Vespa – Simulation User Manual and Reference

Version 0.1.0

Release date: October 1st, 2010

Developed by:

Brian J. Soher, Ph.D.
Philip Semanchuk

Duke University Medical Center,
Department of Radiology, Durham, NC

Karl Young, Ph.D.
David Todd, Ph.D.

University of California, San Francisco
Department of Radiology, San Francisco, CA

Developed with support from NIH, grant # EB008387-01A1

Table of Contents

Overview of the Vespa Package

The Vespa package extends the maintenance and development of three previously developed magnetic resonance spectroscopy (MRS) software tools by migrating them into an integrated, open source, open development platform. Vespa stands for Versatile Simulation Pulses and Analysis. The original tools that have been migrated into this package include GAVA/Gamma - software for spectral simulation, MatPulse – software for RF pulse design and IDL_Vespa – a package for spectral data processing and analysis. The new Vespa project will address current software limitations, including: non-standard data access, closed source multiple language software that complicates algorithm extension and comparison, lack of integration between programs for sharing prior information, and incomplete or missing documentation and educational content.

Introduction to Vespa-Simulation

Vespa-Simulation is a graphical control and visualization program written in the Python programming language that provides a user friendly front end to the Gamma/PyGamma NMR simulation libraries. The Vespa-Simulation interface allows users to:

1) Create and run an Experiment (consisting of one or more spectral simulations) from lists of metabolites and pulse sequences.
2) Store Experiment results in a database.
3) Display the results in a flexible plotting/graphing tool.
4) Compare side-by-side results from one or more Experiments
5) Output results in text or graphical format
6) Export/Import experiments, metabolites or pulse sequences from other users

What is an Experiment? An 'Experiment' object consists of one or more spectral Simulation objects. Each Experiment object uses only one "pulse sequence" but can contain one or more metabolites and one or more sets of timings for the pulse sequence. Each Simulation object contains results for a single metabolite for one set of sequence timings. Each call to the PyGamma library produces results for a single Simulation object. Vespa-Simulation loops through spectral simulations for all timings and metabolites to completely fill an Experiment.

There are a number of predefined pulse sequences in the Vespa-Simulation environment, and users can also develop their own PyGamma pulse sequence scripts. The database also maintains the prior information for the NMR parameters of available compounds (J-coupling and chemical shift values) necessary to run the simulations. NMR parameters are available in this database for approximately 30 compounds commonly observed for *in vivo* ^{1}H MRS.

The following chapters run through the operation of the Vespa-Simulation program both in general and widget by widget.

In this manual, command line instructions will appear in a fixed-width font on individual lines, for example:

```
~/Vespa-Simulation/ % ls
```

Specific file and directory names will appear in a fixed-width font within the main text.

Examples of spectral simulation for pulse optimization and spectral fitting:

Young K, Govindaraju V, Soher BJ and Maudsley AA. *Automated Spectral Analysis I: Formation of a Priori Information by Spectral Simulation*. Magnetic Resonance in Medicine; 40:812-815 (1998)

Young K, Soher BJ and Maudsley AA. *Automated Spectral Analysis II: Application of Wavelet Shrinkage for Characterization of Non-Parameterized Signals.* Magnetic Resonance in Medicine; 40:816-821 (1998)

Soher BJ, Young K, Govindaraju V and Maudsley AA. *Automated Spectral Analysis III: Application to in Vivo Proton MR Spectroscopy and Spectroscopic Imaging.* Magnetic Resonance in Medicine; 40:822-831 (1998)

Soher BJ, Vermathen P, Schuff N, Wiedermann D, Meyerhoff DJ, Weiner MW, Maudsley AA. *Short TE in vivo (1)H MR spectroscopic imaging at 1.5 T: acquisition and automated spectral analysis*. Magn Reson Imaging;18(9):1159-65 (2000).

Using Simulation – A User Manual

1. Overview – How to launch Vespa-Simulation

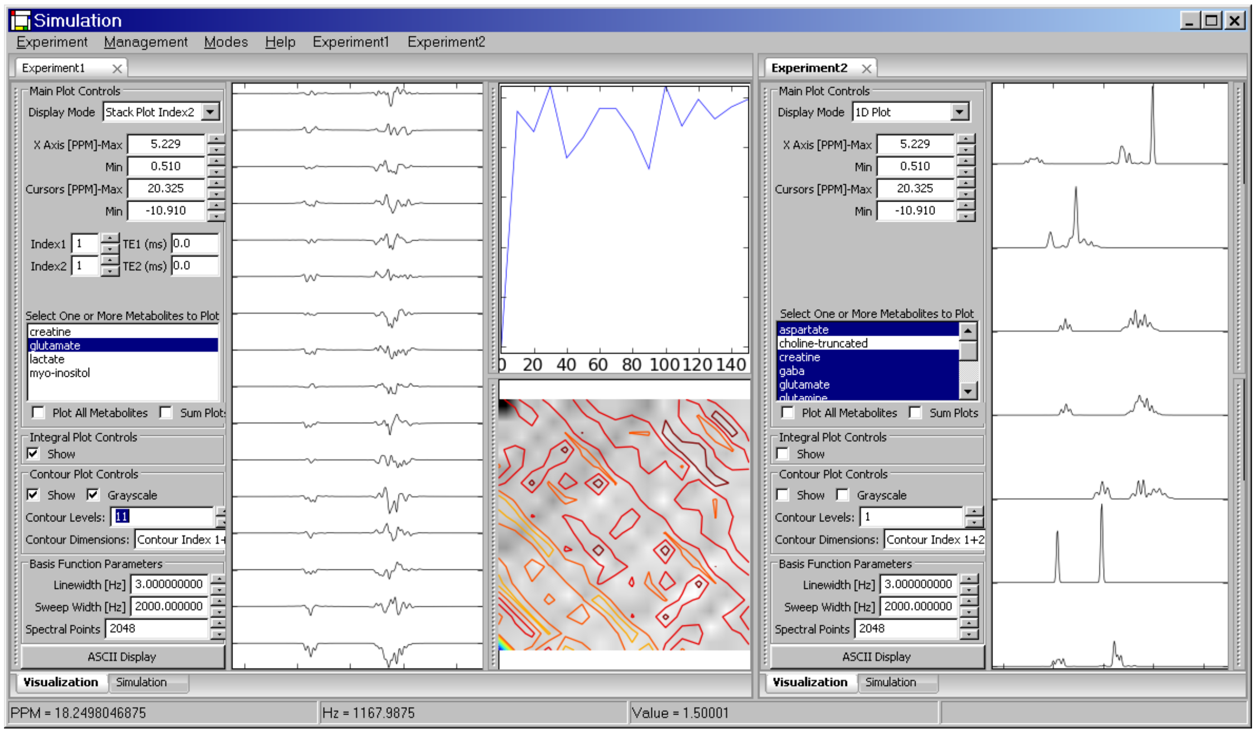
See the Vespa Installation guide for full details on package dependencies and how to install the software. The easiest way to start Simulation under Windows is to create a shortcut that executes the program:

- Right click on the desktop and select **New->Shortcut**
- Type the following Target, change directories as needed:

```
C:\Python25\python.exe C:\vespa\simulation\src\main.py
```

- Click **Next** and name the alias Simulation (or whatever), click **OK**

This shortcut will run python and Vespa-Simulation from a command window. If there is a bug in this release of Vespa-Simulation and the program crashes, it should display information in this window. Please refer to this code if you are reporting bugs.



Experiment1

Main Plot Controls
Display Mode: Stack Plot Index2
X Axis [PPM]-Max: 5.229
Min: 0.510
Cursors [PPM]-Max: 20.325
Min: -10.910
Index1: 1 TE1 (ms): 0.0
Index2: 1 TE2 (ms): 0.0
Select One or More Metabolites to Plot: creatine, glutamate, lactate, myo-inositol
Plot All Metabolites ☐ Sum Plots ☐
Integral Plot Controls: Show ☑
Contour Plot Controls: Show ☑ Grayscale ☑
Contour Levels: 11
Contour Dimensions: Contour Index 1+
Basis Function Parameters
Linewidth [Hz]: 3.000000000
Sweep Width [Hz]: 2000.000000
Spectral Points: 2048

Experiment2

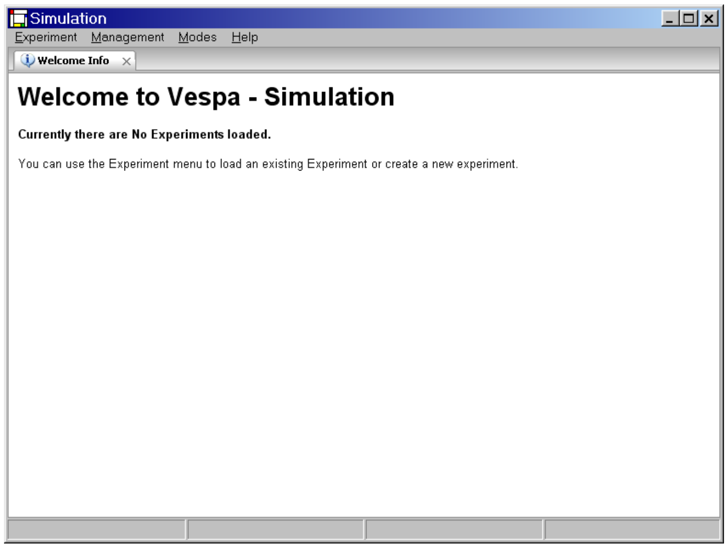
Main Plot Controls
Display Mode: 1D Plot
X Axis [PPM]-Max: 5.229
Min: 0.510
Cursors [PPM]-Max: 20.325
Min: -10.910
Select One or More Metabolites to Plot: aspartate, choline-truncated, creatine, gaba, glutamate, glutamine
Plot All Metabolites ☐ Sum Plots ☐
Integral Plot Controls: Show ☐
Contour Plot Controls: Show ☐ Grayscale ☐
Contour Levels: 1
Contour Dimensions: Contour Index 1+2
Basis Function Parameters
Linewidth [Hz]: 3.000000000
Sweep Width [Hz]: 2000.000000
Spectral Points: 2048

PPM = 18.2498046875 Hz = 1167.9875 Value = 1.50001

Shown here is a screen shot of a Vespa-Simulation session with two Experiment tabs opened side by side for comparison. The functionality of all tools will be described further in the following sections.

2. The Simulation Main Window

This is a view of the main Vespa-Simulation user interface window. It is the first window that pops up when you run the program. It contains the Experiment Notebook, a menu bar and status bar. The Experiment Notebook can be populated with one or more Experiment Tabs, each of which contains the results of one Experiment. An Experiment is a collection of spectral Simulations. Each Simulation contains the result for one metabolite that has been run through a simulated pulse sequence for a given set of sequence parameters. Thus, an Experiment may consist of one metabolite for multiple sets of pulse sequence parameters, or multiple metabolites for one set of pulse sequence parameters, or multiple metabolites for multiple collections of pulse sequence parameters.



Welcome to Vespa - Simulation

Currently there are No Experiments loaded.

You can use the Experiment menu to load an existing Experiment or create a new experiment.

The Experiment Notebook is initially populated with a welcome text window, but no Experiment results. From the Experiment menu bar you can 1) load a previously run Experiment from the Simulation database into a tab, or 2) create a new Experiment in a tab and set it up and run it. The Management menu allows users to run pop-up dialogs to create, edit, view, delete and import/export Experiment, Metabolites and Pulse Sequences from the Simulation database.

The status bar provides information about where the cursor is in various plots and images throughout the program. It also reports short messages that reflect current processing while long running events are occurring.

