you will get a straight, constant line.

PowerPilot has an environment root that contains the default dimension values. If the dimension value is not changed in a particular thread, the root value is used.

You can define your own dimensions. For instance, you can have a number of different ways to calculate a number. Your sources can use the selected value of your dimension to determine which method of calculation should be used. You can compare the different calculations by iterating over your the values of your method dimension to see how the method selection effect the outcome.

2.4.1 The history dimension

The history dimension is used for retrieving the value at a specific date from a timeseries. You can add span information, i.e. *from* and *to*, to speed up data retrieval. You can also add resolution information, i.e. day, minute or millisecond, to limit data access volume.

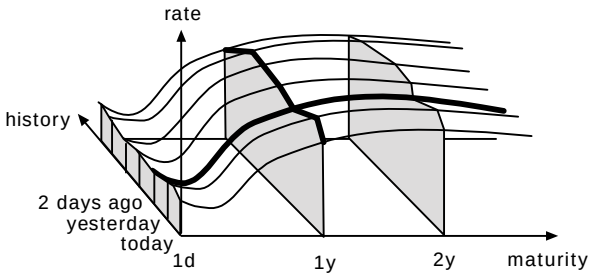
An empty value normally means the current time, although when editing a list of timeseries changes, it means the time of the Big Bang, i.e. negative infinity.

2.4.2 The language dimension

The language dimension may be used for determining the language of an output.

2.4.3 The maturity dimension

The maturity dimension is used for setting the span from one historical date to another. For instance, consider a zero-coupon yield. In its simplest form, it is an interpolation between a number of tbills and bonds, i.e. fixed income or rate instruments. All these instruments have a maturity date. The bonds also have a coupon rate, which is a percentage of its value that is paid every year until the maturity date.



The history and maturity dimensions of a yield

When you display a diagram, you fix all dimensions except one, and iterate over the values of the remaining one. In the figure above, you can either fix the history dimension (in the figure at 2 days ago), and draw a smooth curve by iterating over the maturity dimension values. This curve shows how the zero-coupon rate changes from overnight to the one-year rate at 1 year and so on.

You can also fix the maturity dimension value (in the figure at 1 year in the future), and iterate over the historical dates. This will show how the one-year zero-coupon rate has changed the last days.

You can view the problem as a three-dimensional space, where you can slice either into the figure (fixed maturity) or along the screen (fixed history), depending on what you want to see.

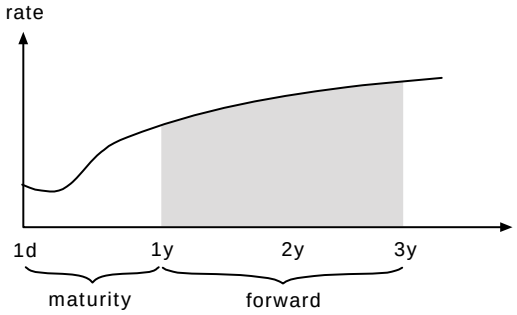
Note that the actual date associated with the 1 year maturity value depends on the value of the history dimension. Today it is one year from today. Yesterday it was one year from yesterday. To achieve a fixed maturity date, you get a diagonal line pointing from the front left to the back right. For instance, 1 year maturity today was 1 year and 1 day maturity yesterday.

It looks like the maturity would be continuous, but you only count distinct days. To make it even more complicated, the maturity is divided into 365 parts up to one year, and 360 parts per year after that. This is how the fixed income market is defined. The minimum value of the maturity dimension is 1 day, i.e. overnight. The value of the yield curve at 1 day maturity is the same as the overnight rate.

2.4.4 The forward dimension

Rate can be interpreted as a measure of the value of delaying payment for some specified time. If you know the value of delaying payment until tomorrow, and the value of delaying it until the day after, i.e. two days from now, you can calculate the value of delaying it one day, starting tomorrow, based on the difference between the one and two day rates. Using this method, you can calculate the market prediction of a future yield curve based on the yield curve today.

The forward dimension is used for predicting what a yield curve says about future rates.



The maturity and forward dimensions

In the figure, you want to determine what a yield is saying about the 2-year rate one year from now. Fixing the history and maturity dimensions, and iterating over the forward dimension will show a curve that resembles a normal yield curve in function. For instance, if you fix the maturity dimension to 1 day, and iterate over the forward dimension, you will get a prediction of the yield tomorrow. If you fix the forward dimension to 1 day, and iterate over the maturity dimension, you will see the predicted overnight rate.

2.5 Timeseries

A timeseries is an edge whose value depends on the time. It can have only one value per given time. This value may be null, which means that it is undefined. The value of the history dimension of the environment is the time used to determine the value of the timeseries.

Normally, the timeseries is defined by changes at specific times. A value is valid until it is replaced by next value. For instance, if the market value of an equity share is 70 at 12:00 and next change is to 71 at 12:02, the value of the timeseries is 70 at 12:01. Other timeseries are possible, where the value is a mathematical function, such as a linear or spline function of the time. For instance, a *daysRemaining* timeseries changes once every day.

The value of a timeseries is not restricted to numbers. A timeseries can represent the current address of a person. When the person changes address, the new address is added at the time for the change, instead of simply replacing the old address. The address will now depend on the history value of the environment.

A timeseries is accessed in its entirety, using the history dimension *from* and *to* values, or only in parts. You can also specify the resolution of a timeseries, using the history dimension *resolution* value. For instance, you can limit the value to one per day, e.g. the closing prices, or you can request all five minute intraday updates.

When you decide that the value of an edge varies with time, you must decide if you want it to be a timeseries that return different nodes, or if it should always return one node, possibly a singleton, whose attributes are timeseries. For instance, the PowerPilot risk model parameter node is a singleton, with parameters that are timeseries. I have implemented it this way since most parameters changes once a year, while few changes once a month. If all parameters were altered at the same time, I would use one separate parameter node for each new change-date, since it would be easier to see which parameters belong to each other.

2.6 User interface components

The Uninode framework provides a number of components based on Java Swing for editing the Uninode net.

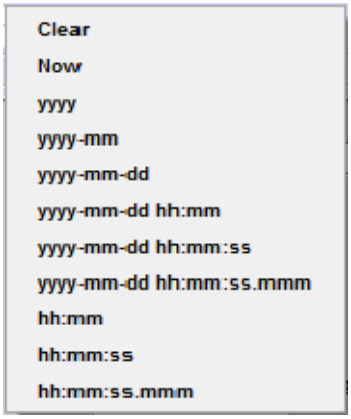
2.6.1 Datetime editor

A datetime editor is used for selecting a date and time.

If the fields are empty, the current time of any request is used.

The first up-arrow button will change the date one day into the future, while the down-arrow button will change the date one day into the past.

The second up-arrow button will change the minute to the last minute of the day (23.59), while the down-arrow button will change the date to the first minute of the day (00.00).



The right-arrow button will display the popup menu.

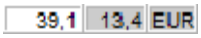
The Clear popup choice will clear the selection. This is the same as the current time at any request.

The Now popup choice will fix the date to the current time.

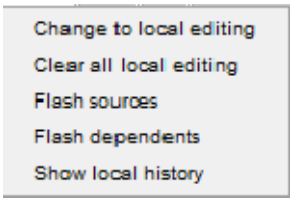
The other popup choices will change the resolution, altering the number of fields shown in the editor. All Uninode timestamps have a resolution, e.g. day, minute, second or millisecond. Oddly enough, when you request a value for a date, you normally mean the last value that day, i.e. the closing value, while all other resolutions mean the value of that time or earlier.

If you write in a field when all other fields are empty, the system will fill in the blanks with the current day and time. For instance, a fast way to enter 11.00 today into an empty time widget is to type 11 in the hour field, press tab, which will fill in the current date in the year, month and day fields, type 0 and press tab.

2.6.2 Input field



The input field is used for entering text and numbers, editing the Uninode net. The input field needs to know which node to use, and which attribute of that node to display. Note that most fields require input according to the current locale (e.g. Sweden), so the decimal separator may be a comma instead of a dot. If you have permission to do so, the new value in the field will be communicated to the responsible Uninode server, and update the global Uninode net.



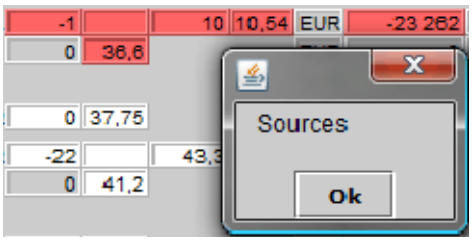
If you right-click on an input field, a popup menu may show up, depending on the function of the input field.

If you select *Change to local editing*, the background color will change to blue, and all changes of the value will stay local, i.e. editable and changes not stored on any server. The popup menu of a local editing field will have the menu choice *Change to global editing* instead, to make the value equal to the global value again. Note that if you select a local editing of a timeseries, it ceases to be a timeseries, and becomes a constant, time independent value. This may be a problem if you construct a risk model that depends on the price yesterday as well as the current price, since you edit both values at the same time. The solution is to have one timeseries attribute for the current price and another for the price the day before.

Note that the value changes in the local Uninode net, so any calculations will be affected until you turn off the local editing. If the value is displayed in any other field, that field will become blue as well.

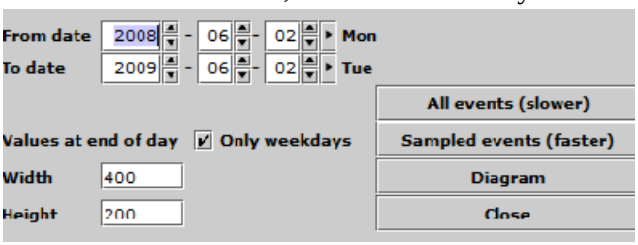
The *Clear all local editing* will remove any local changes of any values in the local Uninode net, resetting the net to its global value.

If you select *Flash sources*, all fields that the value of this field depends on will change background color to red until you press the *Ok* button on the *Sources* dialog. This functionality relies on the Uninode dependency model.



