Vespa – Simulation
User Manual and Reference

Version 0.1.2

Release date: February 9th, 2011

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 enhances and extends 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 addresses 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 a simulated Experiment (consisting of one or more spectral simulations) from lists of metabolites and pulse sequences.
2) Store simulated 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 simulated Experiments
5) Output results in text or graphical format
6) Export/Import experiments, metabolites or pulse sequences from other users
7) Design and test their own PyGAMMA pulse sequences for addition to the list of pulse sequences available for use in Experiments.

What is an Experiment? An 'Experiment' consists of one or more spectral Simulations. Each Experiment 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 contains results for a single metabolite for one set of sequence timings. Each call to the PyGAMMA library produces results for a single Simulation. Vespa-Simulation loops through the spectral simulations for all timings and metabolites to completely fill out the Experiment's results.

There are a number of predefined pulse sequences in the Vespa-Simulation environment, and users can also design and test their own Python pulse sequence scripts using the PyGAMMA library. The database also contains prior information (current literature values) 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.

References: 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

This section assumes Vespa-Simulation has been downloaded and installed. See the Vespa Installation guide on the Vespa main project wiki for details on how to install the software and package dependencies. http://scion.duhs.duke.edu/vespa.

In the following, screenshots are based on running Simulation on the Windows OS, but aside from starting the program, the basic commands are the same on all platforms.

1. Overview – How to launch Vespa-Simulation

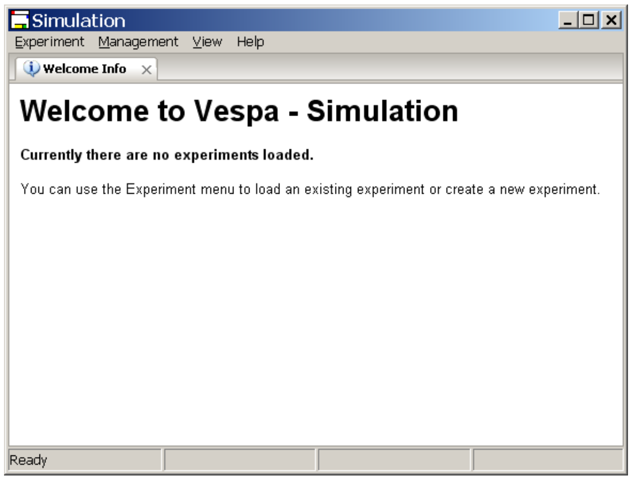
The usual case: If you installed Simulation by downloading and unzipping the package and by running "python setup.py install", then you should already have an Icon on your desktop called Simulation.

- *Double clicking on this icon will launch the application.*
- Note: The shortcuts are set up to launch Simulation without opening a shell/command window.

On all operating systems, but particularly on OS X and Linux one can also just open up a command window, go the directory vespa/simulation/src and type the following command:

```
python main.py
```

Shown below is the Vespa-Simulation main window as it appears on first opening. No actual Experiment windows are open, only the 'Welcome' banner is displayed.



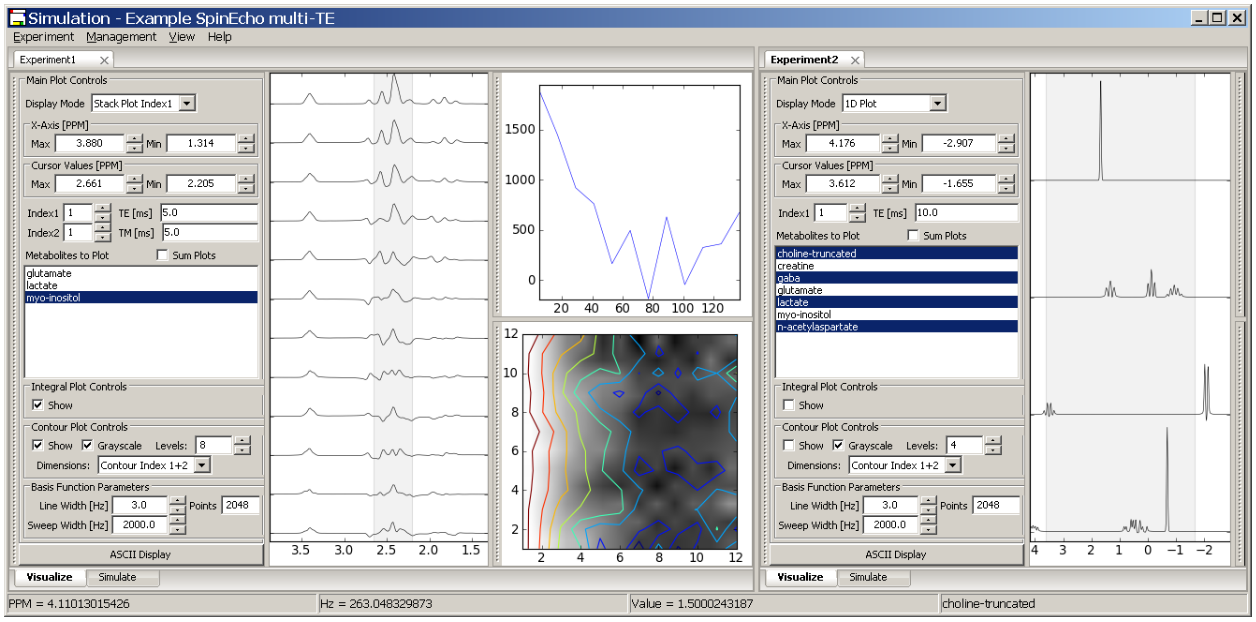
Welcome to Vespa - Simulation

Currently there are no experiments loaded.

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

Use the Experiment menu to open existing Experiments into tabs, or to create a tab for designing a 'new' spectral simulation Experiment.

Shown below 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.