<u>On the Menu Bar</u>

Experiment→New	Opens a new Experiment Tab in the Experiment Notebook.
Experiment→Open	Runs the Experiment Browser dialog.
Management →Manage Experiments	Launches the Experiment Browser dialog. Allows user to view, clone, delete, import and export Experiments.
Management →Manage Metabolites	Launches the Manage Metabolite dialog. Allows user to create, edit, view, clone, (de-)activate, delete, import and export Metabolite prior information.
Management →Manage Pulse Sequences	Launches the Manage Pulse Sequences dialog. Allows user to create, edit, view, clone, delete, import and export Pulse Sequence information.
Modes→Experiment	This menu item allows the user to switch back and forth between preset groups of panes/widgets in the main window. At the moment, there is only the Experiment Mode, which shows the Experiment Notebook. At some point in the future there will also be a Pulse Sequence Design Mode and possibly others.
Help→Show Inspection Tool	Launches the wxPython widget inspection tool. This is a debugging aid to help developers track the events attached to each widget.
Help→About	Giving credit where credit is due.

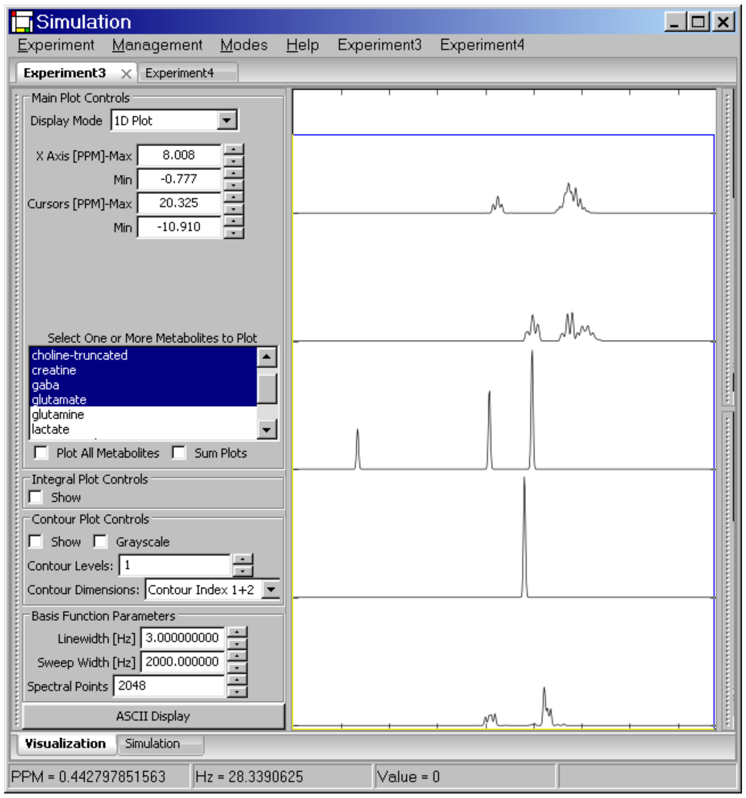
3. The Experiment Notebook

The Experiment Notebook is an “advanced user interface” notebook widget (AUINotebook). Multiple tabs can be opened up inside the window. They can be moved around, arranged and "docked" as the user desires by left-click and dragging the desired tab to a new location inside the notebook boundaries. In this manner, the tabs can be positioned side-by-side, top-to-bottom or stacked (as show in Sections 1 and 4). They can also be arranged in any mixture of these positions.

The Experiment Notebook can be populated with one or more Experiment Tabs, each of which contains the results of one Experiment. Tabs can be closed using the X box on the tab or with a middle-click on the tab itself. When a Tab is closed, the Experiment is removed from memory, but can be reloaded from the database at a future time.

4. The Experiment Tab

An Experiment Tab is a widget pane that is added to the Experiment Notebook. Each tab contains one entire Experiment. An Experiment Tab can be used to run a new Experiment and then look at the results. It can also just be used to load an existing Experiment from the database to look at the results or to add more metabolites to the Experiment.



Display Mode: 1D Plot
X Axis [PPM]-Max: 8.008
Min: -0.777
Cursors [PPM]-Max: 20.325
Min: -10.910
choline-truncated
creatine
gaba
glutamate
glutamine
lactate
Contour Levels: 1
Contour Dimensions: Contour Index 1+2
Linewidth [Hz]: 3.000000000
Sweep Width [Hz]: 2000.000000
Spectral Points: 2048
PPM = 0.442797851563 Hz = 28.3390625 Value = 0

Each Experiment Tab has two sub-tabs called Visualization and Simulation. The Simulation tab is where a new experiment is set up and then run. It is also where the parameters and settings for an existing Experiment can be reviewed when the Experiment is reloaded. The Visualization tab is where the results of an Experiment can be visualized as 1D plots, stack plots, and peak integral contour maps.

When a new Experiment is set up, there are no results to be displayed so the program defaults to the Simulation tab for New Experiments. When an existing Experiment is loaded, there are results to be seen so the program defaults to the Visualization tab.

A New Experiment is typically created, set up and run. On completion, the results are automatically saved to the database and the Visualization tab updated to display these results. Once an Experiment has been run once, it can only be "run again" to add additional metabolites. The exact same parameters are used for subsequent runs, excepting the list of metabolites to be simulated.

As each Experiment Tab is created, a new menu item appears in the menu bar that allows the user to change things inside that particular Experiment Tab. Each tab has an number on it (e.g. Experiment1, Experiment2, etc.). The menu item has a matching label.The following lists a summary of the functions on the Experiment Tab menu item:

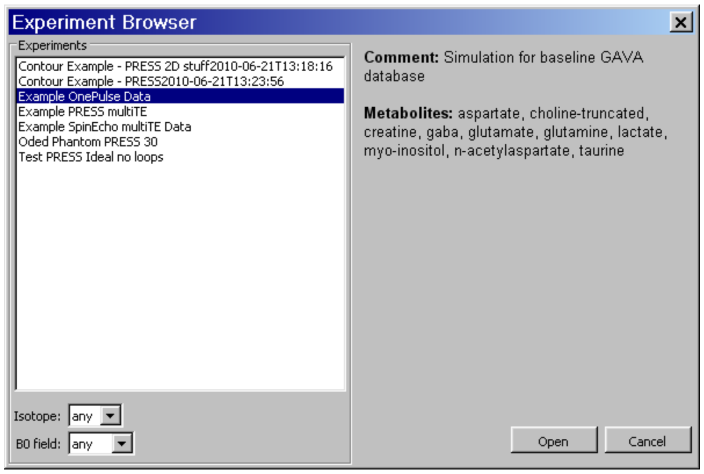
On the Menu Bar

ExperimentX (where 'X' stands for the experiment tab number, e.g. 'Experiment1', 'Experiment2')

→ZeroLine On/Off	toggle zero line off/on in 1D and stack display
→Xaxis →On/Off	white lines on black background or reversed
→Xaxis→PPM/Hz	x-axis value in PPM or Hz
→Plot Color	white lines on black background or reversed
→Data Type	select Real Imaginary or Magnitude spectral data to display
→Lineshape	select Gaussian or Lorentzian lineshapes for the basis functions plotted
→Integral Axes	toggles either x or y, or both axes on/off
→Contour Axes On/Off	toggles both axes on/off
→Output→1D/Stackplot	writes plot currently in the 1D or StackPlot canvas to file as either PNG, SVG, EPS or PDF format
→Output→Integral Plot	writes plot currently in the Integral plot canvas to file as either PNG, SVG, EPS or PDF format
→Output→Contour Plot	writes plot currently in the Contour plot canvas to file as either PNG, SVG, EPS or PDF format
→Output→Text Results	opens the operating systems standard text editor and inserts a textual rendering of the Experiment results. Typically, this is a summary of the general descriptive information, the specific pulse sequence and metabolite parameters included and a listing of all metabolite lines for every loop instance in the Experiment.

4.1 Loading an existing Experiment

The Experiment Browser dialog is launched from the **Experiment→Open** menu and is shown below. A list of Experiment names is shown on left. As an Experiment is clicked on once, its comment and metabolites list are displayed on the right. Experiments can be sorted by the isotopes contained within the metabolites simulated. They can also be sorted by field strength (given in MHz).



Contour Example - PRESS 2D stuff2010-06-21T13:18:16
Contour Example - PRESS2010-06-21T13:23:56
Example OnePulse Data
Example PRESS multiTE
Example SpinEcho multiTE Data
Oded Phantom PRESS 30
Test PRESS Ideal no loops

Comment: Simulation for baseline GAVA database

Metabolites: aspartate, choline-truncated, creatine, gaba, glutamate, glutamine, lactate, myo-inositol, n-acetylaspartate, taurine

Isotope: any
B0 field: any

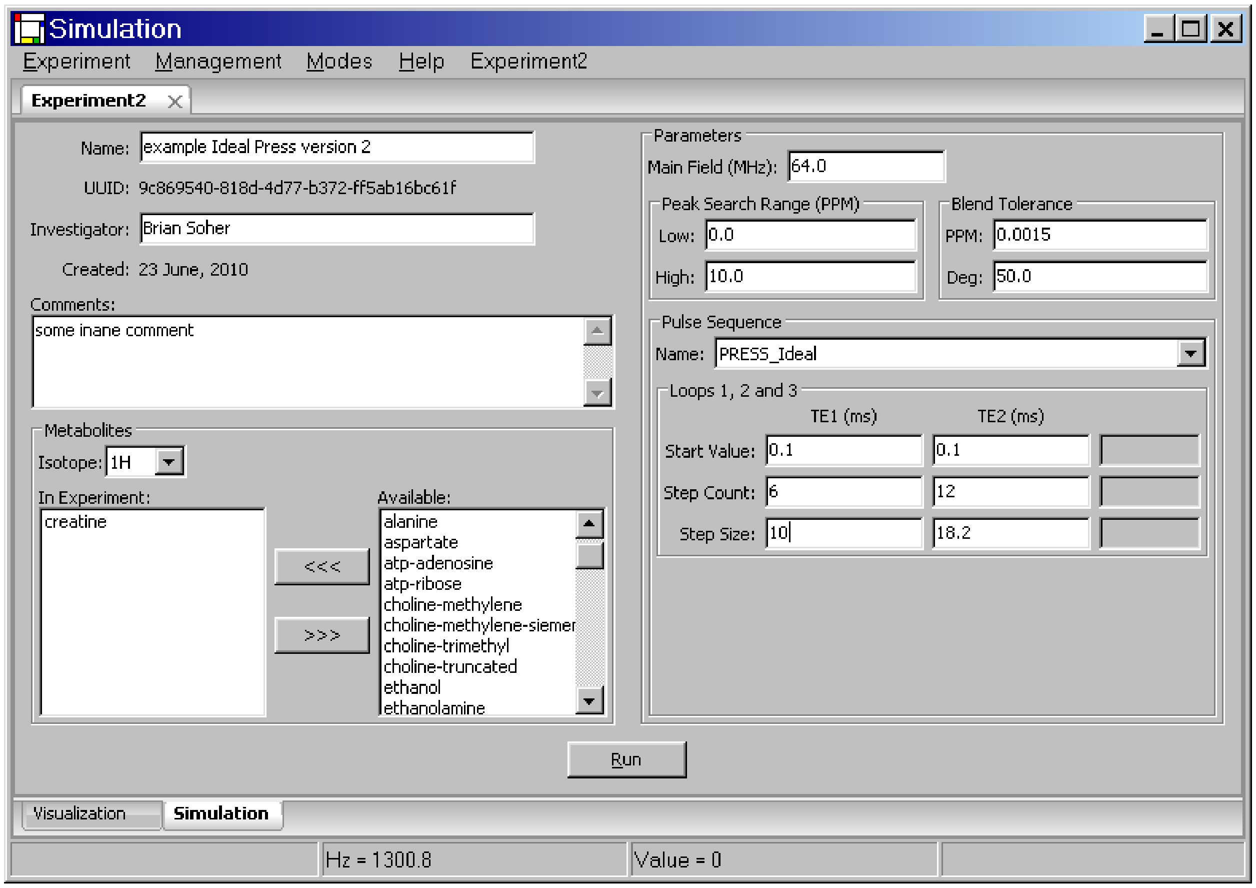
When the open button is clicked (or an Experiment name double-clicked on), the program loads the information for the Experiment from the database into an Experiment object in memory. This object then creates a set of basis functions for all metabolites for use in the Visualization tab plots. NB. In the case of large Experiment, this may take a significant amount of time to calculate, but is indicated on the lower left of the status bar while calculating.