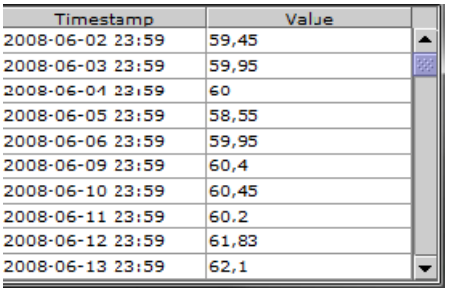
If you select *Flash dependents*, all fields that are dependent on the value of this field will flash.

The *Show local history* is displayed only if the value of the field is a timeseries. Normally, the field displays the value at the time defined in the environment root, but the local history selection is used for displaying the value at a range of timestamps. It is called local history, since it uses any local editing for calculating the values. The local history of a timeseries is the same as the global history if you do not have any local edits.



You can select a start- and an enddate, and if the sampled values and the diagram should include weekends or not.

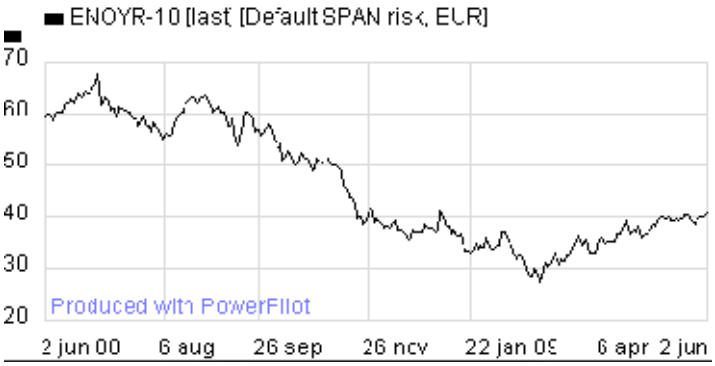
If you want a list of values, e.g. for copying to Excel, you can select if you want sampled values, i.e. values at fixed intervals, or all changes.



Timestamp	Value
2008-06-02 23:59	59,45
2008-06-03 23:59	59,95
2008-06-04 23:59	60
2008-06-05 23:59	58,55
2008-06-06 23:59	59,95
2008-06-09 23:59	60,4
2008-06-10 23:59	60,45
2008-06-11 23:59	60,2
2008-06-12 23:59	61,83
2008-06-13 23:59	62,1

If you want to display a diagram, which always uses sampled values, you can determine the initial size, although it can be resized manually.

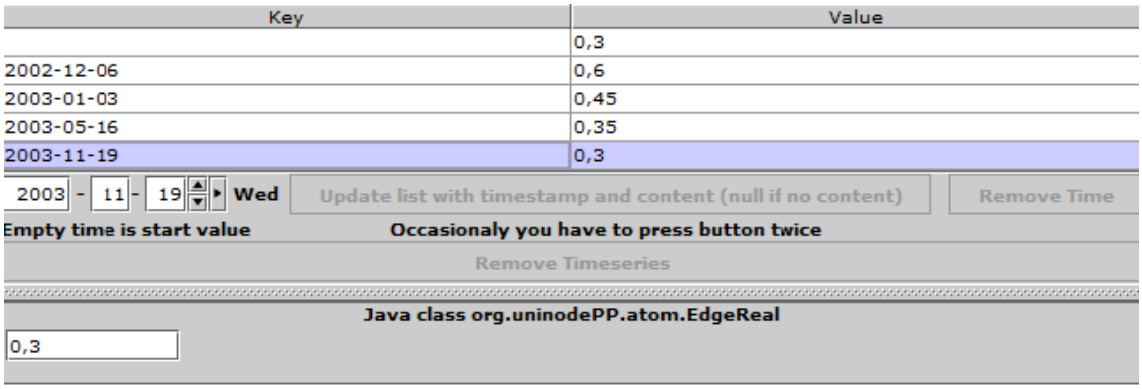
The diagram displays the node, in this case ENOYR-10, the edge (last traded value), as well as the environment root dimension values (risk model and currency).



2.6.3 Diagram widget

Some values in the Uninode net are editable timeseries. These timeseries values can be manipulated using a diagram widget. The widget button, displaying a diagram icon, is red if the value currently depends on the time and blue if it does not.

When you press the diagram button, a dialog will be displayed.



The dialog displays a number of ordered time-value pairs. The value of each row is valid until the time of the next row. The first row may lack a time value, making it valid since the Big Bang.

To add a new row, you enter the time using the time widget and a value, and press the ‘Update’ button. To remove a row, you select the row, and press the ‘Remove Time’ button.

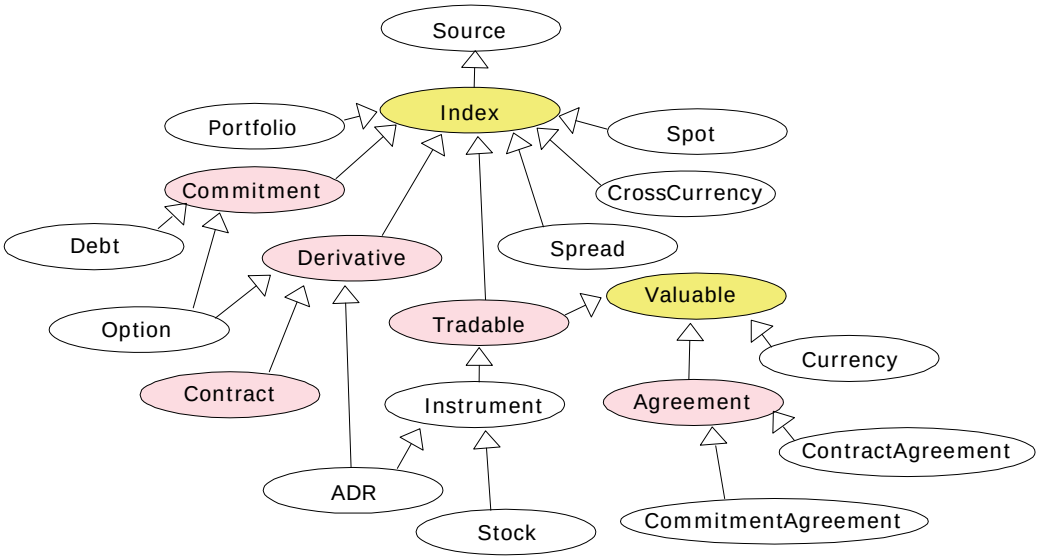
If there is only one value remaining, with no time, it will be converted to that fixed value in the Uninode net.

Note that all changes are applied to the global Uninode net as well as the local.

3. The Financial Instrument Model

This model of the financial market was developed by me during my time at Swedbank Markets, through analysis and conversations, and refined for PowerPilot. It was developed for technical analysis of instruments, but it is valid for many aspects of financial analysis.

Recall that a white arrow means 'is a', a black arrow means 'has a' and a fork means 'has a number of'.

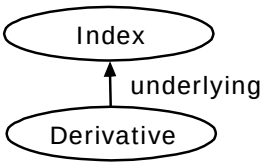


3.1 Index

An index is something measurable. It has some measured value, which normally changes with time. One example of a pure index is the weighted average price of the most valued stocks of a market, e.g. the OMX Stockholm or NASDAQ indices.

3.2 Derivative

A derivative is an index, so it has a value. It also has one or several underlying indices. The value of the underlying indices can be used for theoretical calculations of the value of the derivative itself. A difference between the market value and the theoretical value indicate either a foul market pricing, which can be exploited, or a bad theoretical model.



3.3 Valuable and Tradable

A Valuable is an asset that can be sold and bought.

A Tradable, e.g. an equity share like Ericsson B, can be both measured and traded. The measured value is normally the last price paid for the share, using the currency of the equity. An overnight rate, on the other hand, is just an index that can be used indirectly, often for derivative valuation, but it can not be traded.

3.4 Commitment

A commitment is when the issuer sells a commitment to do something in the future. A debt is when the borrower sells a commitment to pay back a sum of money with interest.

3.5 Option

An option can be described in many different ways. In this model, it is an obligation to do something in the future, e.g. buy or sell something at a certain price.

Another way to view an option is as an insurance. You can buy an option to ensure that you will be able to buy or sell something at a specified minimum or maximum value.

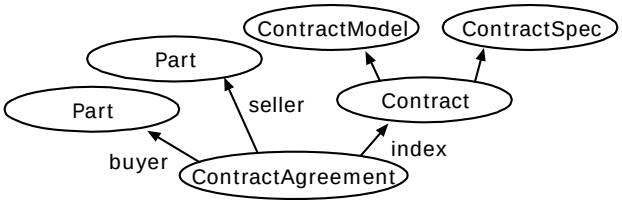
Another way to view an option is purely mathematical. You can buy options to reduce your financial risk measurements, e.g. delta, theta and gamma, to make you less exposed to price variations.

There are two types of options; call and put. A call option gives the holder (buyer) the right to buy the underlying asset by a certain date, the maturity date, for a certain price, the strike price. A put option gives the holder the right to sell the underlying asset by a certain date for a certain price. (Hull, 1997)

Thus, the holder buys an option from an issuer at a certain price, and may or may not exercise the right to buy or sell. The issuer then has a potential expense at the maturity date, equal to the difference between the price at the maturity and the strike price. This is the financial risk for the issuer.

3.6 Contract

In case of a contract, both the buyer and the seller are bound by the contract during its lifetime. The contract model determines the rules, e.g. forward or future, while the contract spec determines delivery period and such.



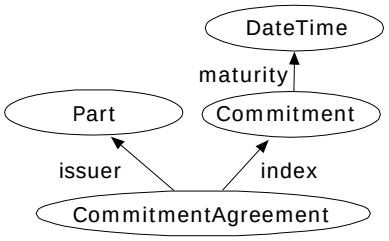
A contract is an obligation by two parts to do something during the lifetime of the contract.

A power contract can be bought or sold to ensure that you will be able to receive or deliver power in the future at a specified price.

3.7 Agreement

Note that neither a contract nor a commitment can be traded as such, since they do not contain any information about the issuer, buyer or seller. An exception is the debt, when the issuer is fundamental for the index value; a more solid issuer is less likely to default, and can thus pay a lower interest rate.

In case of a commitment, a commitment agreement is used. It is sold by the issuer to someone. The issuer is obligated to perform some task in the future. In the case of an option, the issuer must buy or sell at a certain price, if the buyer so wishes. In the case of a debt, the issuer must pay the buyer coupons during the lifetime of the debt as well as full price at the maturity date.