Experiment1

Main Plot Controls
Display Mode: Stack Plot Index1
X-Axis [PPM]: Max 3.880, Min 1.314
Cursor Values [PPM]: Max 2.661, Min 2.205
Index1 1, TE [ms] 5.0
Index2 1, TM [ms] 5.0
Metabolites to Plot: glutamate, lactate, myo-inositol (selected); Sum Plots ☐
Integral Plot Controls: ☑ Show
Contour Plot Controls: ☑ Show ☑ Grayscale Levels: 8
Dimensions: Contour Index 1+2
Basis Function Parameters
Line Width [Hz] 3.0, Points 2048
Sweep Width [Hz] 2000.0

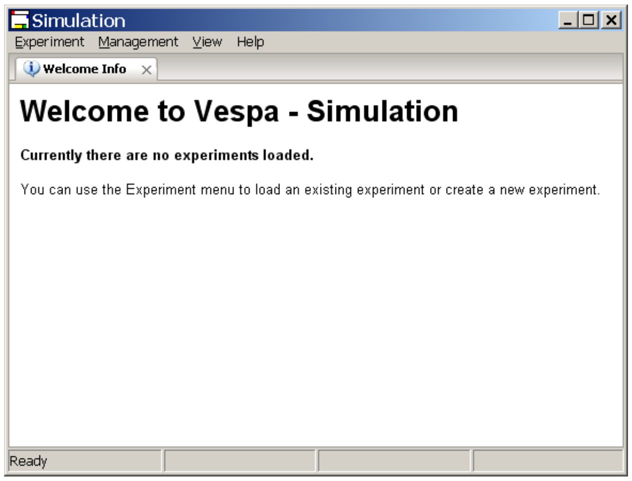
Experiment2

Main Plot Controls
Display Mode: 1D Plot
X-Axis [PPM]: Max 4.176, Min -2.907
Cursor Values [PPM]: Max 3.612, Min -1.655
Index1 1, TE [ms] 10.0
Metabolites to Plot: choline-truncated (selected), creatine, gaba (selected), glutamate, lactate (selected), myo-inositol, n-acetylaspartate (selected); Sum Plots ☐
Integral Plot Controls: ☐ Show
Contour Plot Controls: ☐ Show ☑ Grayscale Levels: 4
Dimensions: Contour Index 1+2
Basis Function Parameters
Line Width [Hz] 3.0, Points 2048
Sweep Width [Hz] 2000.0

PPM = 4.11013015426 | Hz = 263.048329873 | Value = 1.5000243187 | choline-truncated

2. The Simulation Main Window

This is a view of the main Vespa-Simulation user interface window. It is the first window that appears 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 input data and results from one Experiment. As described above, an Experiment is a group 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 and set it up and run it. In either case a tab will appear for each Experiment that is loaded or created. 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 in the interface throughout the program. It also reports short messages that reflect current processing while events are running.

On the Menu Bar

Experiment→New	Opens a new Experiment Tab in the Experiment Notebook.
Experiment →Copy Tab to New	This will open a new Experiment Tab and populate it with the same values that are listed in the current Experiment. No results are copied to the new tab. This is a short cut for varying simulation parameters to get different results and still being able to compare back to a previous results set without having to save them both to the data base.
Experiment→Open	Runs the Experiment Browser dialog, from which you can choose an Experiment from the database to open.
Experiment→Save	Saves the Experiment in the current tab to the data base. Note. Experiment results are not automatically saved to data base after the Run button is hit.
Experiment→Close	Closes current Experiment Tab. Will prompt for save if necessary.
Management →Manage Experiments	Launches the Manage Experiments dialog. Allows user to view, clone, delete, import and export Experiments.
Management →Manage Metabolites	Launches the Manage Metabolites 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.
View→<various>	Changes plot options in the Visualize sub-tab of the active Experiment tab, including: display a zero line, turn x-axis on/off or choose units, changing plot color, selecting data type or line shape, turning axes on/off for the Integral or Contour plot windows, and various output options for all plot windows.
Help→User Manual	Launches the user manual (from vespa/docs) into a PDF file reader.
Help→Simulation/Vespa Online Help	Online wiki for the Simulation application and Vespa project
Help→About	Giving credit where credit is due.

3. The Experiment Notebook

The Experiment Notebook is an "advanced user interface" widget (AUINotebook). What that means to you and me is a lot of flexibility: 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.

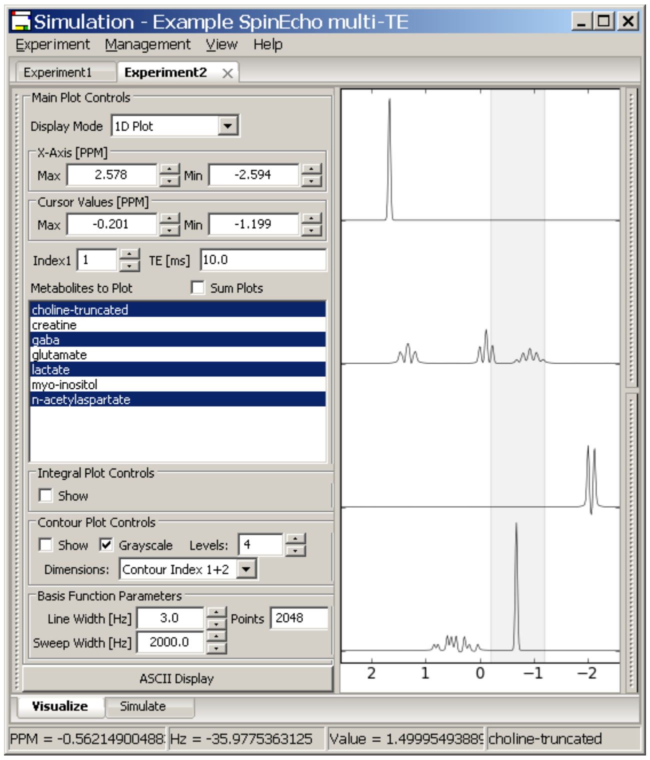
The Experiment Notebook can be populated with one or more Experiment Tabs, each of which contains the results of one Experiment. Tabs can be closed using the X box on the tab or with a middle-click on the tab itself. When a Tab is closed, the Experiment is removed from memory, but can be reloaded from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

4. The Experiment Tab

An Experiment Tab is a tabbed window 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 view the results of that run. It can also be used to load an existing Experiment from the database to view results, or to add more metabolites to the Experiment.

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

When a new Experiment is set up, there are no results to be displayed so the program defaults to the Simulate tab for New Experiments. When an existing Experiment is loaded, it typically contains results from simulations that have been run, so the program defaults to the Visualize tab.



Display Mode: 1D Plot
X-Axis [PPM] Max 2.578 Min -2.594
Cursor Values [PPM] Max -0.201 Min -1.199
Index1 1 TE [ms] 10.0
Metabolites to Plot ☐ Sum Plots
choline-truncated
creatine
gaba
glutamate
lactate
myo-inositol
n-acetylaspartate
☐ Show
☐ Show ☑ Grayscale Levels: 4
Dimensions: Contour Index 1+2
Line Width [Hz] 3.0 Points 2048
Sweep Width [Hz] 2000.0
PPM = -0.56214900488 Hz = -35.9775363125 Value = 1.49995493889 choline-truncated

A New Experiment is typically created, set up and run. Results from running an Experiment are only saved to the database when specifically requested by the user. The Visualize tab is updated to display results after each time the Run button is pushed on the Simulate tab (i.e. after each run). Experiments can be run multiple times, until it has been saved to the database. At that point it is considered ‘frozen’ and it can only be “run again” to add additional metabolites. The same parameters will be used for additional “add metabolite” runs.