4.2 Running a new Experiment

An 'Experiment' object consists of one or more spectral Simulation objects. Each Experiment object uses only one "pulse sequence" but can contain one or more metabolites and one or more sets of timings for the pulse sequence. Each Simulation object contains results for a single metabolite for one set of sequence timings. Each call to the PyGamma library produces results for a single Simulation object. Vespa-Simulation loops through spectral simulations for all timings and metabolites to complete an Experiment.

When a user selects the **Experiment→New** menu option, a new Experiment Tab is created in the Experiment Notebook and the default view is for the Simulate sub-tab. This panel enables the user to select, define and run a new Experiment from the list of defined pulse sequences provided with the Simulation program. Additional pulse sequences can be created by the user and accessed using the methods covered in the next section.

A list of all pulse sequences is kept in the Vespa-Simulation database and can be selected from the **Pulse Sequence: Name** dropdown menu. The Simulation widget will reconfigure itself based on the parameters needed to run that sequence. Users must fill in the **Name**, **Investigator**, **Main Field**, **Peak Search Ranges**, **Blend Tolerances** and all loop **Start Value**, **Step Count** and **Step Size** fields. At least one metabolite must be selected and moved into the **In Experiment** list. Some default values are already included.



Name: example Ideal Press version 2

UUID: 9c869540-818d-4d77-b372-ff5ab16bc61f

Investigator: Brian Soher

Created: 23 June, 2010

Comments: some inane comment

Metabolites

Isotope: 1H

In Experiment: creatine

Available: alanine, aspartate, atp-adenosine, atp-ribose, choline-methylene, choline-methylene-siemer, choline-trimethyl, choline-truncated, ethanol, ethanolamine

Parameters

Main Field (MHz): 64.0

Peak Search Range (PPM) Low: 0.0 High: 10.0

Blend Tolerance PPM: 0.0015 Deg: 50.0

Pulse Sequence Name: PRESS_Ideal

Loops 1, 2 and 3

	TE1 (ms)	TE2 (ms)	
Start Value:	0.1	0.1	
Step Count:	6	12	
Step Size:	10	18.2	

Hz = 1300.8 Value = 0

Simulation provides the user with four looping variables for use in their pulses sequences. This is covered in much more detail in Appendix XX, however, in brief: The first loop is the list of selected metabolites. The remaining three loops are defined as evenly spaced floating point number series. Each series is defined by a starting value, a number of steps and a step size. So for start = 10.2, steps = 4, size = 2.0 we would get the following values in a dimension [10.2, 12.2, 14.2, 16.2]. This value is passed directly to the user's PyGamma code and can be used in any fashion. One use might be to use these values directly as sequence timing values where they represent [ms] timings between RF pulses. Another way might be to use an integer series (e.g. [1,2,3,4,5,6]) as the index for a series of RF pulses stored in a file. This way an Experiment could “loop” through the effects of different RF pulses in an experiment. Either way, the user can set up three of these loops in the **Loops 1, 2 and 3** section of the Simulation sub-tab. Shown is an example of a new Experiment tab configured for a PRESS simulation.

Note: Metabolite Peak Normalization and Blending

The transition tables created by the GAMMA routines frequently contain a large number of lines caused by degenerate splittings and other processes. At the end of each pulse sequence, we call a standard routine to extract all lines from the transition table. These lines are then normalized using a closed form calculation based on the number of spins. Multiple lines are blended by binning them together based on their PPM locations and phases. The following parameters are used to set these procedures:

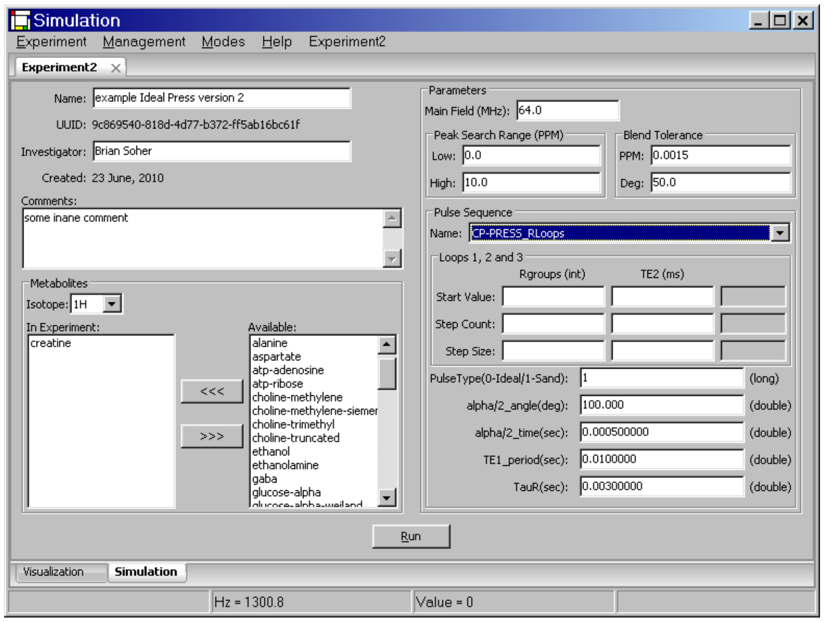
Peak Search Range – Low/High (PPM): the range in PPM that is searched for lines from the metabolite simulation.

Peak Blending Tolerance (PPM and **Degrees):** the width of the bins (+/- in PPM and +/- in PhaseDegrees) that are used to blend the lines in the simulation. Lines that are included in the same bin are summed using complex addition based on Amplitude and Phase.

4.3 New Experiments with additional user defined parameters

A full explanation of how to create additional pulse sequences, with or without additional parameters that Simulation can use, is given in Appendix XX. The Vespa-Simulation **Manage Pulse Sequences** dialog provides an interface for a user to define the additional parameters needed for a given pulse sequence. These are then saved to the Vespa-Simulation database. This section describes the interface used in a new Experiment Simulation Tab to run an Experiment using a sequence with additional parameters.

When a sequence with additional parameters is selected from the **Pulse Sequences** drop-list, the Simulate tab will be modified to set up widget fields where the user can define values for these additional parameters. Additional parameters are displayed in a list below the loop fields. Each line contains one parameter description and a field to set a value. A default value is typically provided. Field names and data types are pre-defined as String, Long or Float data types for data entry. The user is restricted to entering this type of data in any given field. All other widget functions in the first two columns of the widget function as described above.



Name: example Ideal Press version 2
UUID: 9c869540-818d-4d77-b372-ff5ab16bc61f
Investigator: Brian Soher
Created: 23 June, 2010
Comments: some inane comment

Metabolites
Isotope: 1H
In Experiment: creatine
Available: alanine, aspartate, atp-adenosine, atp-ribose, choline-methylene, choline-methylene-siemer, choline-trimethyl, choline-truncated, ethanol, ethanolamine, gaba, glucose-alpha, glucose-alpha-weiland

Parameters
Main Field (MHz): 64.0
Peak Search Range (PPM) — Low: 0.0 High: 10.0
Blend Tolerance — PPM: 0.0015 Deg: 50.0

Pulse Sequence
Name: CP-PRESS_RLoops

Loops 1, 2 and 3

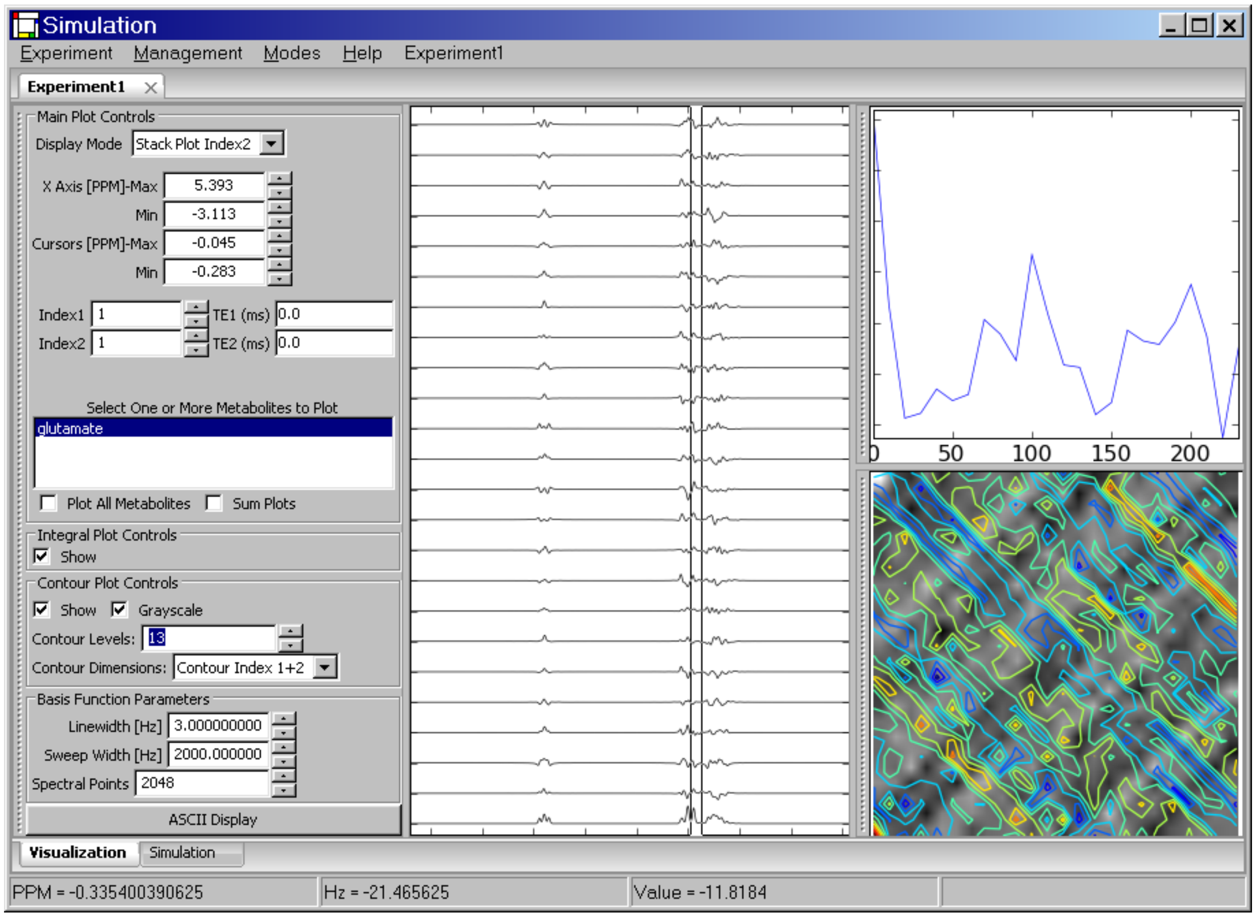
	Rgroups (int)	TE2 (ms)	
Start Value:			
Step Count:			
Step Size:			

PulseType(0-Ideal/1-Sand):	1	(long)
alpha/2_angle(deg):	100.000	(double)
alpha/2_time(sec):	0.000500000	(double)
TE1_period(sec):	0.0100000	(double)
TauR(sec):	0.00300000	(double)

Hz = 1300.8 Value = 0

4.4 Visualizing Experiment Results

Experiments displayed in the Visualize widget can be considered to contain 2, 3, 4 or 5 dimensions that correspond to the Spectral dimension, the number of metabolites in the experiment, and the number of steps in Loops 1, 2 and 3 respectively. Pulse sequences such as One-Pulse or Spin-Echo only allow 0 or 1 Loop dimensions and are thus restricted in the types of displays they can make use of. However, most other pulse sequences can typically use most the plot modes. The three plot modes for displaying results, 1D/StackPlot, Integral Plot and Contour Plot, are shown below:



Main Plot Controls
Display Mode: Stack Plot Index2
X Axis [PPM]-Max: 5.393
Min: -3.113
Cursors [PPM]-Max: -0.045
Min: -0.283
Index1: 1 TE1 (ms): 0.0
Index2: 1 TE2 (ms): 0.0
Select One or More Metabolites to Plot: glutamate
☐ Plot All Metabolites ☐ Sum Plots
Integral Plot Controls
☑ Show
Contour Plot Controls
☑ Show ☑ Grayscale
Contour Levels: 13
Contour Dimensions: Contour Index 1+2
Basis Function Parameters
Linewidth [Hz]: 3.000000000
Sweep Width [Hz]: 2000.000000
Spectral Points: 2048

PPM = -0.335400390625 | Hz = -21.465625 | Value = -11.8184

The **1D/StackPlot** window is always open and centered in the screen. The **Integral Plot** window and the **Contour Plot** window can be toggled on/off using the check box next to their names. Both the Integral and Contour plot windows can be undocked, repositioned and re-docked using the “grab bars” on the left hand side of each window.