As in the case of commitments, a contract agreement uses a contract for valuation, even though the contract is not bound to any particular buyer or seller.

Note that the issuer of a commitment agreement using a debt as index usually is the same as the issuer of the debt, e.g. a bank or mortgage institution.

A Part is a person or company trading on the market.

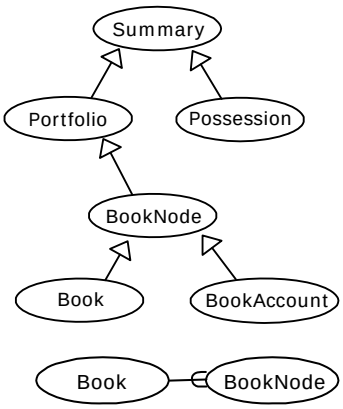
3.8 Ticker

A ticker is a collection of all instruments with the same ticker id, e.g. FWV2-02. It will select which instrument is valid at a certain time. Most tickers, including almost all option tickers, have the same reference instrument during its lifetime. Contract tickers in delivery may change underlying instrument every week.

3.9 Portfolio and book

A portfolio is a summary of a number of possessions and subportfolios.

The most basic type is Summary. It can keep track of various measures of profit, as well as accumulated fees.

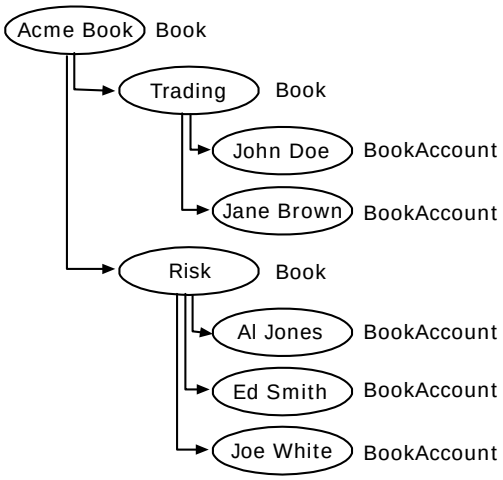


Both the Portfolio and the Possession inherits all the functionality of Summary, but each type has some extra functionality. The possession knows its instrument, quantity and counterpart, while the portfolio knows its possessions and subportfolios.

A portfolio by itself does not have to represent any reality, and a possession may be part of several different portfolios. You can, for instance, construct a risk analysis portfolio for all possessions with a certain counterpart, to evaluate your risk toward that specific company. There is, however, a special kind of portfolio, called BookAccount, which is used for keeping track of actual trades.

A Book is constructed like a tree, with a Book node at the root and BookAccount nodes as leaves.

The figure shows a sample book of the Acme company. It has two departments, Trading and Risk. When the profit and risk for the Trading department is calculated, it uses all positions of all its BookAccounts. Since John Doe may have a possession that is opposite to that of Jane Brown, the total risk of the Trading department is usually less than the sum of the risks of the individual trader. The same is true for the company risk compared to the sum of the risk of the individual departments.



3.10 Depot

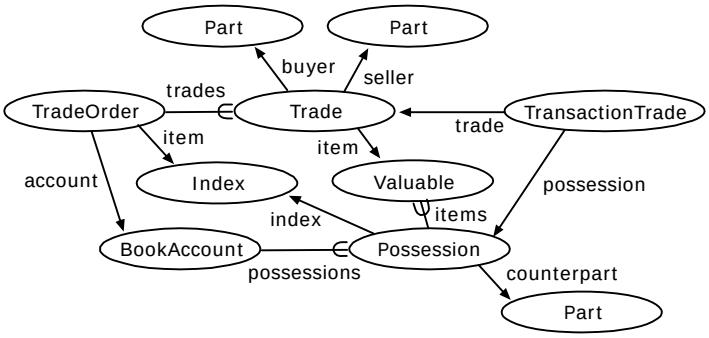
A Depot is like a vault, managed by a manager, in which an owner keeps valuables. These valuables can be used as security for trades, e.g. if the owner sells an option to the manager.

Since Nord Pool uses all member possessions as securities for the remaining possessions of that member, the depot abstraction is not used in PowerPilot.

3.11 Market

A Market is a list of instruments.

3.12 Trade



A TradeOrder is used to register an intention by a part to make a trade.

A Trade node keeps track of some agreement between two parts.

A TransactionTrade is used for associating a Trade with a possession in a BookAccount.

In the figure you can see the difference between an index and a valuable.

If you trade a tradable, like an equity share, the index and the valuable is the same node, i.e. the only item in the items list of the possession is the same node as its index.

In case of a contract or a commitment, on the other hand, you may place the order before you know who your counterpart is, and thus before you know which agreement (valuable) you will trade. Once the trade is executed, the correct agreement is found or created. In case of a contract, the contract agreement will have you as one part and your counterpart as the other. In case of a commitment, the issuer is you if you sell, and your counterpart if you buy. The index of the possession will point at the contract or commitment. The items list of the possession will contain both all agreements where you are the buyer and your counterpart the seller and all agreements where the roles are reversed.

When you place a TradeOrder, it must be associated with exactly one BookAccount. Thus no trade can be associated with two different BookAccounts. This trade will automatically change some timeseries associated with its BookAccount.

4. Risk Analysis

Financial risk is a measurement of expected change, for better or worse; the art of guessing the amount of money that can be lost or gained tomorrow due to changes in price and time.

Generally, a trader does not know whether a price will go up or down, but the best portfolio is one where you earn money regardless of what happens. Then the risk is minimal. Most serious traders are willing to reduce their expected profit (or return) in exchange for reduced risk. One way to reduce risk is to buy options.

Many traders use risk analysis as an argument for taking trading risks. It is important not to confuse a mathematical formula with the actual risk. The formula uses historical changes as input, as if the financial markets were a physical system. They are not. The formulas also do not consider psychological effects, such as group behavior. However, the formulas are often the best known approximations.

There are other risks than financial risks, of course. For instance, a counterpart of an agreement may default, and be unable to fulfill his part at all. This is called the credit risk.

Nord Pool is a counterpart to all its trading, inserting itself between the two trading parts, and takes a commission for that. This means that the member only has to be concerned with Nord Pool defaulting and the Nordic governments not saving the power market.

If a member company defaults, i.e. cannot pay its bills, Nord Pool assumes the debts and contracts of a defaulted member, as well as all securities. This may result in future losses, so the Nord Pool member companies have to have a surety, e.g. a bank, to guarantee that Nord Pool has access to an amount that should cover the losses. Just as a member moves the credit risk from the counterpart to Nord Pool, Nord Pool moves the credit risk from the member to its surety, which should be considered more solid.

The PowerPilot program is used for calculating the limits of future losses of the member companies, which is a measure of the risk profile of the members.

4.1 Scenario risk

Nord Pool uses a scenario-based model in which the maximum loss is evaluated for a number of scenarios, and the worst outcome is chosen. Time-dependent risk parameters that can be altered in the scenarios include price-, volatility- and currency changes.

Volatility is a measurement of expected change, similar to the standard deviation of an index. It is used mainly for calculating the price of options.

The mathematics for calculating these scenario risks are quite complex. Say, for instance, that the price, volatility and currency can go up, down or be still. This is three to the power of three, or 27 scenarios per instrument.

Since the instrument risks are dependent on each other, a change of one instrument may counter the risk of another, so you can normally not just add the risks. Theoretically, you have to consider every possible combination of change. Calculating the scenarios for a portfolio of ten instruments means 27^{10} scenarios. Most members have a lot more than ten possessions. There are also more than three scenarios per parameter.

The solution is to use groups of correlated instruments, whose scenarios behave similarly, to reduce the number of scenarios, and add the risk of each such risk group. The actual risk analysis algorithm is proprietary to Nord Pool, so it will not be covered here in more detail than this.

Since the surety will charge more for guaranteeing a larger amount, a member will want this risk evaluation to be as low as possible, while still covering the expected maximum loss. It is, after all, money not ending up at Nord Pool, so the formula needs to return the minimum amount that is acceptable.

The risk calculation requires a number of parameters, such as current rate, currency exchange loss, scenario changes, correlation- and volatility matrices and more. Since these parameters are altered approximately once per month, they are stored as timeseries edges of a singleton RiskScenarioParameters node, with the Uninode id *com.powerpilot.5000*.

4.2 Liquidation value

This is the value of a possession if it was to be closed at the current time and price, adjusted for currency exchange losses. It is essentially the value of the securities.

4.3 Margin Call

The margin call is the scenario risk minus the liquidation value, i.e. the net risk, capped by zero. This is the amount that the surety of a member must guarantee.

4.4 The Greeks

There are a number of less complex risk measurements that can be used for many kinds of financial instruments. (Hull, 1997)

4.4.1 Delta

Delta is the rate of change of its price with respect to the price of the underlying asset. You can calculate delta for an individual instrument, but PowerPilot displays the possession delta, i.e. the instrument delta multiplied by the number of contracts or options.

Energy delta is the delta multiplied by the number of MWh.

4.4.2 Gamma

Gamma is the rate of change of the portfolio's delta with respect to the price of the underlying asset. As with delta, PowerPilot displays the possession gamma, not the instrument gamma.

Energy gamma is the gamma multiplied by the number of MWh.

4.4.3 Theta

Theta is the rate of change of the value of the portfolio with respect to time with all else remaining the same. The closer you get to the maturity date of an option, the less uncertainty is involved.

4.4.4 Vega

Vega is the rate of change of the value of the portfolio with respect to the volatility of the underlying asset.

4.4.5 Rho

Rho is the rate of change of the portfolio with respect to the interest rate.

5. PowerPilot

The PowerPilot system consists of a PowerPilot server, a dedicated database server, a number of PowerPilot clients, and a number of external data sources.

5.1 Dimensions

In addition to the dimensions mentioned in the Uninode chapter, PowerPilot uses a few special dimensions. The values are set in the environment root, but they may be altered during the evaluation of each thread.

5.1.1 The currency dimension

The currency dimension can be used for determining the currency requested in a calculation. Some edges, e.g. a currency node, may perform a conversion from the native currency to the requested currency at evaluation time. This conversion may include exchange costs.