The View menu on the main menu bar can be used to modify the display of the plots in the Visualize tab. The resulting modifications only affect the settings in the currently activated Experiment Tab. The following lists the functions on the View menu item:

On the Menu Bar

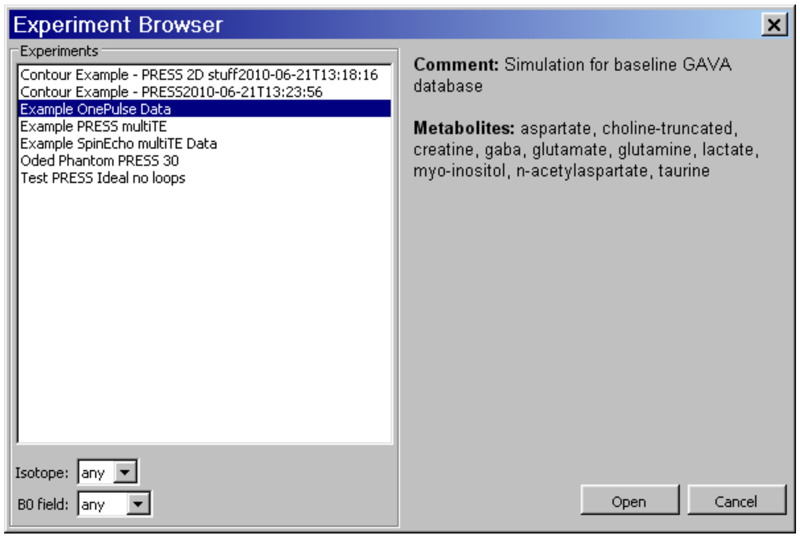
View (this menu affects the plots in the currently active Experiment tab)

→Show ZeroLine	toggle zero line off/on in 1D and stack display
→Xaxis →Show	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→Show x/y	toggles either x or y, or both axes on/off
→Show Contour Axes	toggles both axes on/off
→Output→1D/Stackplot	writes the plot, currently in the 1D or StackPlot canvas, to file as either PNG, SVG, EPS or PDF format
→Output→Integral Plot	writes the plot, currently in the Integral plot canvas, to file as either PNG, SVG, EPS or PDF format
→Output→Contour Plot	writes the 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 Experimental parameters and 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 the left. When an Experiment listed in the browser is clicked on once, its comment and metabolites are displayed on the right. Experiments can be sorted by the isotopes contained within the simulated metabolites. They can also be sorted by field strength (given in MHz).

When the Open button is clicked (or an Experiment’s name is double-clicked on), the program loads the information for that 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


Experiment Browser

Experiments

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

Isotope: any

B0 field: any

Comment: Simulation for baseline GAVA database

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

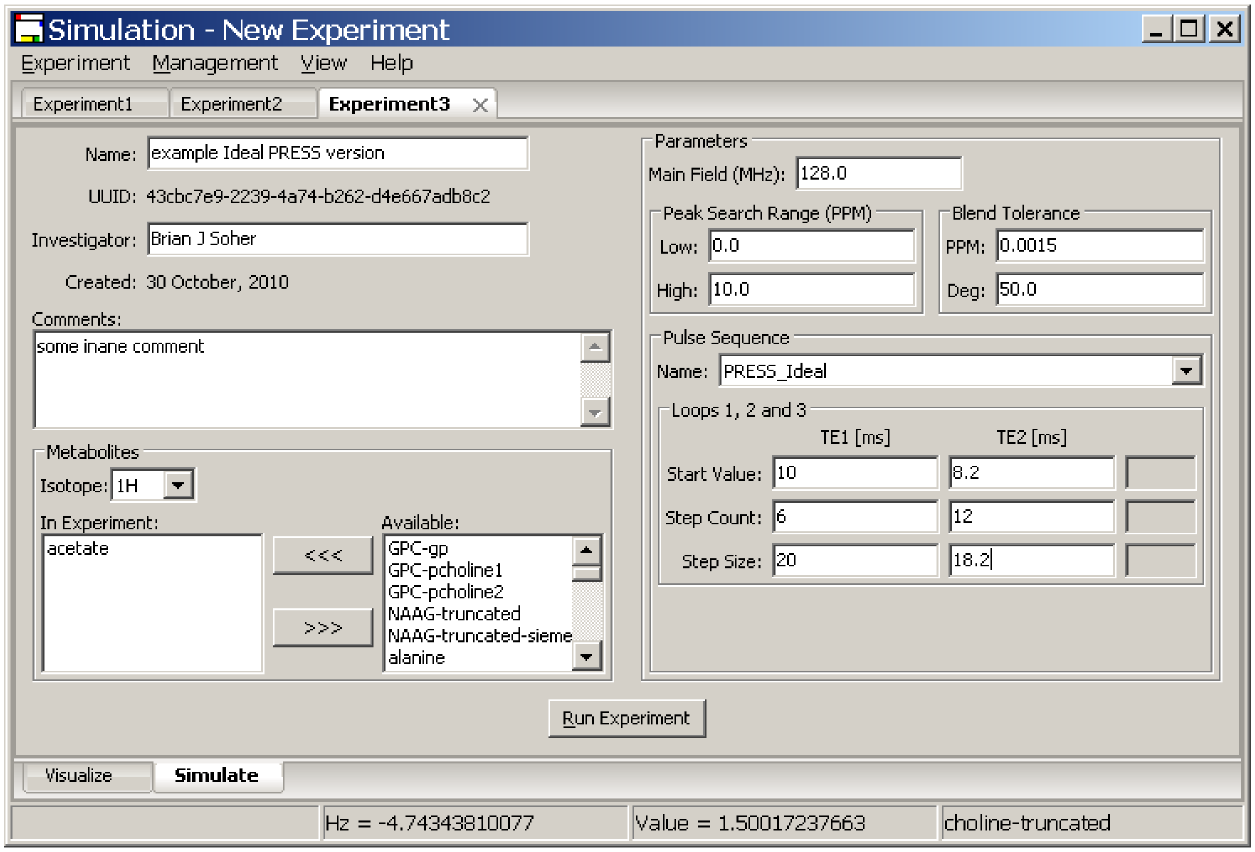
Visualization tab plots. N.B. In the case of a large Experiment, this may take a significant amount of time to calculate, but is indicated on the lower left of the status bar while calculating.