Under the **1D/StackPlot** window, a 1D spectrum for one or more metabolites or a 2D spectral stack plot along any two Loop dimensions for a single metabolite can be selected. If more than one metabolite is selected for a stack plot, only the first metabolite in the list is displayed

The mouse can be use to set the x-axis and cursor values in the **1D** plots. The left mouse button sets the X-axis Min/Max PPM values. Click and hold the left mouse button in the window and a vertical cursor will appear. Drag the mouse either left or right and a second vertical cursor will appear. PPM value changes will be reflected in the Plot Control widget. Release the mouse and the plot will be redisplayed for the Min/Max PPM axis values. In a similar fashion, two vertical cursors can be set inside the plot window. Click and drag then release to set the two cursors anywhere in the window. Changes to the cursor values will be updated in the Integral and Contour plots (described below) after these values are changed by the user. Click and release the left mouse button in place and the plot will zoom out to its max setting. Click and release the right mouse button in place and the plot will zoom to its last zoom setting (ie. If you use the left mouse to zoom in three times getting tighter and tighter on a region, clicking in place with the right mouse will take you from zoom 3 to zoom2 to zoom1 to full with each successive click).

An **Integral** plot can be created from a 2D Spectral stack plot experiment for a single metabolite. Metabolite areas are measured between the **Left and Right Cursor** settings in each spectrum and for the real, imaginary or magnitude data shown. The plot will show the integral along the Stack Plot axis displayed in the 1D/StackPlot Once the **Integral** plot is displayed, changes to the **Left and Right Cursor** values or to the Loop index widgets are reflected in the plot.

The **Contour** plot works best for Experiments that contain at least two Loop dimensions, but will create a "pseudo-2D" contour plot from an Experiment with only one Loop dimension by repeating the first dimension. Contours are integrated over all steps in the two loop dimensions selected in the **Contour Dimensions** drop-box, for the **Left and Right Cursor** settings shown in the Plot control widget and for the real, imaginary or magnitude data shown. Plotted contours change as the cursor settings change, but are only refreshed when the right mouse button is released.

<u>On the Visualize Widget</u>

Display Mode	(drop-list) Selects 1D, or Stack Plots along index 1, 2 or 3 to be displayed in the 1D window.
X Axis Max/Min	(click fields) Controls the PPM limits of the spectrum displayed in the 1D and 2D plots. Alternatively, the left mouse button can be used interactively in the 1D Display window to set these axes. Click down the left mouse button and drag to set the min/max settings using an interactive 'rubber-band' display method. X-axis cursors are displayed in gray/red.
Cursors Max/Min	(click fields) Controls the PPM limits of the cursors displayed in the 1D and Stack Plots. These also act as the PPM integral regions calculated in the Integral and Contour plots. The cursors are displayed in purple and may not be displayed on the screen if set to values outside the X Axis min/max values. Alternatively, the right mouse button can be used in an interactive 'rubber-band' display method in the 1D Display window to set these axes. Click down the left mouse button and drag to set the left/right values. Cursors are displayed in gray/yellow.
Index 1, 2, 3	(click fields) These fields allow the user to step thru the Loop1 and Loop2 dimensions for the various plot modes. As each Index widget is incremented, the sequence timing's actual value is shown in the adjoining field.
Metabolites	(list) A list of metabolites in the experiment that can be included in the display.
Plot ALL Metabolites	Highlight all metabolites in the list.
Plot ALL Metabolites	Sums all metabolite plots in a list. For 1D display, this sums different metabolite spectra together. For Stack Plots the different sequence timings for one metabolite are summed.
Integral Plot - Show	(check) Toggles Integral Plot display.
Contour Plot - Show	(check) Toggles Contour Plot display.
Grayscale	(check) Toggles whether a grayscale image overlay is applied as a background to the contour plot.

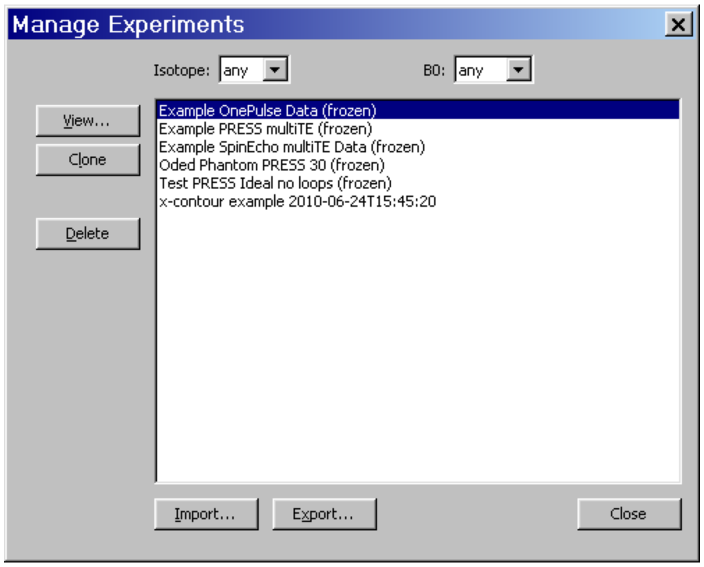
Contour Levels	(click field) Select the number of levels to display in the Contour Plot. Note that setting too many levels may limit the ability of level values from being displayed.
Contour Dimensions	(drop-list) Selects index pairs among index 1, 2 and 3 for display in plot.
Linewidth	(click field) Set the full-width half-max linewidth in Hz of the peaks displayed in the plots.
Sweep Width	(click field) Set the sweep width in Hz used to reconstruct the spectra.
Spectral Points	The number of points used to reconstruct the spectra.
ASCII Display	Displays the current Experiment results in text form. Top information is a summary of the Experiment parameters followed by a line by line report of metabolite results. Each line is tab-delineated and shows a: Metabolite Name, Loop1, Loop2 Index, Loop3 Index, Group Number Index, Line Number Index, Frequency(PPM), Amplitude, and Phase(deg) for each line extracted from the transition table for a given simulation.

5. Management Dialogs

The Management dialogs allows the user to Create, Delete, Edit, Import, Export or View Metabolites, Experiments and Pulse Sequences. These dialogs allow the user to manage the data in the Simulation database and to add new prior metabolite and pulse sequence information. It also provides the means for users to share information between themselves via XML files created using the Import/Export functions.

5.1 Manage Experiments dialog

Access this dialog by clicking on the **Management→Manage Experiments** menu item. The dialog opens and blocks other activity until it is closed. An example of this dialog is shown in the figure. Experiment names are listed in the window on the right. This list may be sorted by isotope or main B0 field strength from the drop-list widgets above the list. Users may View, Clone, Delete, Import or Export Experiments. These functions are summarized below.



Example OnePulse Data (frozen)
Example PRESS multiTE (frozen)
Example SpinEcho multiTE Data (frozen)
Oded Phantom PRESS 30 (frozen)
Test PRESS Ideal no loops (frozen)
x-contour example 2010-06-24T15:45:20

View: Not currently implemented, use Output→Text Results from the ExperimentN menu item instead.

Clone: Cloning allows the user to more easily re-run an Experiment with modifications to its parameters. Once Experiments are run, they can not have their parameters modified other than to add additional metabolites to the run list. The clone button allows a user to create a copy of an Experiment that contains all the parameter settings of the Experiment, but none of the results. The clone appears in the list as a copy of the Experiment name with the date and the word “clone” appended to its name. The user can now quit out of the dialog and load this cloned Experiment into a New Experiment Tab, modify parameters and run it.

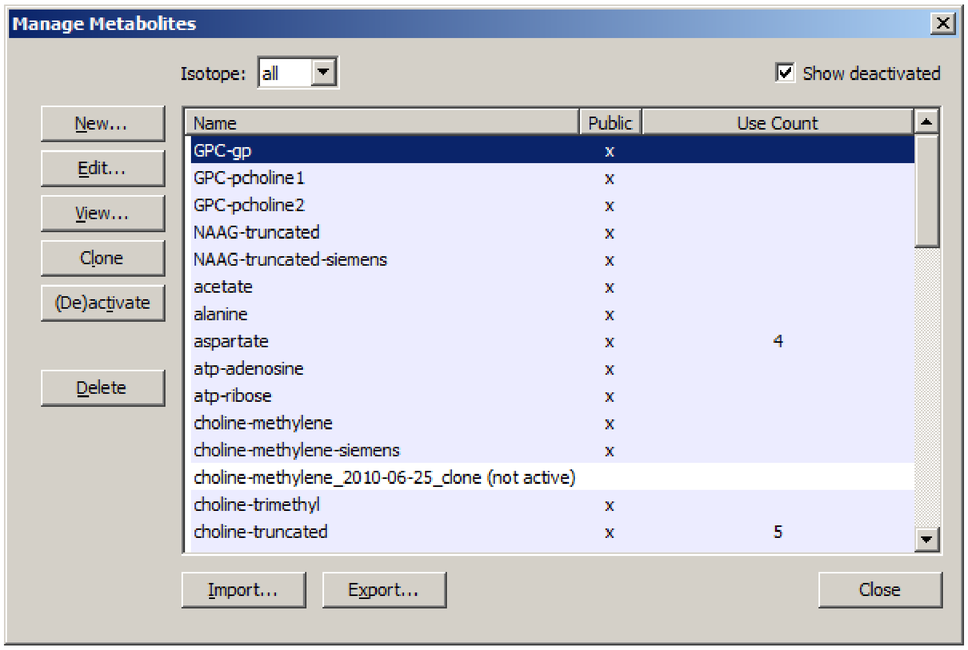
Delete: Removes the Experiment from the database.

Import: Allows the user to select an XML file that contains an Experiment. If the UUID in the file is unique, it is added to the Simulation database.

Export: The user selects an Experiment from the list. Simulation asks if both parameter and results should be included in export or just parameters. A second dialog allows the user to browse for the output filename, select if output should be compressed and allows an additional export comment to be typed in. Note that the action of exporting an object caused it to be marked as “frozen” in the database. This means that no changes can be made. This is for the sake of consistency as results are shared. However, a frozen Experiment can still be deleted from the database if needed. This file can be imported into another Vespa-Simulation installation using the Import function.