5.1.2 The rate dimension

The rate dimension contains an indicator of how much you value delayed payments. Its value is the overnight rate.

5.1.3 The risk model dimension

This dimension contains the selected risk model.

5.2 PowerPilot Server

The main task of the PowerPilot server is to update the dedicated database with instrument definitions, account information and risk calculations.

The PowerPilot server is a Uninode server, with special hooks for PowerPilot-specific tasks.

5.2.1 Instrument definitions

The Nord Pool web site contains a number of text files, called SPAN-files, containing the instrument definitions for risk calculations the next day. There is another set of files defining the trade dates for various contract and options. The PowerPilot server periodically polls the website for new definitions and trade dates.

5.2.2 Realtime prices

The Nord Pool web site also continuously updates a text file with option- and contract prices. The server parses this file, and adds new prices to the instrument price timeseries.

5.2.3 Database duplication

The Nord Pool database design is based on transactions, i.e. when who did what, and as a storage for account calculations. The databases lack good indices for ad hoc queries.

This is one reason for continuously copying the external database content to local storage, with indices for the SQL queries the PowerPilot server uses. Two other reasons are easier debugging and better database load control.

5.2.4 Possession timeseries calculations

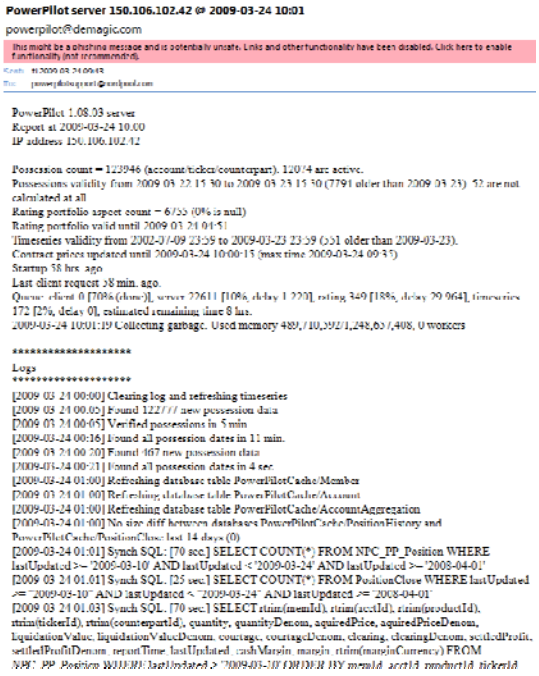
Normally possessions are updated once per day, and trades continuously during the day. This process calculates and updates the possession timeseries in the database, e.g. the quantity and liquidation value timeseries.

5.2.5 Rating series calculation

There are a number of indicators that are continuously calculated for each member account by the server, so that a PowerPilot user easily can compare or filter members. Such indicators may include recent losses or high risk.

5.2.6 Status report

If the server detects an error, e.g. a new, unexpected file format, a failing external database or a memory failure due to excess caching, it will automatically restart, although never less than three hours after last restart. Once per day, at 10 am, and two hours after each restart, it will send an email to a mailing list, with a report on its current status, including possession calculations, price validity, client usage and error messages. The status report contains no secret information.


PowerPilot server 150.106.102.42 @ 2009-03-24 10:01

powerpilot@demagic.com

this might be a phishing message and is potentially unsafe. Links and other functionality have been disabled. Click here to enable functionality (not recommended).

PowerPilot 1.08.03 server
Report at 2009-03-24 10:00
IP address 150.106.102.42

Possession count = 123946 (account/ticker/counterpart). 120/4 are active.
Possessions validity from 2009-03-22 15:30 to 2009-03-23 15:30 (7791 older than 2009-03-23). 52 are not calculated at all
Rating portfolio aspect count = 6/55 (0% is null)
Rating portfolio valid until 2009-03-24 04:51
Timeseries validity from 2002-07-09 23:59 to 2009-03-23 23:59 (551 older than 2009-03-23).
Contract prices updated until 2009-03-24 10:00:15 (max time 2009-03-24 09:35)
Startup 38 hrs. ago
Last client request 38 min. ago.
Queue: client 0 [70% (done)], server 22611 [10%, delay 1.220], rating 349 [18%, delay 29.964], timeseries 172 [2%, delay 0], estimated remaining time 8 hrs.
2009-03-24 10:01:19 Collecting garbage. Used memory 489,710,592/1,248,63 /,408, 0 workers

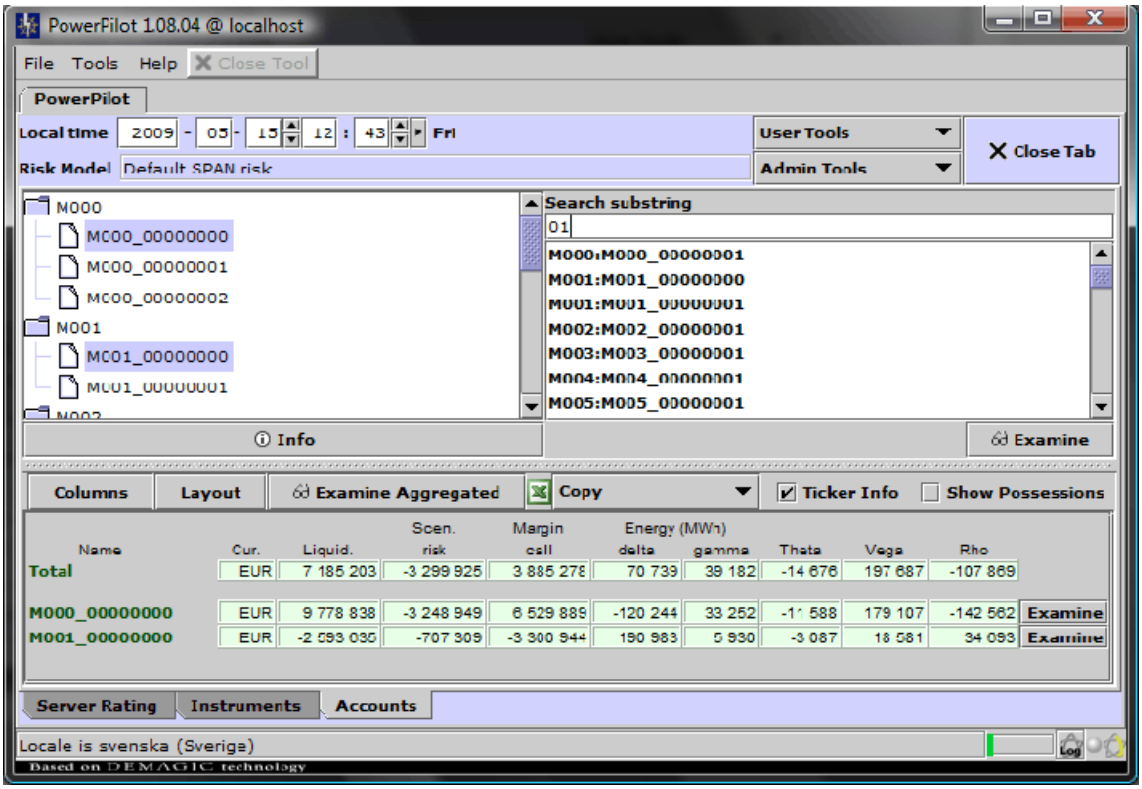
****
Logs
****
[2009-03-24 00:00] Clearing log and refreshing timeseries
[2009-03-24 00:05] Found 122/// new possession data
[2009-03-24 00:05] Verified possessions in 5 min
[2009-03-24 00:16] Found all possession dates in 11 min.
[2009-03-24 00:20] Found 467 new possession data
[2009-03-24 00:21] Found all possession dates in 4 sec
[2009-03-24 01:00] Refreshing database table PowerPilotCache/Member
[2009-03-24 01:00] Refreshing database table PowerPilotCache/Account
[2009-03-24 01:00] Refreshing database table PowerPilotCache/AccountAggregation
[2009-03-24 01:00] No size diff between database PowerPilotCache/PositionHistory and PowerPilotCache/PositionClose last 14 days (0)
[2009-03-24 01:01] Synch SQL: [70 sec.] SELECT COUNT(*) FROM NFC_PP_Position WHERE lastUpdated >= '2005-03-10' AND lastUpdated < '2009-03-24' AND lastUpdated >= '2008-04-01'
[2009-03-24 01:01] Synch SQL. [25 sec.] SELECT COUNT(*) FROM PositionClose WHERE lastUpdated >= '2009-03-10' AND lastUpdated < '2009-03-24' AND lastUpdated >= '2008-04-01'
[2009-03-24 01:03] Synch SQL. [70 sec.] SELECT rtrim(memId), rtrim(accId), rtrim(productId), rtrim(tickerId), rtrim(counterpartId), quantity, quantityDenom, aquiredPrice, aquiredPriceDenom, liquidationValue, liquidationValueDenom, courtage, courtageDenom, clearing, clearingDenom, settledProfit, settledProfitDenom, reportTime, lastUpdated, cashMargin, margin, rtrim(marginCurrency) FROM
NFC_PP_Position WHERE lastUpdated > '2009-03-10 00:00:0 ORDER BY memId, accId, productId, tickerId

If the server detects an error, e.g. a new, unexpected file format, a failing external database or a memory failure due to excess caching, it will automatically restart, although never less than three hours after last restart. Once per day, at 10 am, and two hours after each restart, it will send an email to a mailing list, with a report on its current status, including possession calculations, price validity, client usage and error messages. The status report contains no secret information.

Since PowerPilot is a secondary system, relying on the output of the primary system, it is used for validating output from the primary. If there are errors, e.g. formatting errors, the PowerPilot server will immediately send an email to the support group. Although it is not implemented, it could also validate the content of external databases, e.g. matching trades with possession updates.

5.3 PowerPilot Client

5.3.1 The Main Window



When the PowerPilot client is started, the user is prompted for a password. The client then loads information from the PowerPilot database and server, and opens the main window. The user can open different tools for different tasks.

New tools are opened using the User or Admin Tools dropdown, and accessed using main window tabs. At startup, the main window has three open tools: Server Rating, Instrument and Accounts.