4.2 Running a new Experiment

As noted previously, 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 object.

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 available 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
UUID: 43cbc7e9-2239-4a74-b262-d4e667adb8c2
Investigator: Brian J Soher
Created: 30 October, 2010
Comments: some inane comment

Isotope: 1H
In Experiment: acetate
Available: GPC-gp, GPC-pcholine1, GPC-pcholine2, NAAG-truncated, NAAG-truncated-sieme, alanine

Main Field (MHz): 128.0
Peak Search Range (PPM) Low: 0.0 High: 10.0
Blend Tolerance PPM: 0.0015 Deg: 50.0
Pulse Sequence Name: PRESS_Ideal

	TE1 [ms]	TE2 [ms]	
Start Value:	10	8.2	
Step Count:	6	12	
Step Size:	20	18.2	

Hz = -4.74343810077 | Value = 1.50017237663 | choline-truncated

Simulation provides the user with four loop variables for use in their pulses sequences. This is covered in detail in Appendix A, 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 these values, start = 10.2, steps = 4, size = 2.0, that dimension would contain the following values [10.2, 12.2, 14.2, 16.2]. These values are passed directly to the user's PyGAMMA code and can be used in any fashion. One might use these values directly as sequence timing values where they represent [ms] timings between RF pulses. Another use might be as an integer series (e.g. [1,2,3,4,5,6]) indexing 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 in the figure is an example of a new Experiment tab configured for a PRESS simulation.

Note: Metabolite Peak Normalization and Blending

The transition tables calculated by the GAMMA density matrix simulations frequently contain a large number of transitions caused by degenerate splittings and other processes. At the conclusion of each simulation run a routine is called to extract lines from the transition table. These lines are then normalized using a closed form calculation based on the number of spins. To reduce the number of lines required for display, multiple lines are blended by binning them together based on their PPM locations and phases. The following parameters are used to customize these procedures:

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

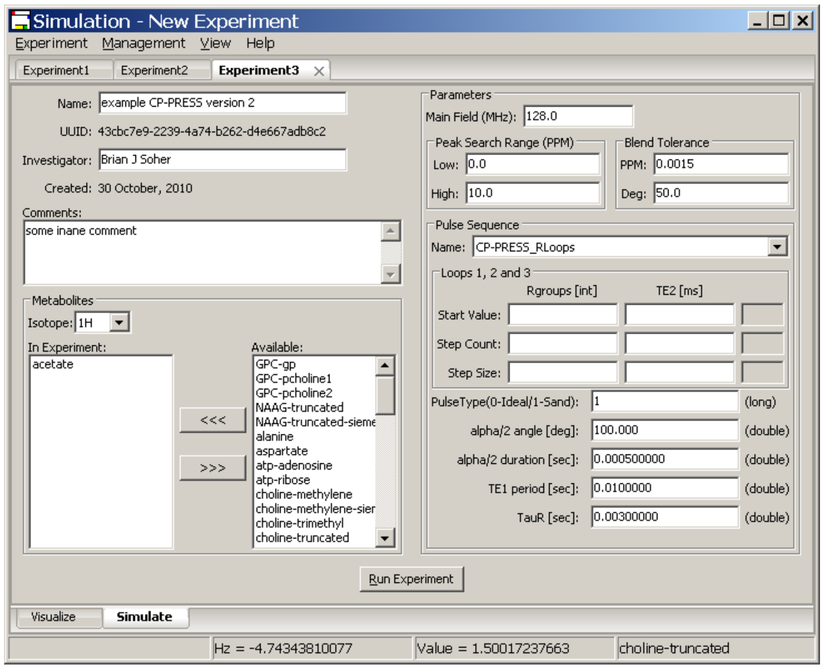
Peak Blending Tolerance (PPM and **Degrees):** the width of the bins (+/- in PPM and +/- in PhaseDegrees) that are used to blend the lines in the simulation. Lines that are included in the same bin are summed using complex addition based on Amplitude and Phase.

4.3 New Experiments with additional user defined parameters

A full explanation of how to create additional pulse sequences, with any additional parameters that may be required, is given in Appendix A. 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 to run an Experiment using a pulse 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 display input fields where the user can set the values for these additional parameters. These additional parameters



Name: example CP-PRESS version 2
UUID: 43cbc7e9-2239-4a74-b262-d4e667adb8c2
Investigator: Brian J Soher
Created: 30 October, 2010
Comments: some inane comment

Metabolites
Isotope: 1H
In Experiment: acetate
Available: GPC-gp, GPC-pcholine1, GPC-pcholine2, NAAG-truncated, NAAG-truncated-sieme, alanine, aspartate, atp-adenosine, atp-ribose, choline-methylene, choline-methylene-sier, choline-trimethyl, choline-truncated

Parameters
Main Field (MHz): 128.0
Peak Search Range (PPM) Low: 0.0 High: 10.0
Blend Tolerance PPM: 0.0015 Deg: 50.0