5.2 Manage Metabolites dialog

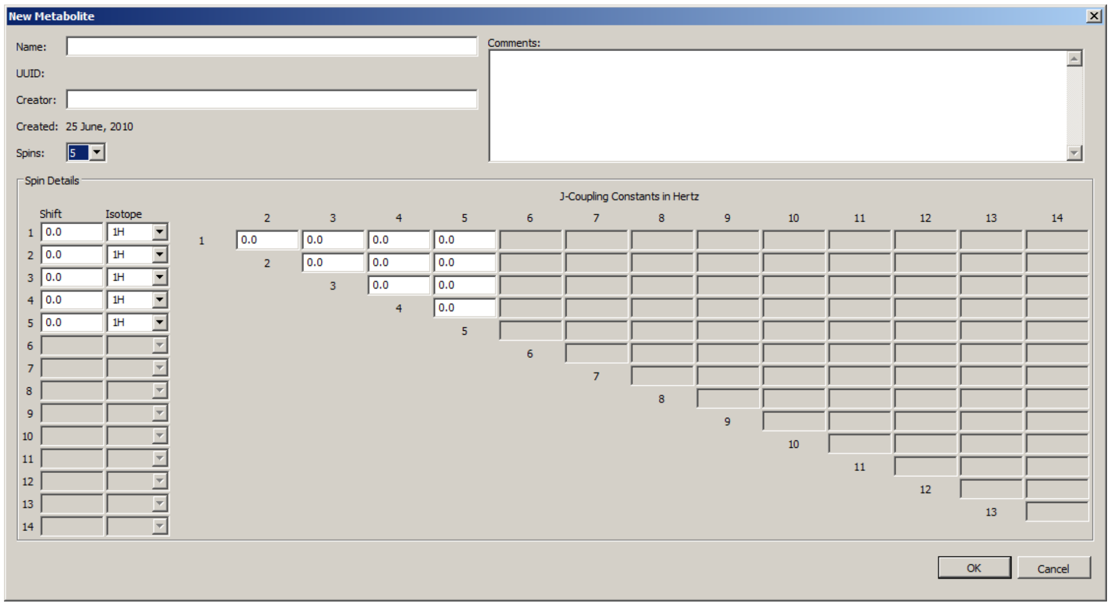
Access this dialog by clicking on the **Management→Manage Metabolites** menu item. Actions that can be taken on the Metabolite dialog include, New, Edit, View, Clone, (De)activate, Delete, Import and Export. An example of the widget used to display and edit Metabolite information is shown. The "Public" column indicates if a metabolite has ever been exported (or imported from someone else). This means that it should not be edited. The "Use Count" column indicates how many local Experiments use this metabolite. While in use by any Experiments, the metabolite can not be deleted.



Name	Public	Use Count
GPC-gp	x	
GPC-pcholine1	x	
GPC-pcholine2	x	
NAAG-truncated	x	
NAAG-truncated-siemens	x	
acetate	x	
alanine	x	
aspartate	x	4
atp-adenosine	x	
atp-ribose	x	
choline-methylene	x	
choline-methylene-siemens	x	
choline-methylene_2010-06-25_clone (not active)		
choline-trimethyl	x	
choline-truncated	x	5

New: A dialog will pop up that gives the user a blank metabolite form to fill out. Select the number of spins in the metabolite and the form will enable the appropriate chemical shift and j-coupling fields. Edit the fields appropriately and hit ACCEPT or Cancel. See the sample in the figure below.

Edit: The highlighted metabolite is opened in a metabolite form. Only the metabolite Name, and Comment are editable. The name is editable because Experiments save Metabolite references by UUID which are not editable. Use the "Clone" option to create a copy of a Metabolite that is editable.



Name:
UUID:
Creator:
Created: 25 June, 2010
Spins: 5

	Shift	Isotope
1	0.0	1H
2	0.0	1H
3	0.0	1H
4	0.0	1H
5	0.0	1H
6		
7		
8		
9		
10		
11		
12		
13		
14		

	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.0	0.0	0.0	0.0									
2		0.0	0.0	0.0									
3			0.0	0.0									
4				0.0									
5													
6													
7													
8													
9													
10													
11													
12													
13													

View: Similar to Edit but no fields are editable.

Clone: Select a metabolite in the list, hit clone and a copy of that metabolite is made that is now fully editable. The new metabolite has the name of the original metabolite followed by the date and the word "_clone".

Delete: Only metabolites that have not been used by an experiment may be deleted. This is because to reconstruct any given Experiment, that object must refer to the original list of metabolites used to create it. The "Use Count" column indicates if a metabolite is in use by an Experiment. If not in use by an Experiment, the highlighted metabolite in the list is deleted from the database.

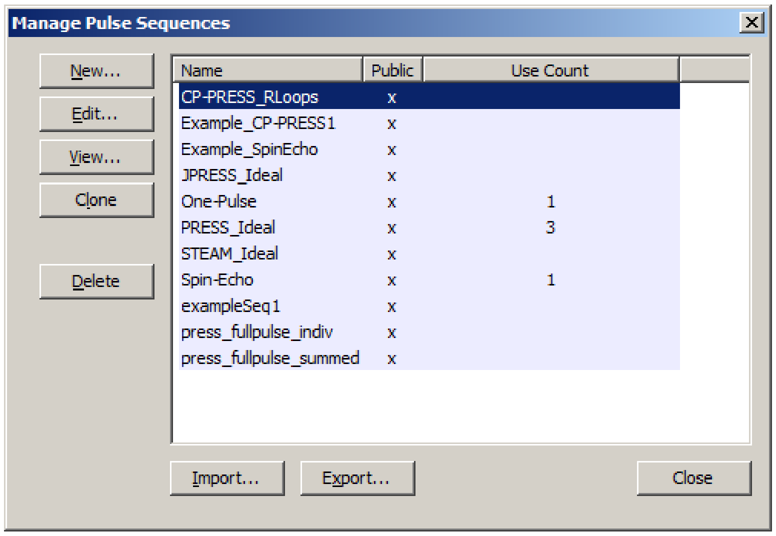
(De-)activate : When a metabolite is no longer being used, it can be set to a "deactivated" state where it no longer shows up in the Experiment Tab - Simulate metabolite list for use in new Experiments. This state is indicated in the Metabolite dialog by the word "(not active)" appended to the metabolite name in the list.

Import: Allows the user to select an XML file that contains a Metabolite. If the UUID in the file is unique, it is added to the Simulation database.

Export: The user selects an Metabolite from the list. A second dialog allows the user to browse for the output filename, select if output should be compressed and allows an additional export comment to be typed in. Note that the action of exporting an object caused it to be marked as "frozen" in the database. This means that no changes can be made. This is for the sake of consistency as results are shared. However, a frozen Metabolite can still be deleted from the database if needed. This file can be imported into another Vespa-Simulation installation using the Import function.

5.3 Manage Pulse Sequences dialog

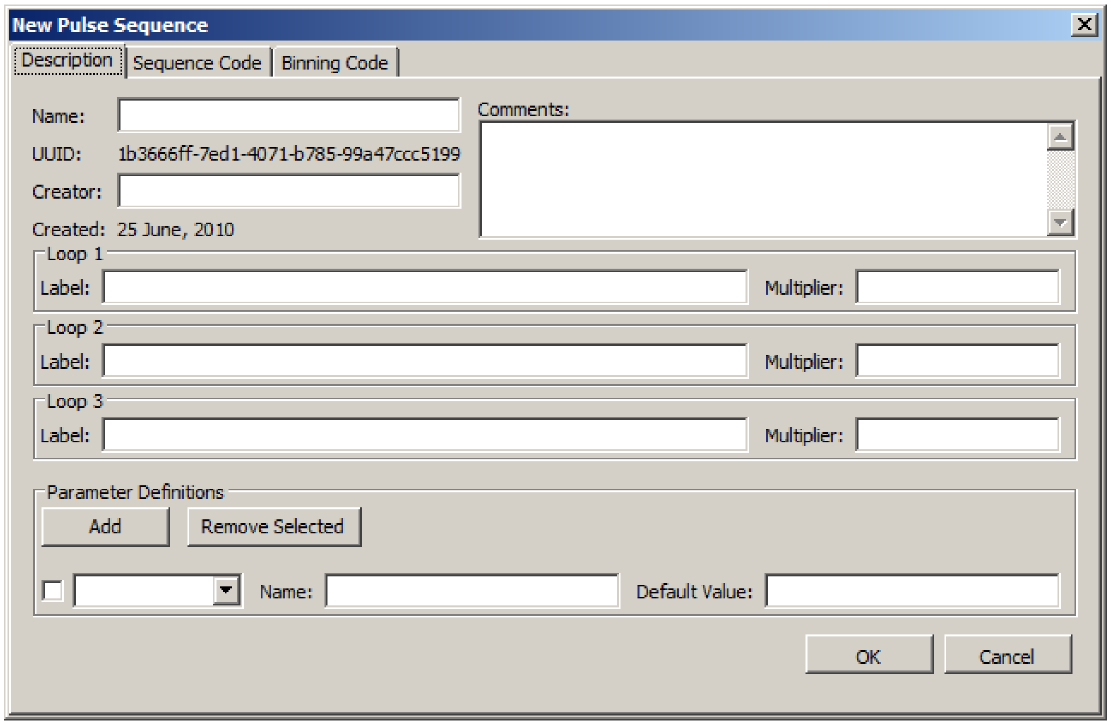
Access this dialog by clicking on the **Management→Manage Pulse Sequences** menu item. Actions that can be taken on the Pulse Sequences dialog include, New, Edit, View, Clone, Delete, Import and Export. An example of the widget used to display and edit pulse sequence information is shown. The "Public" column indicates if a sequence has ever been exported (or imported from someone else). This means that it should not be edited. The "Use Count" column indicates how many local Experiments use this sequence. While in use by any Experiments, the sequence can not be deleted.



Name	Public	Use Count
CP-PRESS_RLoops	x	
Example_CP-PRESS1	x	
Example_SpinEcho	x	
JPRESS_Ideal	x	
One-Pulse	x	1
PRESS_Ideal	x	3
STEAM_Ideal	x	
Spin-Echo	x	1
exampleSeq1	x	
press_fullpulse_indiv	x	
press_fullpulse_summed	x	

As mentioned already, the Vespa-Simulation program enables users to create their own pulse sequences that can then be saved in the database. The input parameters for user functions are very specific and are described in more detail in Appendix XX. This section describes a couple of ways that external sequences can be included into the Vespa-Simulation program

New: A widget pops up that allows the user to describe to Vespa-Simulation the PyGamma code to run for a pulse sequence and how to lay out the Experiment Tab - Simulate window to display standard and user-defined parameters. The New Pulse Sequence widget is shown below. Please note that there are 3 tabs affiliated with this dialog: Description, Sequence Code and Binning Code. All three tabs must be filled out properly to successfully enter a new pulse sequence.



UUID: 1b3666ff-7ed1-4071-b785-99a47ccc5199

Created: 25 June, 2010

Description Tab

Name: This is how the Experiment Tab - Simulate lists the pulse sequence in the drop-list.