The local time, located at the top left corner, is used for most calculations of the PowerPilot tools. If the fields are empty, the most recent values are used. The main window also has a field for the current local risk model. You can change this value by opening a Risk Model tool. Both these values are used by the local environment root.

The User Tools drop down menu is used for opening tools (tabs) that normal users often use, while the Admin Tools drop down menu is used for opening less frequent tools.

At the bottom, there is a field for system messages, a load indicator, which will become red if the network is down, the server is down or very busy, or the client is very busy, a communications indicator that lights up when the client is communicating with the server and a progress indicator is animated when some task is in progress.

The log button in the bottom right corner opens a log window. It is used for debugging.

5.3.2 Accounts Tool

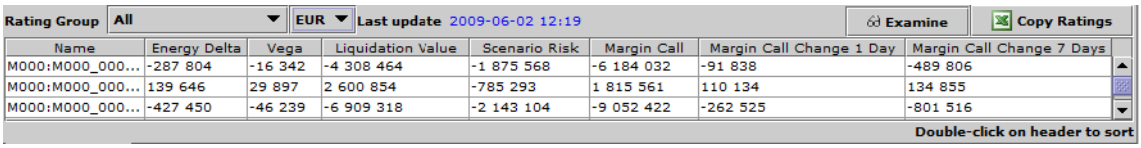
The Accounts tool, shown in the main window image above, is the main PowerPilot tool, used for finding accounts and opening tools for individual accounts.

The account tree displays all available accounts. The top level contains brokers, the next member companies, if different from the broker, and the lowest contains the accounts. You can also use the search field to search for account name substrings.

You can aggregate, i.e. combine, any number of accounts using Control-click or Shift-click to select individual accounts or account ranges.

Use an 'Examine' button to open an Account Tool for the selection.

5.3.3 Server Rating Tool



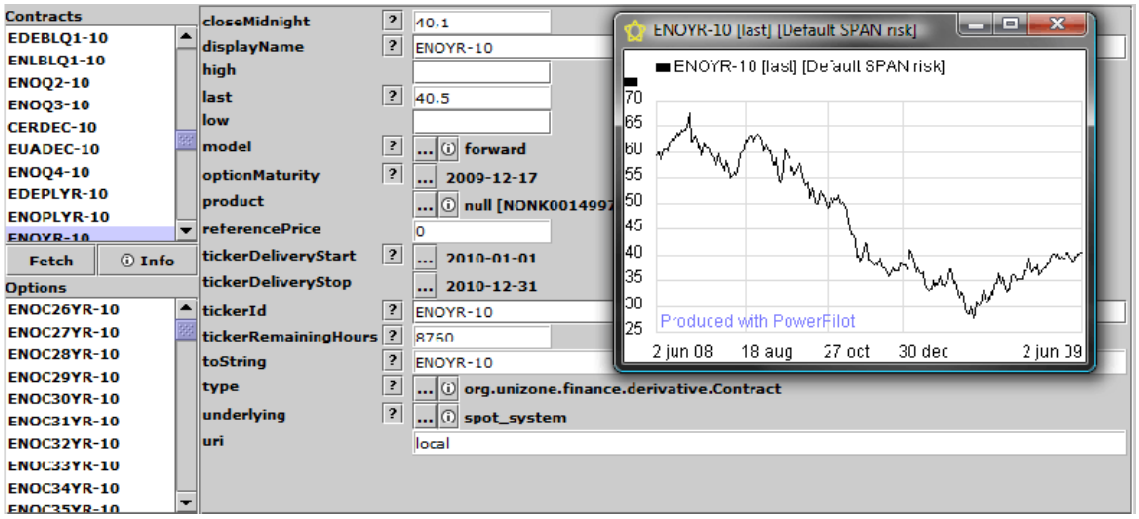
The server continuously calculates certain indicators for most accounts, using both NOK and EUR as currency for all calculations.

You can select one rating group, and the precalculated rating values are displayed.

The timestamp of the rating is lagging by at least half an hour, partly because the rating values take time to calculate, partly so that price updates do not change the portfolio values.

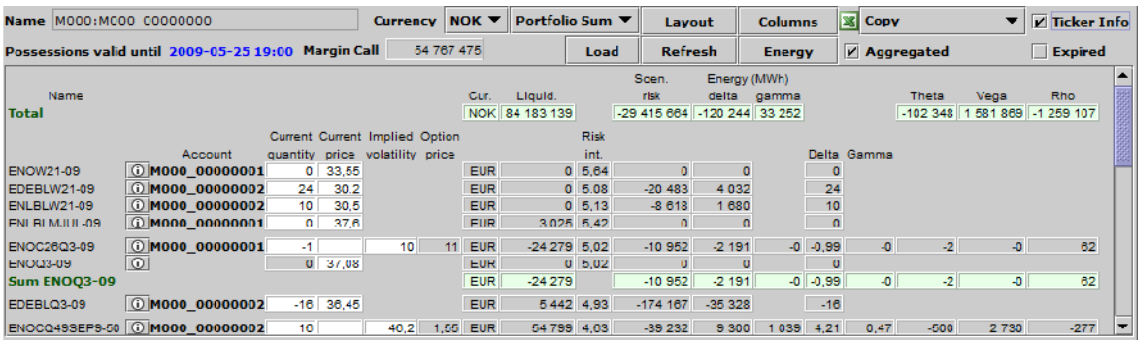
Note that these values are precalculated by the server. You can use the Local Rating Tool to calculate the values for altered parameters, e.g. time, currency or risk model.

5.3.4 Instrument Tool



The Instrument tool is used for finding and viewing aspects of the Nord Pool instruments. Most values displayed are timeseries, so you can right-click to view the history.

5.3.5 Account Tool



The account tool is used for viewing a portfolio. If you use the datetime selector of the main window to set a date, the values from that date will be used. Otherwise, the realtime value will be used.

Some instruments are grouped together for viewing instrument-specific risk.

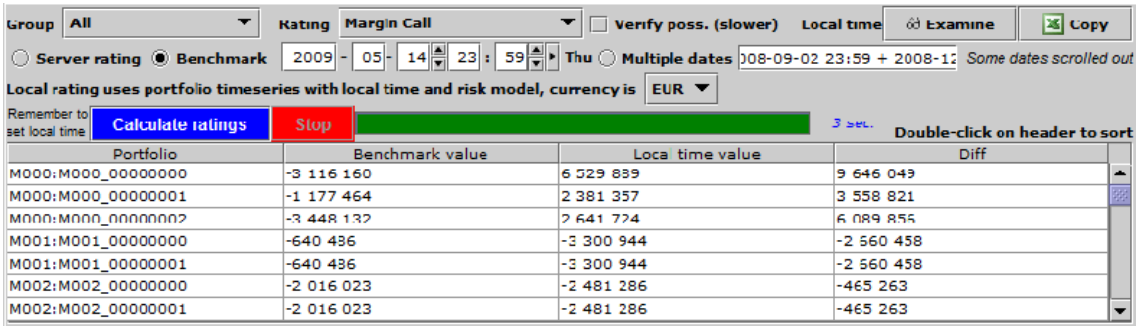
You can display values in various currencies. You can select to calculate either just the portfolio sum for this currency, using the native currency of the instruments for the individual instrument rows, or use the selected currency for all calculations and displays.

You can select which columns are displayed.

You can edit the values locally to experiment, using Local Editing.

You can show the energy distribution of the portfolio, i.e. the number of MWh, positive or negative, that the contracts and options of this portfolio cover.

5.3.6 Local Rating Tool

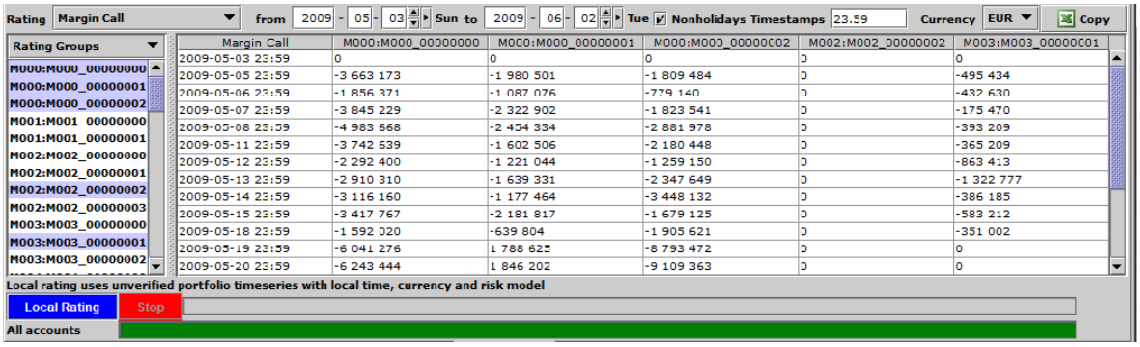


Portfolio	Benchmark value	Loca time value	Diff
M000:M000_00000000	-3 116 160	6 529 839	9 646 049
M000:M000_00000001	-1 177 464	2 381 357	3 558 821
M000:M000_00000002	-3 448 132	2 641 724	6 089 855
M001:M001_00000000	-640 486	-3 300 944	-2 560 458
M001:M001_00000001	-640 486	-3 300 944	-2 560 458
M002:M002_00000000	-2 016 023	-2 481 286	-465 263
M002:M002_00000001	-2 016 023	-2 481 286	-465 263

This tool is similar to the Server Rating Tool, but you can use local values instead. It is more versatile, but considerably slower than the server rating.

You can compare a local value with the value for another time, calculate values for a range of dates, or compare it to the server value.