Pulse Sequence
Name: CP-PRESS_RLoops

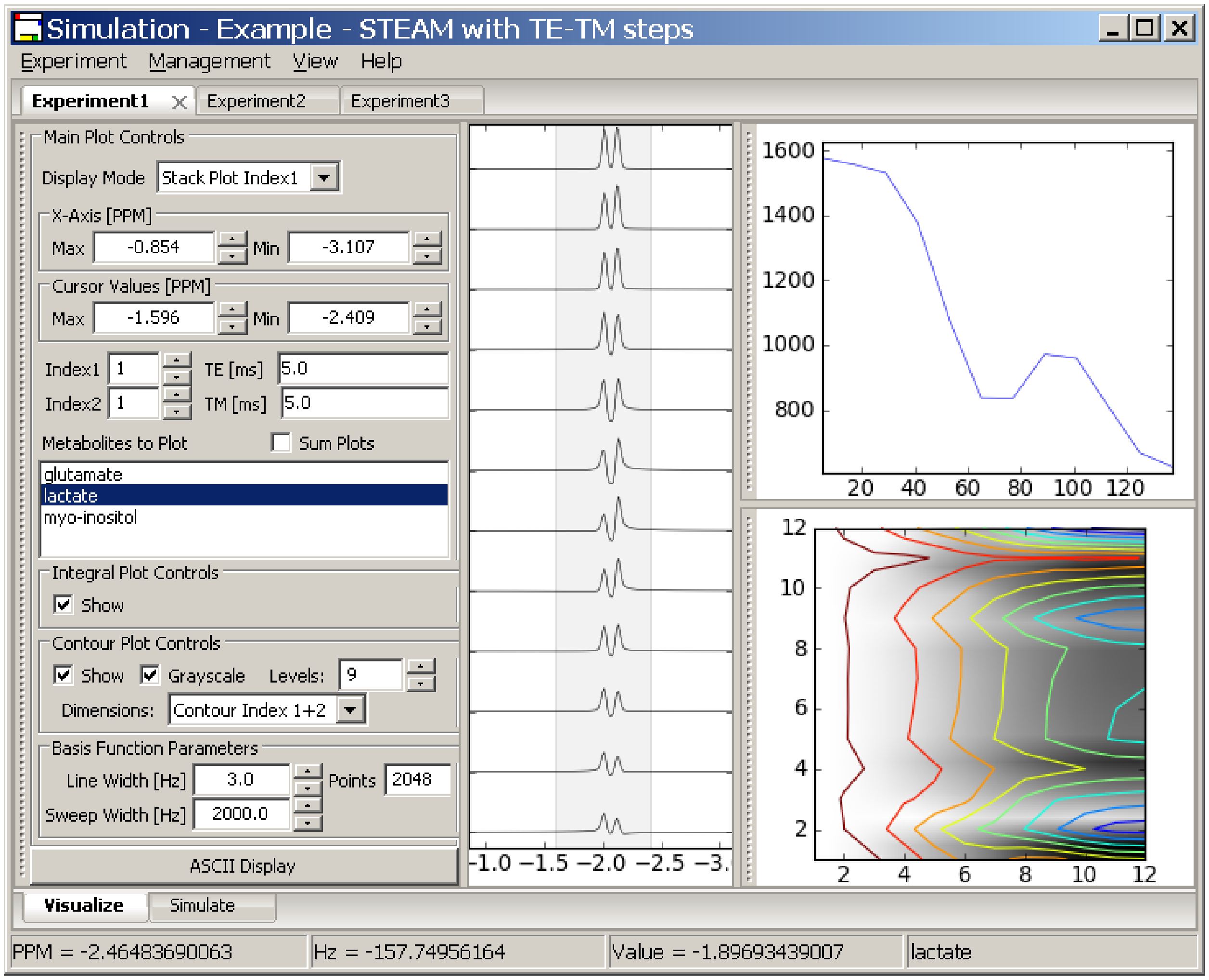
Loops 1, 2 and 3	Rgroups [int]	TE2 [ms]	
Start Value:			
Step Count:			
Step Size:			

PulseType(0-Ideal/1-Sand):	1	(long)
alpha/2 angle [deg]:	100.000	(double)
alpha/2 duration [sec]:	0.000500000	(double)
TE1 period [sec]:	0.0100000	(double)
TauR [sec]:	0.00300000	(double)

Hz = -4.74343810077 Value = 1.50017237663 choline-truncated are displayed in a list below the loop fields. Each line contains only one parameter description and a field to set a value. When appropriate, a default value is provided. Note: Data types are limited to String, Long or Float data types for data entry. The user is restricted to entering this type of data in any given field.

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 the types of available display are appropriately restricted. However, other pulse sequences can typically use most of the plot modes. The three plot modes for displaying results, 1D/StackPlot, Integral Plot and Contour Plot, are shown below:



Display Mode: Stack Plot Index1

X-Axis [PPM]: Max -0.854, Min -3.107

Cursor Values [PPM]: Max -1.596, Min -2.409

Index1: 1 TE [ms]: 5.0

Index2: 1 TM [ms]: 5.0

Metabolites to Plot: glutamate, lactate, myo-inositol
Sum Plots: ☐

Integral Plot Controls: ☑ Show

Contour Plot Controls: ☑ Show ☑ Grayscale Levels: 9
Dimensions: Contour Index 1+2

Basis Function Parameters: Line Width [Hz] 3.0 Points 2048
Sweep Width [Hz] 2000.0

PPM = -2.46483690063 | Hz = -157.74956164 | Value = -1.89693439007 | lactate

The **1D/StackPlot** window is always open and centered in the screen. The **Integral Plot** and the **Contour Plot** can be toggled on/off using the check box next to their names (though their windows remain ‘open’ whether they are being plotted or not). 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. This Zoom Span will display its range in a pale yellow that disappears when the left mouse is released.

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. This Cursor Span will display as a light gray span. Click in place with the right mouse button and the Cursor span will be turned off.

The cursor values are used to determine the "area under the peak" values that are plotted in the Integral and Contour windows. Changes to the cursor settings, either by mouse or in the respective widgets, 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 cursor span will be turned off.

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.

On the Visualize Widget

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 on 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.
Cursor 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 on 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, Loop2 and Loop3 dimensions for the various plot modes. As each Index widget is incremented, the sequence timing's actual value is shown in the adjoining field. If a given Experiment did not use a Loop dimension, that index is not displayed (e.g. you will often not see Index 3).

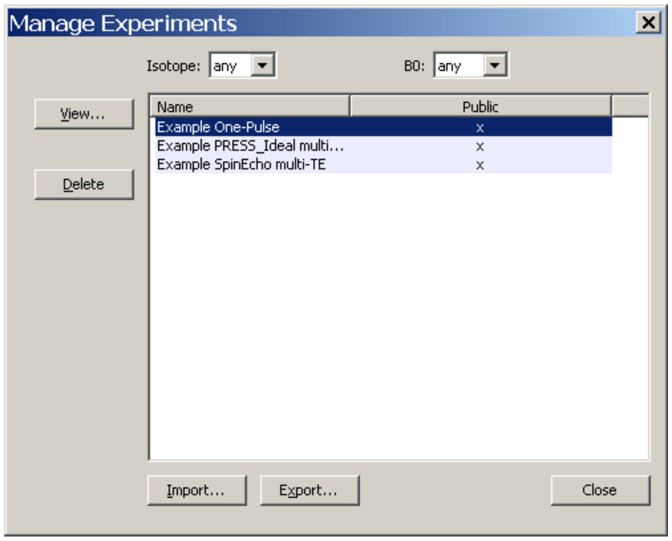
Metabolites to Plot	(list) A list of metabolites in the experiment that can be included in the display.
Sum Plots	Sums all metabolite plots selected (highlighted) in the list. For 1D display, this sums different metabolite spectra together. For Stack Plots the different sequence timings for one metabolite 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.
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.
Line Width	(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.
Points	The number of spectral points used to reconstruct the spectra.
ASCII Display	Displays the current Experiment results in text form. The information at the top is a summary of the Experiment parameters, which is 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 therefore allow the user to manage the data in the Simulation database, and to add new metabolite and pulse sequence information that can be used as prior information for simulation and processing. 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, Delete, Import or Export Experiments. These functions are summarized below.