Creator: The name of the person creating the pulse sequnce

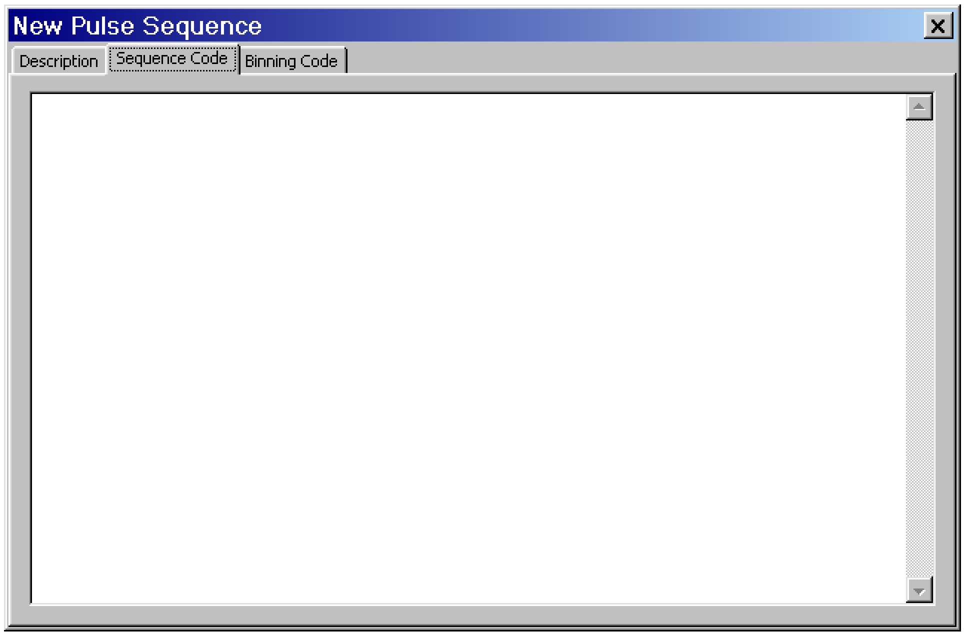
Loop Labels and Multipliers: When the pulse sequence is called, it can make use of up to three looping variables to create a variety of conditions for investigating metabolite behavior. In the Loop1, Loop2 and Loop3 rows the user gives information that allows Simulation to parse these loop variables. The Label field is a string used in creating the Experiment Tab - Simulate window that describe these loops. An example would be “TE [ms]” for a spin echo experiment. In this case, the user could also set the Mult field equal to 1000 to indicate that the Vespa-Simulation program should divide the values given for the Loop-Start and Step values before passing them to the pulse sequence function (unless the user specifically writes their function to process these variables as millisecond values, then the Mult would remain 1.0).

Additional Parameter Definitions: The user may add/remove additional parameters expected by the external pulse sequence using the respective buttons. Additional widgets will appear as requested. Each parameter is described by three fields: a data type drop-list, "Name" string and "Default Value" string. The name field will be used by the Experiment Tab - Simulate window as a label to describe this field in its far right hand column when the pulse sequence is selected for an Experiment. The corresponding default value is the value that is displayed in that field. As described in Appendix XX, the label also serves as a key in the control dictionary sent to run the actual PyGamma simulation, which can be referenced to return the user set value.

NB. By selecting a data type in the drop down menu, the user will be forced to enter that field as that variable type, and will be passed to the PyGamma simulation as that type. Please select your default types and values accordingly.

Sequence Code Tab

This is a text window in which PyGamma code can be pasted. As defined in more detail in Appendix XX, this code is executed using the Python "exec()" command and so should be constructed as if it were going to be run in-place with appropriate space/tab layout.



Binning Code Tab

This is a text window like the Sequence Code tab in which PyGamma code can be pasted. The default binning code, as defined in Appendix XX, is already in place when the New Sequence dialog is created, but can be deleted/edited as the user wishes. As defined (again) in more detail in Appendix XX, this code is executed using the Python "exec()" command immediately following the Sequence Code and so should be constructed as if it were going to be run in-place with appropriate space/tab layout.

Edit: The highlighted sequence is opened in a form similar to the New Sequence dialog. Only the metabolite Name, and Comment are editable. The name is editable because Experiments save Pulse Sequence references by UUID which are not editable. Use the "Clone" option to create a copy of a Pulse Sequence that is editable.

View: Similar to Edit but no fields are editable.

Clone: Select a sequence in the list, hit clone and a copy of that sequence is made that is now fully editable. The new sequence has the name of the original sequence followed by the date and the word "_clone".

Delete: Only sequences that have not been used by an experiment may be deleted. This is because to reconstruct any given Experiment, that object must refer to the original sequence used to create it. The "Use Count" column indicates if a sequence is in use by an Experiment. If not in use by an Experiment, the highlighted sequence in the list is deleted from the database.

Import: Allows the user to select an XML file that contains a Pulse Sequence. If the UUID in the file is unique, it is added to the Simulation database.

Export: The user selects a Pulse Sequence from the list. A second dialog allows the user to browse for the output filename, select if output should be compressed and allows an additional export comment to be typed in. Note that the action of exporting an object causes it to be marked as “frozen” in the database. This means that it can not be changed. This is for the sake of consistency as results are shared. However, a frozen Pulse Sequence can still be deleted from the database if needed. This file can be imported into another Vespa-Simulation installation using the Import function.

6. Results Output

6.1 Results output into standard text editor

On the Vespa-Simulation window menu bar, each Experiment Tab has its own menu item added to the menu bar. These are named the same as the Experiment Tabs, ie. Experiment1, Experiment2, etc. Select the **Experiment2→Output→Text Results** option and a tab-delineated text description of the Experiment is created and loaded into the local computer’s standard text editor. On Windows, this is typically Notepad. From here the user can save it wherever they please. NB. This command can also be launched from the Experiment Tab – Visualize sub-tab using the ASCII Results button.

The first section of the text file describes the settings of the Experiment. Metabolite simulations are saved as a collection of lines with amplitude, PPM and phase that can be used to recreate a time domain spectrum. Each line contains: metabolite Name, loop1_value, loop2_value, loop3_value, line_number, PPM, area and phase (deg). The index_loop variables may be set to other than 0 if the Experiment contains multiple steps in pulse sequence timings. E.g. an Experiment could run NAA, Cr and Cho for 10 TE values, with TE1 being held fixed and TE2 having 10 values. In the output file, loop1_index would be fixed and loop2_index would increment 10 times. The metabolite Name(s) would repeat 10 times as well, as loop2_value is incremented. In this way, a 2D Experiment is flattened into a 1D output file.

```
--- Experiment 9a146ac7-c47d-4ae2-b7b2-961e942d7d18 ---
Name: Example OnePulse Data
Public: True
Created: 2010-03-24T16:20:18
Comment (abbr.): Simulation for baseline GAVA database
PI: bsoher
Parameters:
b0: 64.000000
Peak Search PPM low/high: 0.000000 / 10.000000
Blend tol. PPM/phase: 0.001500 / 50.000000
Pulse seq.: bf0b302c-ce1f-46c9-b852-0e7c6b77f95c (One-Pulse)
3 Metabolites: aspartate, choline-truncated, creatine
1 Simulations: (not shown)

Simulation Results
---------------------------------------------------------------------------

aspartate      0.0     0.0     0.0     0       2.3706  0.03836 0.0
aspartate      0.0     0.0     0.0     1       2.49372 0.02196 0.0
aspartate      0.0     0.0     0.0     2       2.64232 0.409   0.0
aspartate      0.0     0.0     0.0     3       2.70787 0.42219 0.0
aspartate      0.0     0.0     0.0     4       2.76544 0.52731 0.0
aspartate      0.0     0.0     0.0     5       2.78347 0.5175  0.0
aspartate      0.0     0.0     0.0     6       2.97959 0.04772 0.0
aspartate      0.0     0.0     0.0     7       3.05519 0.01597 0.0
aspartate      0.0     0.0     0.0     8       3.58274 0.00563 0.0
aspartate      0.0     0.0     0.0     9       3.79689 0.29328 0.0
aspartate      0.0     0.0     0.0     10      3.87249 0.25374 0.0
aspartate      0.0     0.0     0.0     11      3.92001 0.23456 0.0
aspartate      0.0     0.0     0.0     12      3.99561 0.21054 0.0
aspartate      0.0     0.0     0.0     13      4.20976 0.00225 0.0
choline-truncated      0.0     0.0     0.0     0       3.185   3.0     0.0
creatine       0.0     0.0     0.0     0       3.027   3.0     0.0
creatine       0.0     0.0     0.0     1       3.913   2.0     0.0
creatine       0.0     0.0     0.0     2       6.649   1.0     0.0
```

6.2 Plot results to image file formats

Results in the 1D/StackPlot, Integral Plot and Contour Plot windows can all be saved to file in PNG (portable network graphic), PDF (portable document file) or EPS (encapsulated postscript) formats to save the results as an image. On the Vespa-Simulation window menu bar, each Experiment Tab has its own menu item added to the menu bar. These are named the same as the Experiment Tabs, ie. Experiment1, Experiment2, etc. Select the **Experiment2→Output→** option and further select either the **1D/StackPlot**, **IntegralPlot** or **ContourPlot** menu item. Finally, select either **Plot to PNG**, **Plot to PDF** or **Plot to EPS** item. The user will be prompted to pick an output filename to which will be appended the appropriate suffix.

6.3 Plot results to vector graphics formats

Results in the 1D/StackPlot, Integral Plot and Contour Plot windows can all be save to file in SVG (scalable vector graphics) or EPS (encapsulated postscript) formats to save the results as a vector graphics file that can be decomposed into various parts. This is particularly desirable when creating graphics in PowerPoint or other drawing programs. At the time of writing this, only the EPS files were inherently readable into PowerPoint.

On the Vespa-Simulation window menu bar, each Experiment Tab has its own menu item added to the menu bar. These are named the same as the Experiment Tabs, ie. Experiment1, Experiment2, etc. Select the **Experiment2→Output→** option and further select either the **1D/StackPlot**, **IntegralPlot** or **ContourPlot** menu item. Finally, select either **Plot to SVG**, or **Plot to EPS** item. The user will be prompted to pick an output filename to which will be appended the appropriate suffix.

Appendix A. Pulse Sequence Design

A.1 What's under the hood?

Vespa-Simulation Basic Concepts

This is a combination of logical concepts and limitations that determine how Simulation works. These rules are enforced through the application and, to some extent, the database.

The main objects in the system are experiments, simulations, spectra, pulse sequences and metabolites. Experiments are the primary objects; everything else is secondary. Here's how they're related --

- Each experiment has zero to many simulations. Simulations are the whole point of an experiment, and there's not much to an experiment besides the metatdata that defines the simulations. Since entering the experiment metadata is pretty trivial, we don't let users save experiments that define zero simulations. Experiments with zero simulations can exist, but only in memory. They are never saved to the database or an export file.

- Each experiment makes use of and refers to exactly one pulse sequence, but the experiment may define one or more timing sets for the pulse sequence.

- Each simulation creates one spectrum.

- Each spectrum has zero or more lines. Zero is an unusual case, but possible.

- Each spectrum line has a one PPM, area and phase value in it.

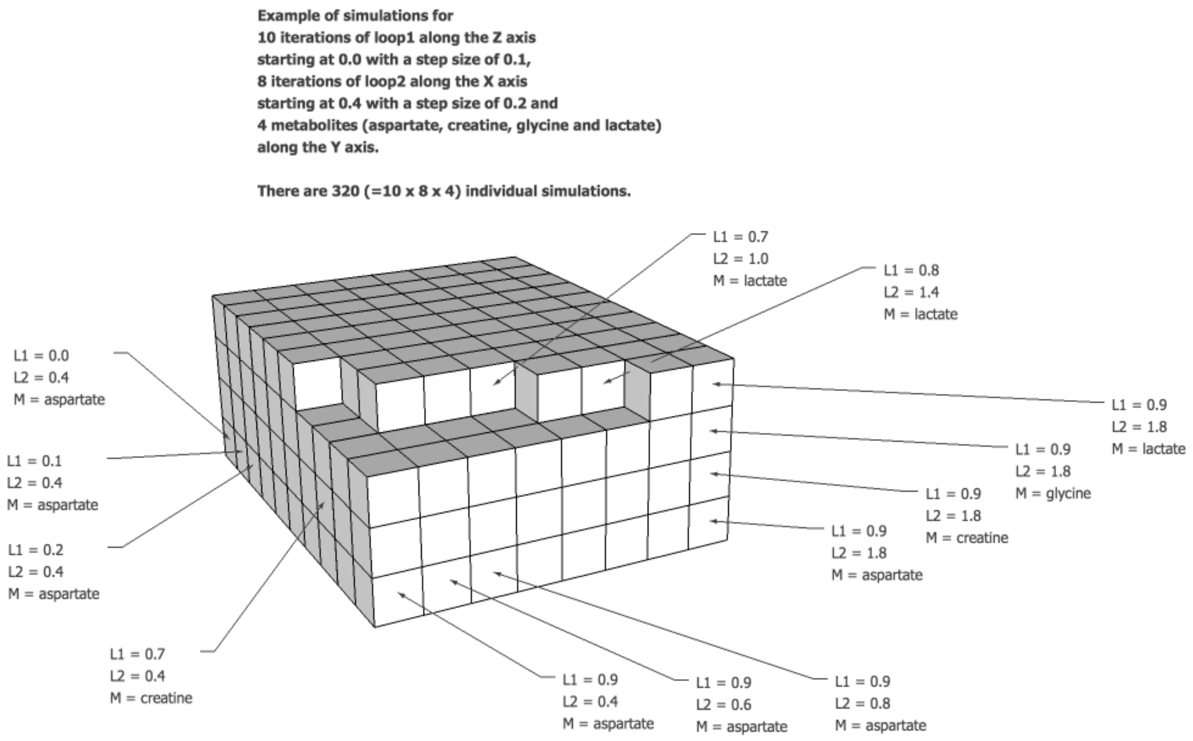
We expect users to share data via Simulation's export and import functions. For this reason, several of Simulation's objects (experiments, pulse sequences and metabolites) have universally unique ids (UUIDs) rather than just ordinary integer ids.