5.3.7 Rating Series Tool

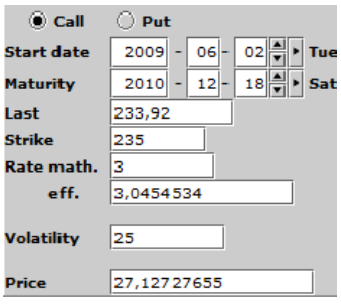


Margin Call	M000:M000_00000000	M000:M000_00000001	M000:M000_00000002	M002:M002_00000002	M003:M003_00000001
2009-05-03 23:59	0	0	0	0	0
2009-05-05 23:59	-3 663 173	-1 980 501	-1 809 484	0	-495 434
2009-05-06 23:59	-1 856 371	-1 087 076	-779 140	0	-432 630
2009-05-07 23:59	-3 845 229	-2 322 902	-1 823 541	0	-175 470
2009-05-08 23:59	-4 983 568	-2 454 334	-2 881 978	0	-393 209
2009-05-11 23:59	-3 742 539	-1 602 506	-2 180 448	0	-365 209
2009-05-12 23:59	-2 292 400	-1 221 044	-1 259 150	0	-863 413
2009-05-13 23:59	-2 910 310	-1 639 331	-2 347 649	0	-1 322 777
2009-05-14 23:59	-3 116 160	-1 177 464	-3 448 132	0	-386 185
2009-05-15 23:59	-3 417 767	-2 181 817	-1 679 125	0	-583 212
2009-05-18 23:59	-1 592 020	-639 804	-1 905 621	0	-351 002
2009-05-19 23:59	-6 041 276	1 788 625	-8 793 472	0	0
2009-05-20 23:59	-6 243 444	1 846 202	-9 109 363	0	0

This tool is similar to Local Rating, but you can calculate a timeseries of values, e.g. for copying to Excel for diagrams.

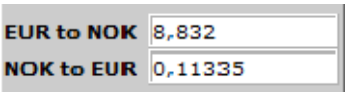
5.3.8 Black76 Tool

This tool is used for calculating option price or volatility. Changing any field except the price will set the new price. Changing the price will calculate the volatility.

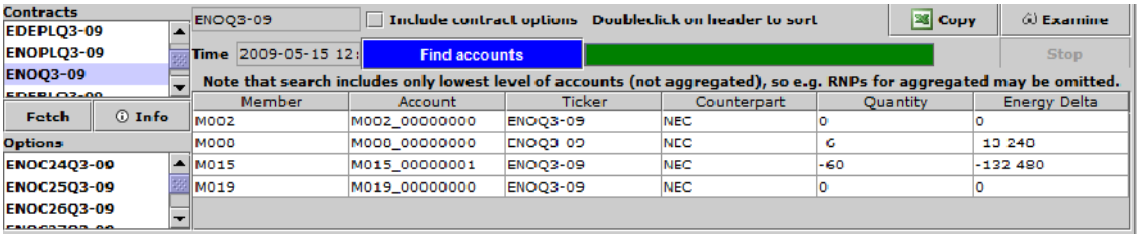


5.3.9 Currency Tool

Here you can view and edit the exchange rates, currently of EUR and NOK. The values are timeseries, so you can view a table of values or a diagram by right-clicking the field. You can also edit a value locally.

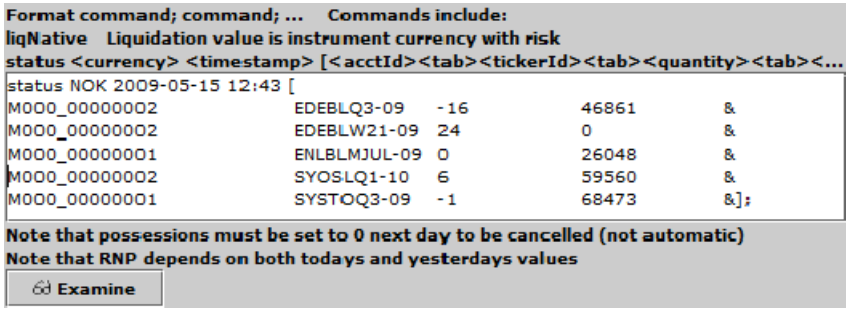


5.3.10 Ticker Search Tool



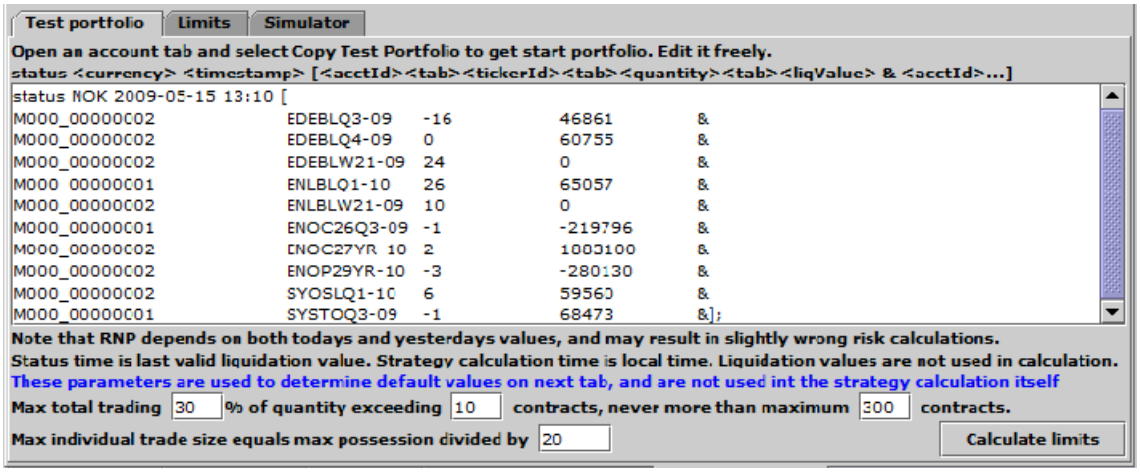
This tool is used for finding accounts that contain a certain instrument, and calculating the Energy Delta, which is the quantity multiplied by the instrument delta multiplied by the contract size.

5.3.11 Portfolio Test Tool

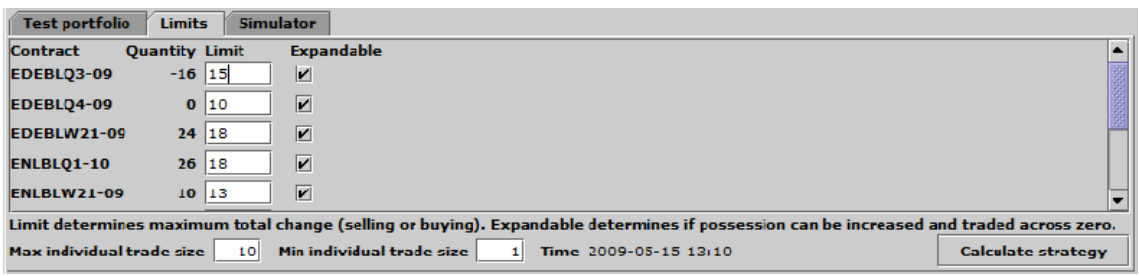


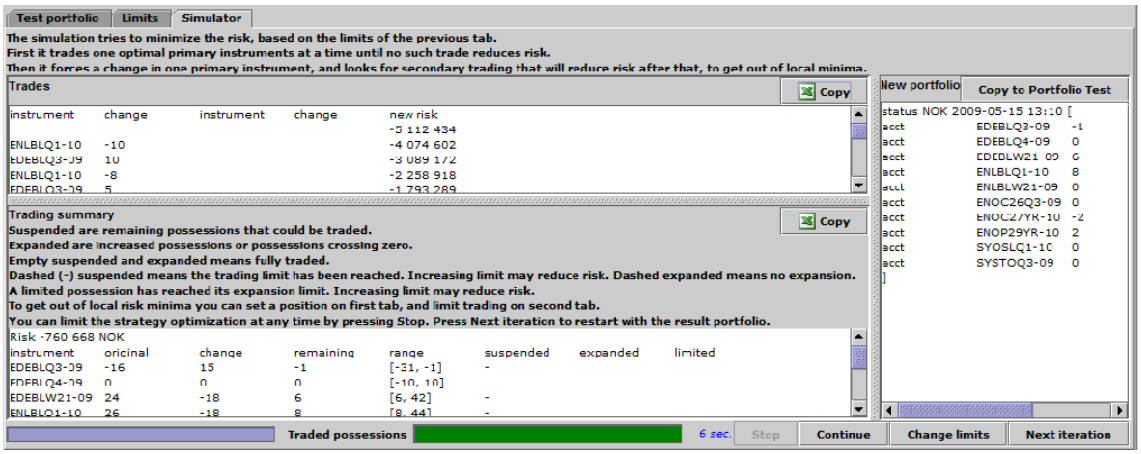
The Portfolio Test Tool is used for examining experimental portfolios. You can use the Copy dropdown menu of the Account tool of an account to create a portfolio, which you can alter with this tool, e.g. adding or removing possessions.

5.3.12 Closing Strategy Tool



This is, in my opinion, the most powerful tool of PowerPilot. It is used for finding the optimal closing strategy of a portfolio, e.g. if a member company defaults. It will create trades that reduce the margin call of the portfolio as much as possible, using a number of limits, such as maximum number of traded contracts in a day.



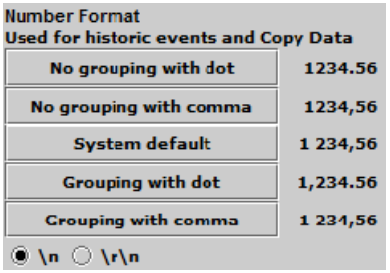


The result is a strategy for one day to reduce the risk; the trades, a summary of what limits were met, as well as the resulting portfolio. This portfolio can be copied to the start, and the strategy for the next day can be optimized.