Name	Public
Example One-Pulse	x
Example PRESS_Ideal multi...	x
Example SpinEcho multi-TE	x

View: Creates a brief textual description of the Experiment that is displayed in a native text editor for the platform being used. Use View→Output→Text Results menu item on the main menu bar with the Experiment loaded into a tab in the Notebook for a more detailed textual description of the Experiment and it's results.

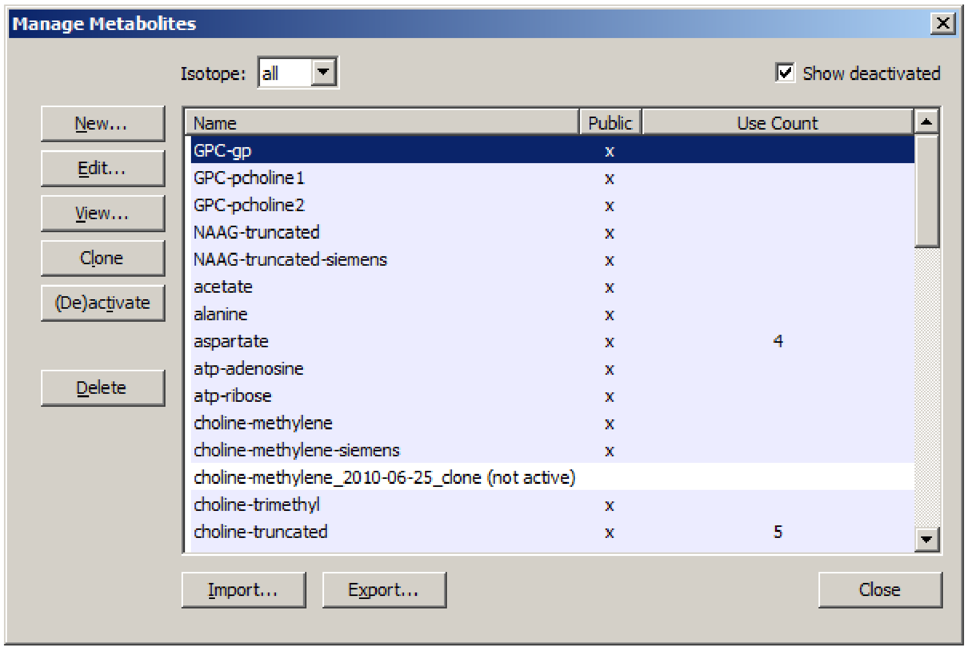
Delete: Removes the Experiment from the database.

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. The program asks if both parameters and results should be included in the 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 Experiment (or other objects) 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. If additional changes are desired a new Experiment, using the same Pulse Sequence object, can be created and edited.

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 *Manage Metabolites* window is shown below. The "Public" column indicates if a metabolite has ever been exported (or imported from someone else). If the public flag is set then it can 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.

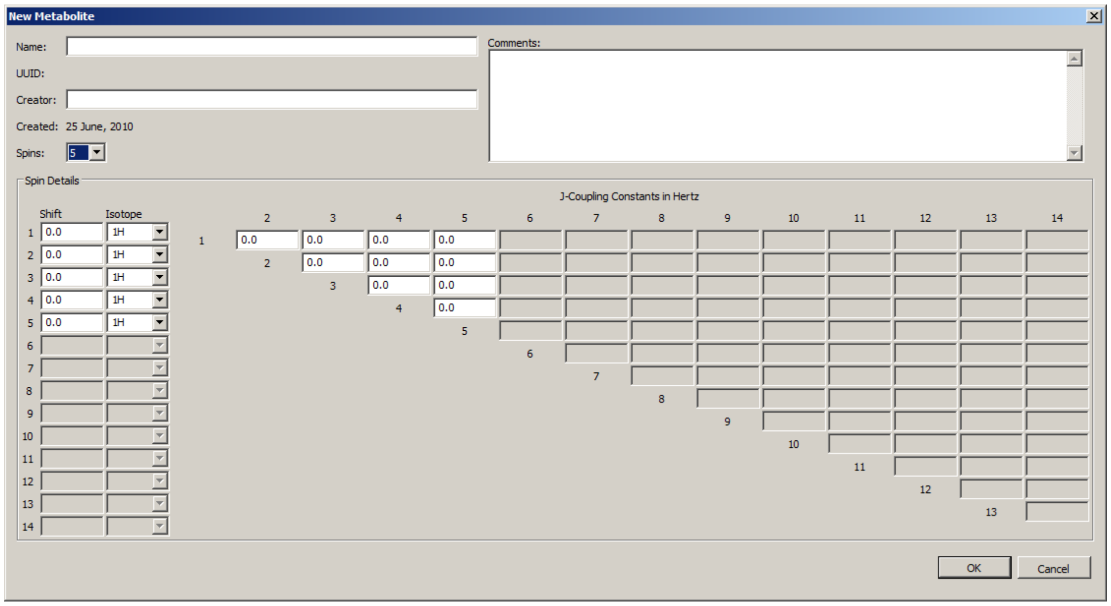


Isotope: all

[✓] Show deactivated

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.



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

	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													

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 fully editable.

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 can be 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.

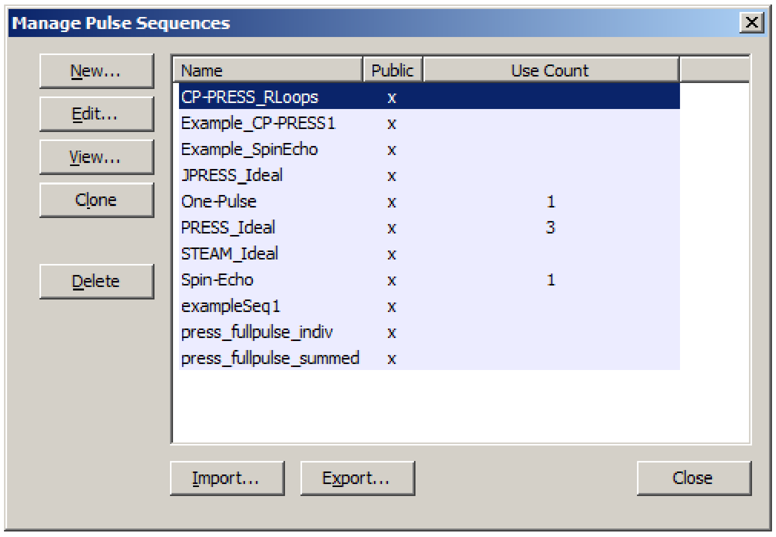
Import: Allows the user to select an XML file that contains a Metabolite. If the UUID in the file is unique, it is added to the Simulation database.