Experiments

Experiments are the main focus of the Simulation application. An experiment's *raison d'etre* is to run a set of simulations. This set of simulations is the experiment's *results space*.

Currently, that space is defined by one to four nested loops. The first loop covers the list of metabolites the user has involved in the experiment. The other one, two or three loops are user-defined lists of numbers.

The figure below is a visual representation of a 3D results space (one set of metabs and two lists of user-defined numbers). For clarity we do not show the 4th dimension (a.k.a. the last user defined loop) as stacks of cubes are hard to visualize.


Example of simulations for
10 iterations of loop1 along the Z axis
starting at 0.0 with a step size of 0.1,
8 iterations of loop2 along the X axis
starting at 0.4 with a step size of 0.2 and
4 metabolites (aspartate, creatine, glycine and lactate)
along the Y axis.
There are 320 (=10 x 8 x 4) individual simulations.
L1 = 0.7 L2 = 1.0 M = lactate
L1 = 0.8 L2 = 1.4 M = lactate
L1 = 0.0 L2 = 0.4 M = aspartate
L1 = 0.9 L2 = 1.8 M = lactate
L1 = 0.9 L2 = 1.8 M = glycine
L1 = 0.1 L2 = 0.4 M = aspartate
L1 = 0.9 L2 = 1.8 M = creatine
L1 = 0.9 L2 = 1.8 M = aspartate
L1 = 0.2 L2 = 0.4 M = aspartate
L1 = 0.7 L2 = 0.4 M = creatine
L1 = 0.9 L2 = 0.4 M = aspartate
L1 = 0.9 L2 = 0.6 M = aspartate
L1 = 0.9 L2 = 0.8 M = aspartate

Simulations themselves know nothing about one another and are agnostic to the order in which they're run. Thus, while the existing code is geared towards generating a very regular results space that we iterate over in a very straightforward order, more complex result spaces and iteration orders are possible. The sky's the limit, really, provided you can dream up a GUI that allows users to describe the results space.

A few other “rules” of note:

- Once an experiment has been saved, the following attributes become read-only: pulse sequence, investigator, user parameters, b0, isotope, peak_search_ppm_low, peak_search_ppm_high, blend_tolerance_ppm, blend_tolerance_phase.

- One can associate additional metabolites with an experiment, but once it is associated and the experiment is saved, the metabolite remains with the experiment forever. In other words, a metabolite can't be removed from a saved experiment.

- An experiment's b0 value is always stored in megahertz.

The take-home lesson from this section is that the Vespa-Simulation application provides 4 dynamic (looping) variables and 12 standard static variables to each spectral simulation that is run. In the example below, we will specify what these are and how they can typically be used. In the second example below, we will specify how user defined static variables can also be passed into spectral simulations.

A.2 Creating a Pulse Sequence without Extra Parameters

A.2.1 How to create a "One-Pulse" pulse sequence

The most important thing to remember in pulse sequence design is that no matter the size of your looping variables, each distinct set of pulse sequence parameters are sent to their spectral simulation independent of all others. To achieve this, a list of "control dictionaries" is created to store each distinct set of parameters. Then each dictionary is sent to a function that executes the PyGamma spectral simulation that it describes. On completion of each dictionary execution, lists of areas, ppms, phases and start/finish time stamps are returned and stored in the database.

The standard variables that are stored as key:value pairs in the control dictionary include:

'field' – (float) main B0 field strength in MHz

'sequence_code' – (string) PyGamma code string executed using exec() to perform the simulation

'binning_code' – (string) PyGamma code string executed using exec() immediately after the sequence_code to extract the areas, ppms and phases from the transition table.

'peak_search_ppm_low' – (float) range in ppm to be searched in binning code (see below)

'peak_search_ppm_high' – (float) range in ppm to be searched in binning code (see below)

'blend_tolerance_ppm' – (float) width of bins in ppm into which similar lines can be combined (see below)

'blend_tolerance_phase' – (float) width of bins in phase degrees into which similar lines can be combined (see below)

'dims – (list) this list contains the values of the 4 loops as set for this particular simulation. Specifically, dims[0] is a string containing the metabolite name, dims[1] dims[2] and dims[3] contain the float values of the three counting loops.

'met_iso – (list) string value for the isotope of each spin in the current metabolite

'met_cs – (list) float ppm value for chemical shift of each spin in the current metabolite

'met_js – (list) float ppm value for J-couplings of each spin pair in the current metabolite

'nspins – (int) number of spins in the metabolite (for convenience)

Inside the execution function there are a number of code bits that are automatically provided. One is the "import pygamma as pg" statement. Another takes the field, isotopes, chemical shifts and j-coupling values from the control dictionary and creates a PyGamma spin_system variable called "sys". However, this sys variable could be over-written by user code as needed, it is only provided as a convenience.

The One-Pulse Example

Here is the PyGamma code that is in the sequence_code string for the One-Pulse sequence:

```
H   = pg.Hcs(sys) + pg.HJ(sys)
D   = pg.Fm(sys, "1H")
```

```
ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
ACQ = ac

sigma  = pg.sigma_eq(sys)
sigma0 = pg.Iypuls(sys, sigma, "1H", 90.0)
mx     = ACQ.table(sigma0)
```

The first thing to note is that other than the "sys" spin_system variable, this pulse sequence does not make use of any of the variables in the control_dict dictionary. There are no loops in this simulation and no user-defined static parameters. For examples of how to use these variables see the following examples.

The final line of code demonstrates the one "output" code requirement if user is going to use the standard 'binning_code' provided by Simulation by default. The user must set up a transition table variable called "mx"

Here is the PyGamma code that is the default binning_code string which is automatically inserted into the Binning Code tab for each new pulse sequence definition, and subsequently is used in the One-Pulse sequence:

```
area   = pg.DoubleVector(0)
ppm    = pg.DoubleVector(0)
phase  = pg.DoubleVector(0)
field  = sys.Omega()
nspins = sys.spins()
tolppm = float(sim_dict["blend_tolerance_ppm"])
tolpha = float(sim_dict["blend_tolerance_phase"])
ppmlow = float(sim_dict["peak_search_ppm_low"])
ppmhi  = float(sim_dict["peak_search_ppm_high"])

bins = mx.calc_spectra(ppm, area, phase, field, nspins, \
                         tolppm, tolpha, ppmlow, ppmhi)

if bins > 0:
    area  = [i for i in area]
    ppm   = [i for i in ppm]
    phase = [i for i in phase]
else:
    area  = []
    ppm   = []
    phase = []
```

This code expects that there exists already in the namespace a variable named "mx" that is a PyGamma transition table. The actual binning code is written in C++ and accessed through a Swig mapping. This code creates three equal length lists called area, ppm and phase that are subsequently returned from the execution function to the main Simulation application for storage in the database.

If the user wants to write their own 'binning' code then they must follow these requirements. If the user is careful about what is provided/executed in the 'sequence_code' and subsequently used in the 'binning_code', there may be no need for the "mx" variable. But, the user must **always** return the three equal length lists named area, ppm and phase.

A.2.2 The “Ideal-PRESS” pulse sequence – typical use of standard parameters

Here is the PyGamma code that is in the sequence_code string for the PRESS_Ideal sequence:

```
te1 = sim_dict["dims"][1]
te2 = sim_dict["dims"][2]

H   = pg.Hcs(sys) + pg.HJ(sys)
D   = pg.Fm(sys, "1H")
ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
ACQ = ac

sigma0 = pg.sigma_eq(sys)
sigma1 = pg.Iypuls(sys, sigma0, "1H", 90.0)

Udelay = pg.prop(H, te1*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

sigma1 = pg.Iypuls(sys, sigma0, "1H", 180.0)

Udelay = pg.prop(H, (te1+te2)*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

sigma1 = pg.Iypuls(sys, sigma0, "1H", 180.0)

Udelay = pg.prop(H, te2*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

mx = ACQ.table(sigma0)
```

The first thing to note is that this pulse sequence accesses the “sys” spin_system variable and also the control_dict dictionary for the Loop1 and Loop2 values in the “`te1 = sim_dict["dims"][1]`” and “`te1 = sim_dict["dims"][2]`” lines. There are no user-defined static parameters. Similarly to the example above a transition table variable called “mx” is set up in the last line of code.

(Not shown) The default binning_code string is used to return the values from the transition table to the main Simulation program.

A.3 Creating a Pulse Sequence with Extra Parameters

A.3.1 The “PRESS-CP with Variable R-groups” Pulse Sequence

Here is the PyGamma code that is in the sequence_code string for the One-Pulse sequence:

```
te1     = sim_dict["dims"][1]
te2     = sim_dict["dims"][2]
rgroups = sim_dict["dims"][3]
pd90    = sim_dict["alpha/2 Pulse Time (sec)"]
ang90   = sim_dict["alpha/2 Pulse Angle (deg)"]
tauR    = sim_dict["tauR Pulse Duration (sec)"]
```

```
type    = sim_dict["PulseType(0-Ideal/1-Sand)"]

pd180   = pd90 * 2.0
ang180  = ang90 * 2.0

H   = pg.Hcs(sys) + pg.HJ(sys)
D   = pg.Fm(sys, "1H")
ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
ACQ = ac

sigma0 = pg.sigma_eq(sys)
sigma1 = pg.Iypuls(sys, sigma0, "1H", 90.0)

Udelay = pg.prop(H, te1*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

sigma1 = pg.Iypuls(sys, sigma0, "1H", 180.0)

Udelay = pg.prop(H, te1*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

sigma1 = sigma0

if type == 0:

    # using Ideal 180 pulses
    for k in range(rgroups):
        Udelay = pg.prop(H, tauR/2.0);
        sigma0 = pg.evolve(sigma1,Udelay);

        sigma1 = pg.Iypuls(sys,sigma0,180);

        Udelay = pg.prop(H, tauR/2.0);
        sigma0 = pg.evolve(sigma1,Udelay);

        sigma1 = sigma0;

else:

    for k in range(rgroups):

           # using 90-180-90 square 'Sandwich' pulses with MLEV16 phase cycling
        if (k % 4) == 0:

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = sigma0;

        if (k % 4) == 1:

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = sigma0;

        if (k % 4) == 2:

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
```

```
            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = sigma0;

        if (k % 4) == 3:

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, -ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  -ang90);

            Udelay = pg.prop(H, tauR);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);
            sigma0 = pg.Sypuls(sys, sigma1, H, "1H", offhz, pd180, ang180);
            sigma1 = pg.Sxpuls(sys, sigma0, H, "1H", offhz, pd90,  ang90);

            Udelay = pg.prop(H, tauR/2.0);
            sigma0 = pg.evolve(sigma1,Udelay);

            sigma1 = sigma0;


Udelay = pg.prop(H, te2*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

sigma1 = pg.Iypuls(sys, sigma0, "1H", 180.0)

Udelay = pg.prop(H, te2*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

mx = ACQ.table(sigma0)
```