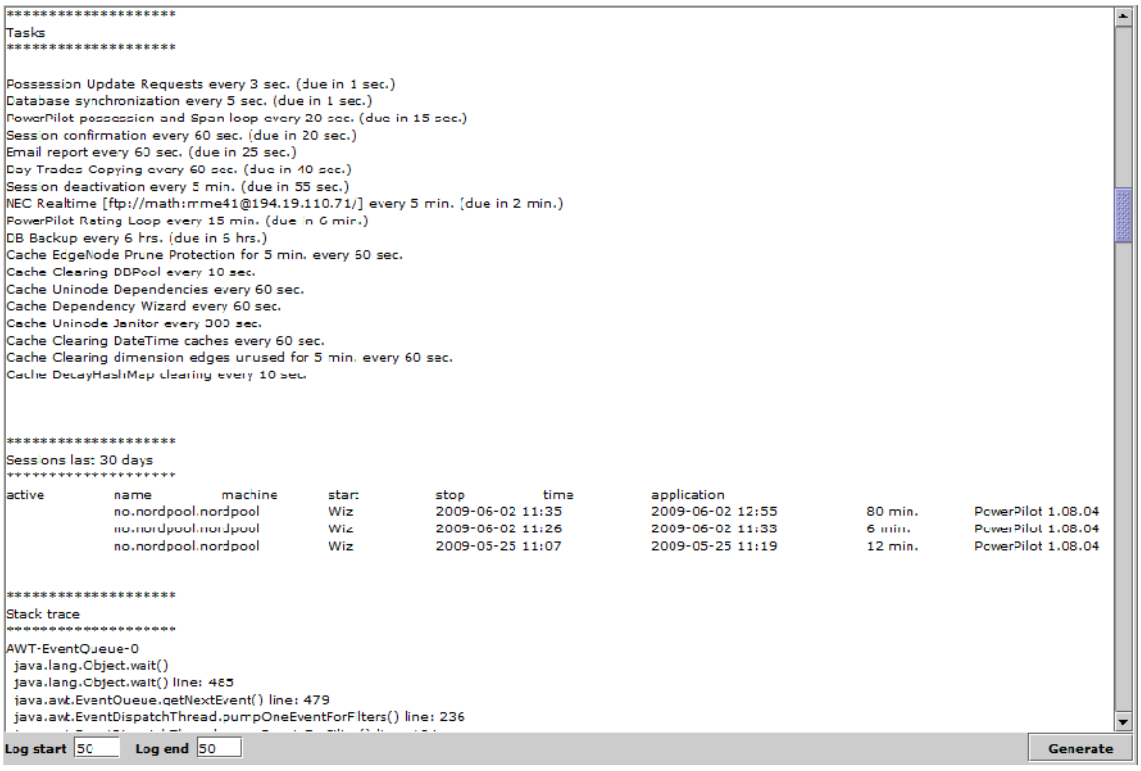
5.3.13 Preference Tool

Use this tool to change how you want number output to look like, e.g. when you copy to Excel.

You can also change if you want Windows or UNIX line breaks.

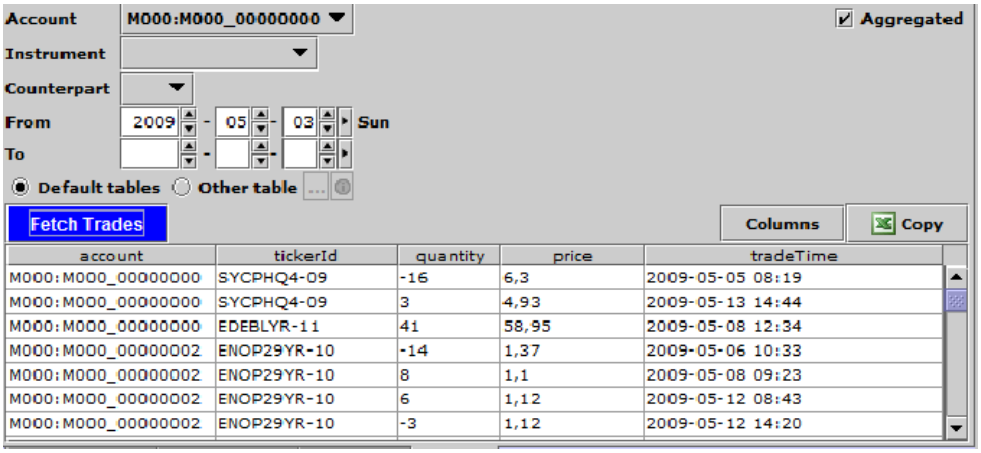


5.3.14 Server Report Tool



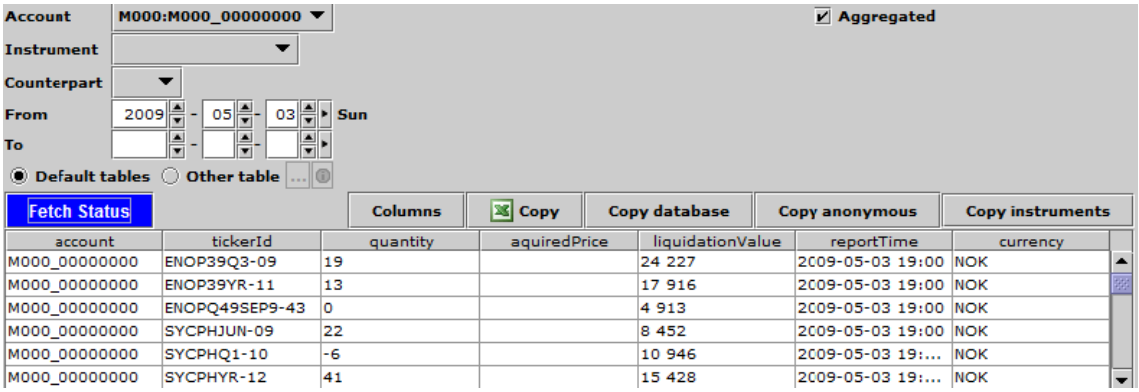
This tool is used for examining the status of the PowerPilot server. The status is the same as the one emailed to the support every day. The report includes update information, as well as progress reports of periodical tasks, usage, installed client versions and stack traces.

5.3.15 Trades Tool



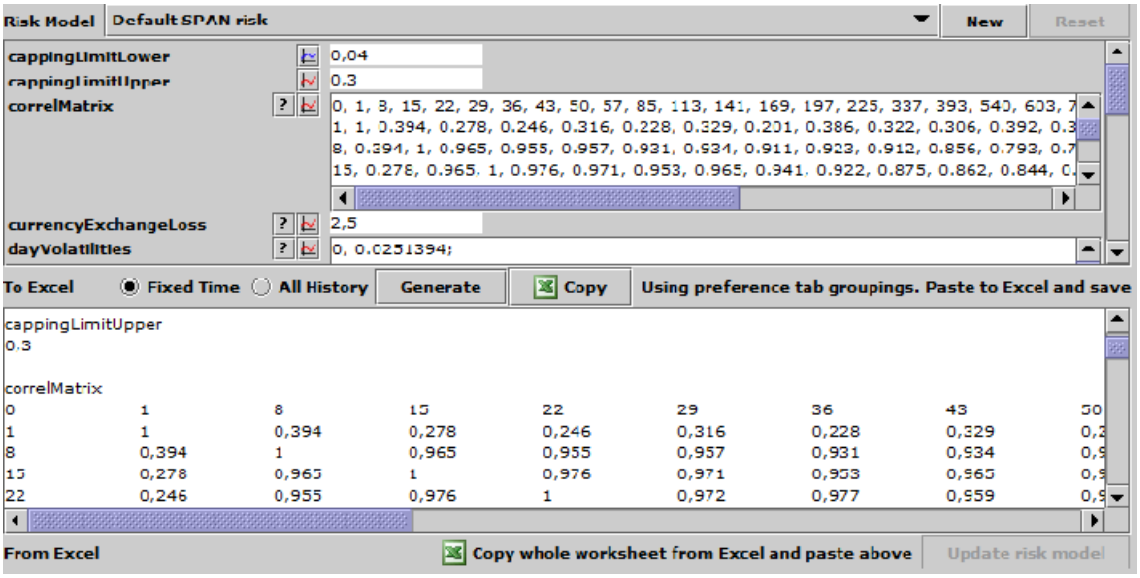
This tool is used for examining trade entries in the database.

5.3.16 Status Tool



This tool is used for examining end-of-day accounts status in the database. It is mainly used for debugging.

5.3.17 Risk Model Tool

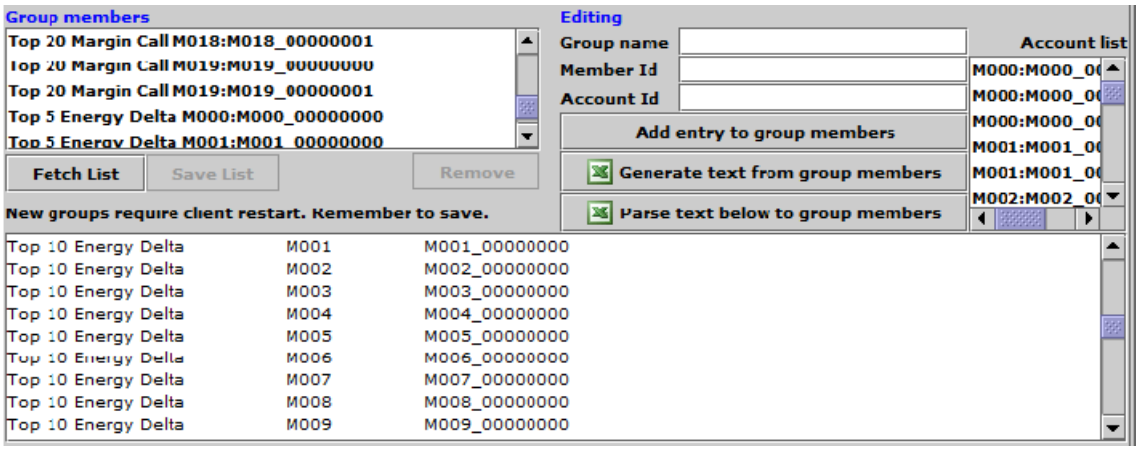


The Risk Model Tool is used for testing altered risk model parameters, such as correlation matrices and rate changes. You can select a new Risk Model here, which will be displayed in the main window, and used for all tools, e.g. the Local Rating tool.

To avoid mistakes, you must change the model here, and not in the main window itself.