Export: The user selects a Metabolite from the list. A second dialog allows the user to browse for the output filename, select if the 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 no further changes can be made. This is for the sake of consistency when results are shared. However, a frozen Metabolite can still be deleted from the database if needed. The exported file can be imported into another Vespa-Simulation installation using the Import function.

Note. An interesting case for which one might want to create a new metabolite would be if one discovered during for example a long TE experiment that literature values for a particular metabolite were not adequately precise in terms of modeling the result of the experiment. One could obtain improved values via some combination of experimental and optimization methods, then clone the existing metabolite and enter the improved values. These improved values could later be submitted to the public VeSPA database, perhaps after publication of the results.

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 window used to display and edit pulse sequence information is shown. The New, Edit, View, Import and Export buttons all launch secondary dialogs as part of their functionality. Clone and Delete only affect the list in the main pulse sequence management dialog.

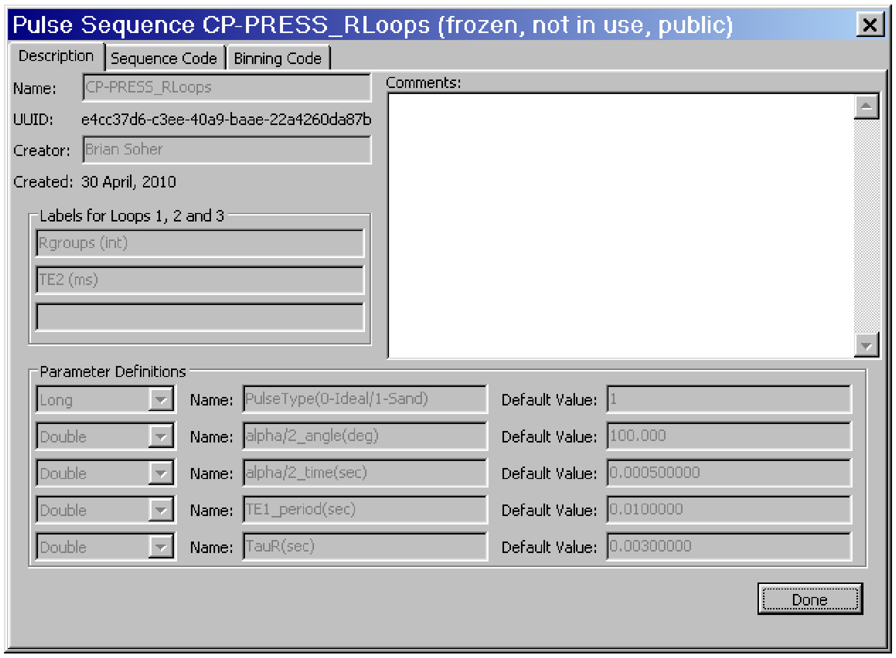


Name	Public	Use Count
CP-PRESS_RLoops	x	
Example_CP-PRESS1	x	
Example_SpinEcho	x	
JPRESS_Ideal	x	
One-Pulse	x	1
PRESS_Ideal	x	3
STEAM_Ideal	x	
Spin-Echo	x	1
exampleSeq1	x	
press_fullpulse_indiv	x	
press_fullpulse_summed	x	

The "Public" column indicates if a sequence has ever been exported (or imported from someone else). Pulse Sequences with the Public column marked ‘x’ can not be edited except in the Name and Comment fields. The "Use Count" column indicates how many local

Experiments use this sequence. While in use by any Experiments, the sequence can not be deleted.

View: Select a sequence from the main list. If more than one is selected the first on in the list is viewed. This button pops up a secondary dialog with three tabs that contain the sequence creation information, widget descriptors and pulse sequence and binning code. These tabs are not editable. See figure below for example of View.


Pulse Sequence CP-PRESS_RLoops (frozen, not in use, public)

Description | Sequence Code | Binning Code

Name: CP-PRESS_RLoops

UUID: e4cc37d6-c3ee-40a9-baae-22a4260da87b

Creator: Brian Soher

Created: 30 April, 2010

Comments:

Labels for Loops 1, 2 and 3

Rgroups (int)

TE2 (ms)

Parameter Definitions

Long	Name: PulseType(0-Ideal/1-Sand)	Default Value: 1
Double	Name: alpha/2_angle(deg)	Default Value: 100.000
Double	Name: alpha/2_time(sec)	Default Value: 0.000500000
Double	Name: TE1_period(sec)	Default Value: 0.0100000
Double	Name: TauR(sec)	Default Value: 0.00300000

Clone: This option allows a user to make a copy of an existing pulse sequence. This is most useful when an existing sequence is “public” or otherwise not editable because it is referenced by an existing Experiment. 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 are not referenced by an experiment may be deleted. 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(s) in the list can be deleted from the database.

Import: Pops up a secondary dialog that allows the user to select an XML file that contains one or more Vespa Simulation pulse sequences. Any pulse sequences in the file are added to the database, provided that they aren't in the database already. Pulse sequences with UUIDs that match those already in the database are simply ignored. Please be sure to import/export pulse sequences with the “Manage Pulse Sequence” utility to ensure proper operation.