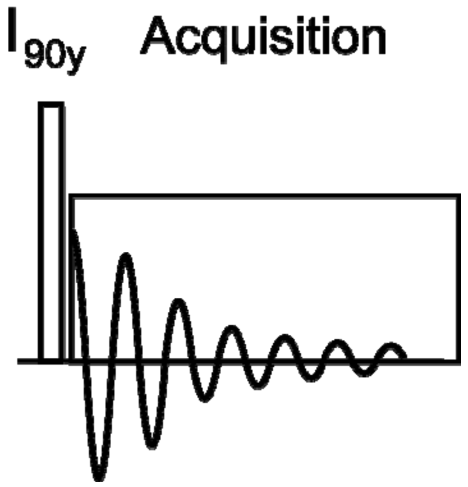
The pulse sequence accesses the "sys" spin_system variable. The first seven lines of code are good examples of how to access the control_dict dictionary for all three loop parameters and some user-defined static parameters. Note that the dictionary key for each user-defined parameter is just the label from the Experiment Tab – Simulate sub-tab panel for each additional parameter, e.g. the '`pd90 = sim_dict["alpha/2 Pulse Time (sec)"]`' line. Similarly to the examples above a transition table variable called "mx" is set up in the last line of code.
Also of note in this example is the fact that typical Python control structures can be used in these sequence_code strings, for loops, if statements, etc. However, extreme care should be taken to have consistent spacing and (lack of) tabs in the code that is pasted into the new pulse sequence dialog tab.

Appendix B. Pulse Sequence Diagrams

This section provides some basic information about the standard simulated pulse sequences that are provided as part of the Vespa distribution. The full PyGamma code for each pulse sequence can be accessed through the Pulse Sequence Management Dialog widget using the View or Edit functions.

B.1 one-pulse

B.1.1 Sequence Diagram



B.1.2 Loop Variable 1,2,3 Descriptions

Loop1 – not used

Loop2 – not used

Loop3 – not used

B.1.3 User Defined Static Parameters

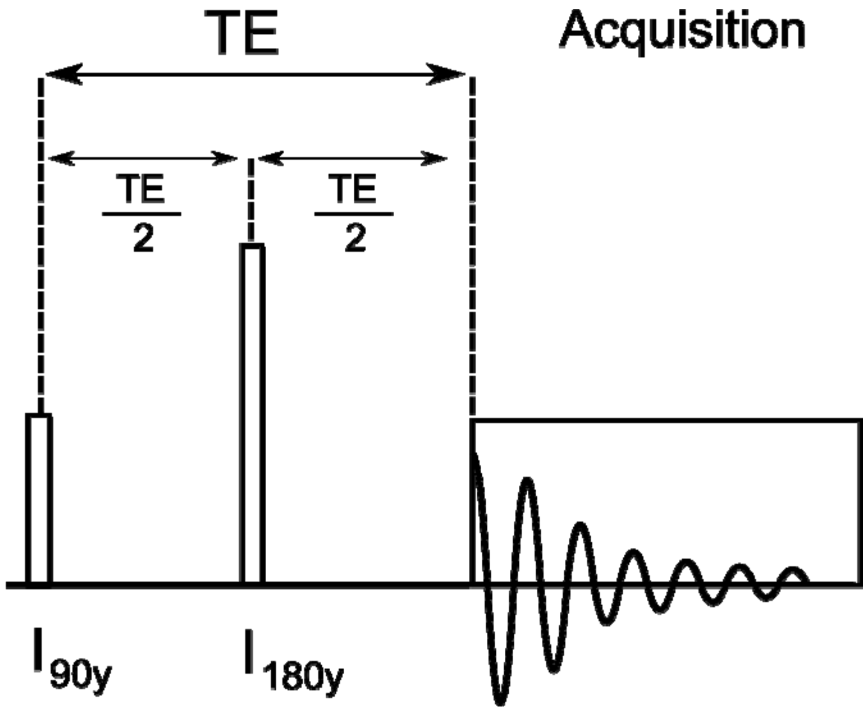
none

B.1.4 General Description

This is a simulation of a pulse and observe, or one-pulse, pulse sequence. The typical 90y degree hard pulse is modeled by an ideal GAMMA pulse. Despite the slight spacing in the sequence diagram, there is no evolution period after the excitation pulse prior to transition table acquisition.

B.2 spin-echo

B.2.1 Sequence Diagram



B.2.2 Loop Variable 1,2,3 Descriptions

Loop1 – Describes the number of TE values to loop over in [ms].

Loop2 – not used

Loop3 – not used

B.2.3 User Defined Static Parameters

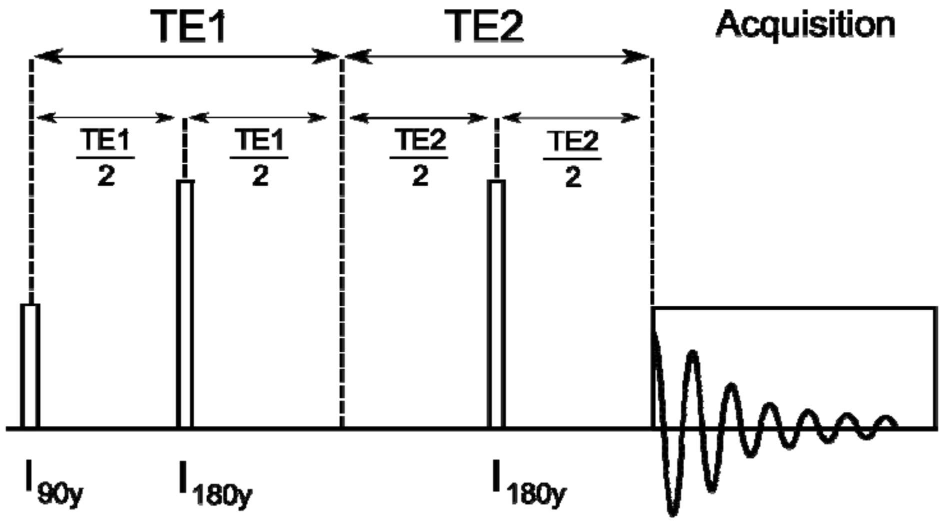
none

B.2.4 General Description

This is a simulation of a spin-echo sequence using ideal GAMMA pulses for the 90y and 180y localization pulses.

B.3 PRESS_Ideal

B.3.1 Sequence Diagram



B.3.2 Loop Variable 1,2,3 Descriptions

Loop1 – Describes the number of TE1 values to loop over in [ms].

Loop2 – Describes the number of TE2 values to loop over in [ms].

Loop3 – not used

Notes – Pulse sequence TE = TE1+TE2.

B.3.3 User Defined Static Parameters

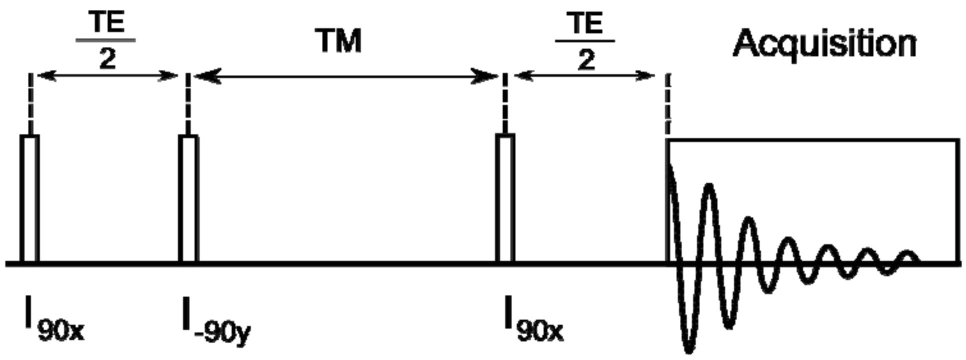
none

B.3.4 General Description

This is a simulation of a Point Resolved Spectroscopy (PRESS). The typical 90-180-180 localization pulses of the PRESS sequence are modeled by ideal GAMMA pulses. The TE1 period is controlled by the settings of loop variable 1, the TE2 period is controlled by the settings of loop variable 2, thus either a symmetric or asymmetric PRESS experiment can be simulated.

B.4 STEAM_Ideal

B.4.1 Sequence Diagram



B.4.2 Loop Variable 1,2,3 Descriptions

Loop1 – Describes the number of TE values to loop over in [ms].

Loop2 – Describes the number of TM values to loop over in [ms].

Loop3 – not used

B.4.3 User Defined Static Parameters

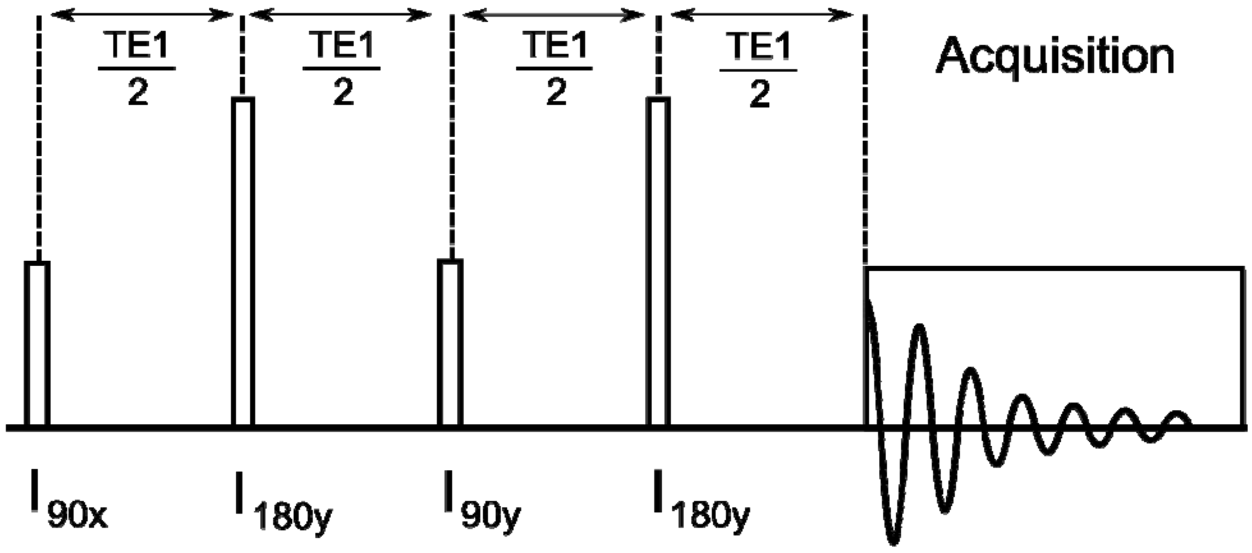
none

B.4.4 General Description

This is a simulation of a STimulated Excitation Acquisition Mode (STEAM) pulse sequence. The typical 90-90-90 pulses of the STEAM sequence are modeled by ideal GAMMA pulses. The total TE period is controlled by the settings of loop variable 1, the TM (mixing time) period is controlled by the settings of loop variable 2.

B.5 JPRESS_Ideal

B.5.1 Sequence Diagram



B.5.2 Loop Variable 1,2,3 Descriptions

Loop1 – Describes the number of TE1 values to loop over in [ms].

Loop2 – not used

Loop3 – not used

B.5.3 User Defined Static Parameters

none

B.5.4 General Description

This is a simulation of a J-PRESS pulse sequence. The typical 90-180-90-180 pulses of the JPRESS sequence are modeled by ideal GAMMA pulses. The total TE period is controlled by the settings of loop variable 1.