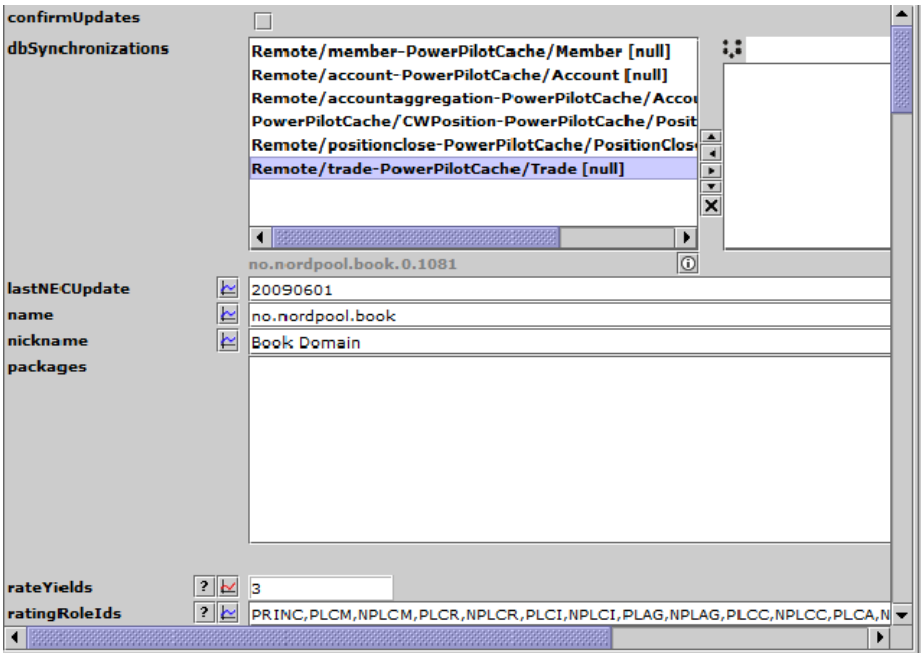
To alter the global risk model, you must use the Uninode Node Editor to alter node com.powerpilot.5000. This is something you do not do by mistake.

5.3.18 Rating Groups Tool



This tool is used for creating and editing rating groups, which are selections of accounts that are especially interesting to someone for some reason.

5.3.19 Domain Tool

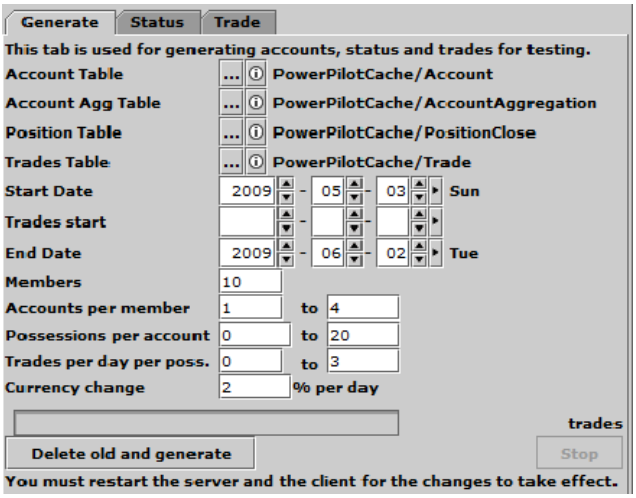


The server parameters are part of the Uninode net, more specifically attributes of the node that represent the book domain, containing all book-, account- and instrument information.

You can specify external databases that should be copied to the local database cache.

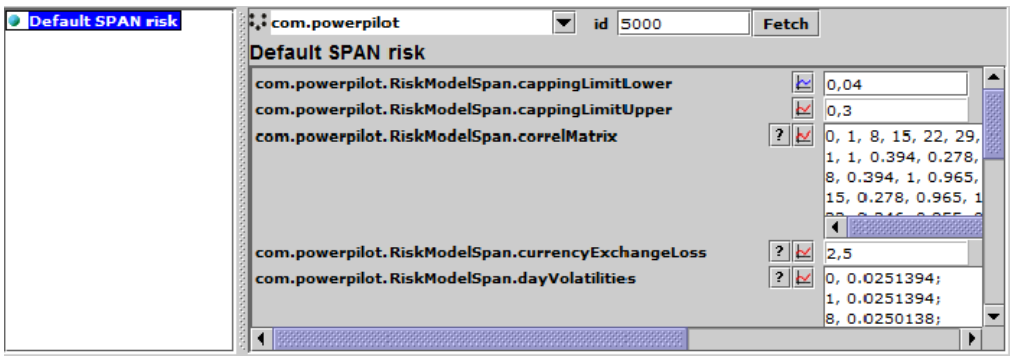
You can specify the rate timeseries that is used for option valuation and such. This value is used by the local environment root.

5.3.20 Simulator Tool



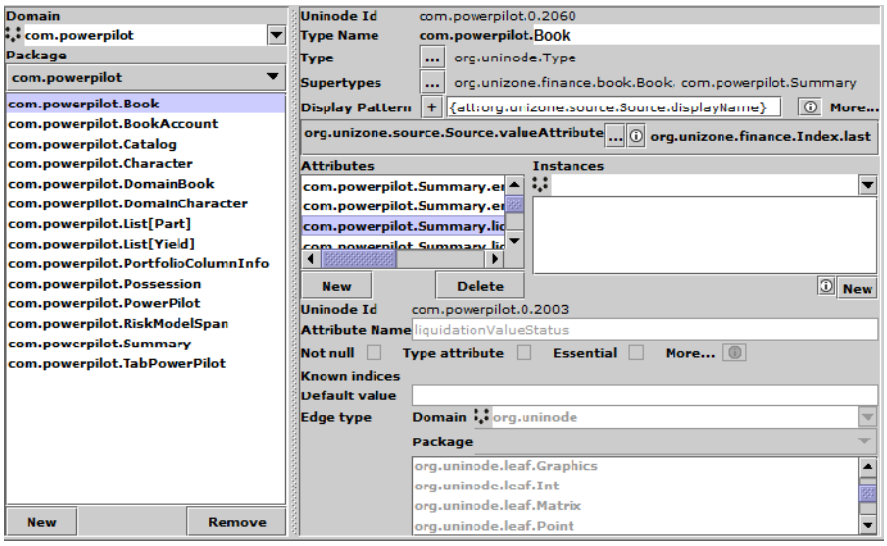
The Simulator tool is used for creating random accounts, trades and possessions for debugging. You can set the simulator to add errors to be detected by automatic trade validators.

5.3.21 Node Editor



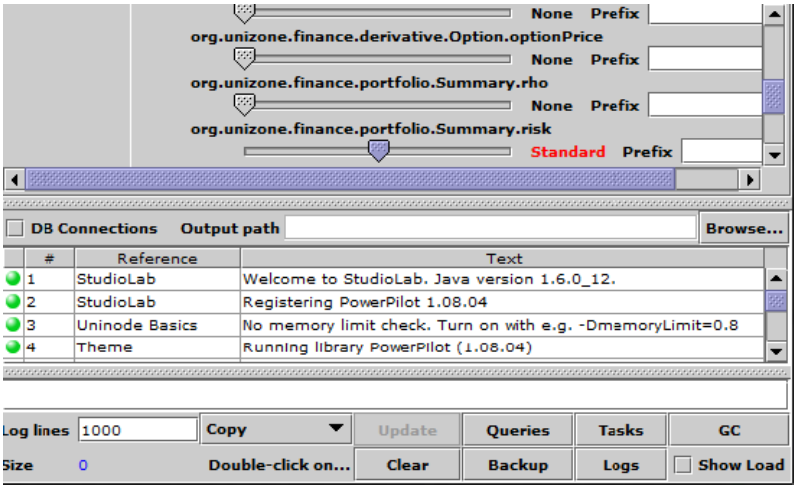
The Node editor is used for manually editing the global Uninode net.

5.3.22 Type Editor



The type editor is used for manually creating and editing Uninode types, attributes and instances.

5.3.23 Logging



Extensive logging is crucial for implementing and debugging complex, multithreaded mathematical methods, especially when dealing with mid-calculation rounding and truncation. An automatic testing environment is not sufficient to find all boundary cases, although it is useful for ensuring that the errors found are dispelled forever. Logging is used for finding cases that should be added to the automatic testing set.

6. Conclusions

Developing PowerPilot has been a joyful experience. Although the first year was very intense, with hardware and communication issues and new requirements part of everyday life, it soon settled into a very robust system. Now, new instruments and tools are generally added within two days of initial specification, mostly thanks to a very general model for financial instruments.

As far as I know, the risk group at Nord Pool has been pleased with PowerPilot, at least for the last five years. It is used daily by a number of analysts, and they install new updates as I write them.

6.1 Development

PowerPilot is not the most modern software. It has been developed in a fairly organical fashion by one developer. The size is about 130,000 rows of code.

If I were to develop it today, or if I needed to clean up the code for the benefit of someone else, I would consider the current implementation a legacy system, or a specification for the next generation, and use test-driven development to a greater extent. This is another strategy than the solid reality analysis that is the basis of PowerPilot, since it is based on testable examples, not principles.

I have written a new version of Uninode, with support for reflection and more generic Uninode ids, but I have not updated the PowerPilot code base.

6.2 Usage

One major user interface problem is to make the user confident that he is getting the right information. PowerPilot is used both for realtime and historical analysis. It is also used both for monitoring and experimentation. When a user sees a number, he must be sure of the background behind its calculation.

The most critical assumption is the timestamp. In pure realtime systems, the numbers are always the latest. With batch- or Excel systems, the time is generally either current or obvious. The solution is to put a general time editor and other environment editors at the same place in full view all the time.

Another way to instill confidence is to generate reports and send by email. One example is a rating report that is sent to a specified recipient every day, with key values. It has no parameters, and cannot be easily misunderstood. It is basically a batch system summary.

6.3 Support

The daily server report email generally alerts me of misbehavior before the risk group has discovered it. Since the servers generally behave, it is the kind of positive feedback I need as a single developer without any external connection to the inner network at Nord Pool.

We have two separate PowerPilot servers, one main server and one backup. They are completely unaware of each other, and can be used for monitoring each other, using manual comparison of the key indicator reports.

6.4 The future of PowerPilot

The risk group at Nord Pool Clearing is planning the development of a PowerPilot replacement in-house. This is one reason why I write this report. PowerPilot will not be supported beyond 2010.

The decision is a sound one. It is not good practice for a risk analysis group to depend on an application that is owned by an external company, and supported by a single developer, for too long. Seven years is on the upper end of the scale. I would call this a support risk.

7. References

HULL, JOHN C. 1997. *Options, futures, and other derivatives, third edition.* Prentice Hall. ISBN 0132643677.

NORD POOL 2009. *About Nord Pool.* http://www.nordpool.com/en/asa/General-information/