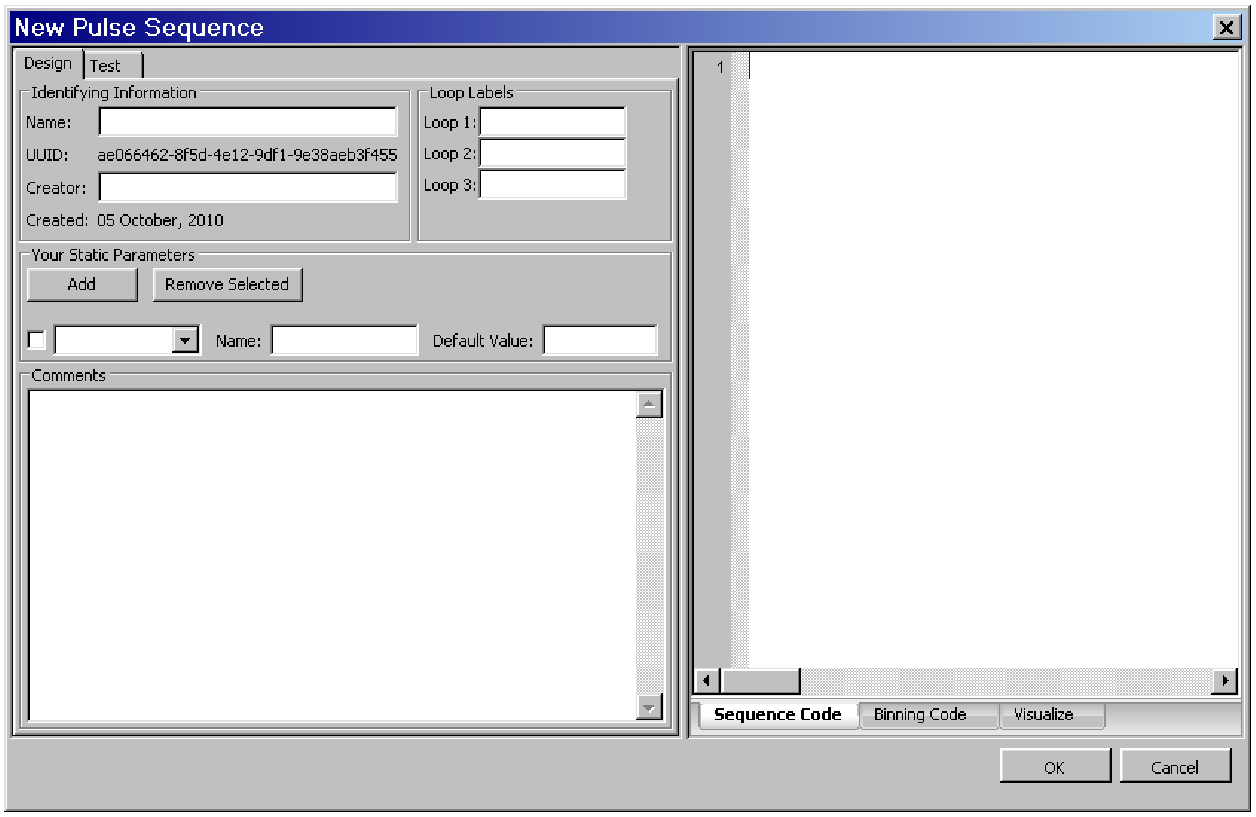
Export: Select a Pulse Sequence from the list. A second dialog pops up that allows the user to browse for the output filename, select if output should be compressed and allows an additional export comment to be typed in. Note that the action of exporting an object causes it to be marked as “frozen” in the database and “public” in the pulse sequence management dialog. 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. Please be sure to import/export pulse sequences with the “Manage Pulse Sequence” utility to ensure proper operation.

New: A “Pulse Sequence Editor” dialog pops up that allows the user to design and test a pulse sequence using PyGAMMA code. The user must provide general descriptive information about the sequence. They must describe how to lay out the pulse sequence in the Experiment tab → Simulate sub-tab, both for the standard loop variables as well as any user-defined parameters. The user must also provide PyGAMMA code (i.e. a Python script that uses calls to the PyGAMMA library) for the main pulse sequence. Default code for binning results is provided. You can keep this code, alter it, replace it or delete it entirely. (See Appendix A for details.).

The New Pulse Sequence Editor widget is shown below. Please note that there are 2 main windows: 1) the Design/Test notebook (left) and 2) the Code/Display notebook (right). To create a pulse sequence, fill in the “Design” tab, the Sequence Code tab and Binning Code tab. At this point, if you have filled them in correctly, you have created a pulse sequence and if desired, could quit the dialog. Alternatively, you can hit the Update Testing Control button and proceed to test and modify your pulse sequence as desired.

The “Test” tab and “Visualize” tab allow you to test your pulse sequence before running it in an Experiment. Effectively, it allows you to run a mini-Experiment where only one metabolite and one value, for any loops you defined, are allowed. More information on these is provided below.

When you hit the OK button (lower right), the pulse sequence is saved to the database, the New Pulse Sequence dialog goes away, and you should see your new sequence listed in the main Manage Pulse Sequence dialog list. If you do not wish to save your pulse sequence, hit Cancel.



Design Tab – Data input fields

Name: This is how the pulse-sequence is displayed in the dropdown list in the Experiment tab → Simulate sub-tab .

Creator: The name of the person creating the pulse sequence

Loop Labels: 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 sub-tab that describe these loops. An example would be “TE [ms]” for a spin echo experiment. N.B. If you indicate that a user should provide a timing in [ms], don’t forget to divide by 1000 in your program to get a timing value in [sec] that PyGAMMA requires. The examples demonstrate how to define and use these parameters in PyGAMMA code.

Your Static Parameter Definitions: Each pulse sequence GUI has a section where users can set values for additional static parameters that are passed into the simulation. The GUI for these parameters needs to be described in the Pulse Sequence Editor so that the main program can display them properly. By hitting the Add button, a row of widgets will appear that contain three fields used to describe the GUI for one static parameter: A data type (selected from a drop-list), a "Name" string, and a "Default Value" string. The Name string will be used by the Experiment Tab → Simulate sub-tab as a label to describe this field when the pulse sequence is selected for an Experiment. The data type shows up as a label in the far right hand side as a reminder. The default value is inserted as the initial value that is displayed in that field. The Remove Selected button can be used to remove unwanted static parameters while designing a pulse sequence. Select the check box on the left side of each row of parameters you want to remove, and then hit the Remove Selected button.

As described in more detail Appendix A, the values of these user-specified parameters are passed to each Simulation that is run as part of an Experiment. The results of setting up your pulse sequence loops and additional parameters can be viewed in the “Test” tab. The examples demonstrate how to define and use these parameters in PyGAMMA code.

Note: By selecting a data type for a user-specified parameter in the drop down menu, the user will be reminded to enter a variable of that type, but the actual field value will be passed as a string that must be appropriately converted before being used in PyGAMMA simulation code. Please select your default types and values accordingly.

Comments: A field where you can enter a lot of text to remind yourself why you make this pulse sequence when you check back on it 3 months from now. This is also a good place to put information for users on how to use this sequence.

Sequence Code Notebook Tab

Note: This tab can be moved and positioned in a variety of ways. Left click and drag the tab of the pane that you want to re-locate to the position that you want it.

The Sequence Code tab is a text window in which PyGamma code can be pasted and/or edited. See Appendix A for details of how Simulation interacts with your PyGAMMA code. There's an example in the figure below.

Binning Code Tab

Note: This tab can be moved and positioned in a variety of ways. Left click and drag the tab of the pane that you want to re-locate to the position that you want it.

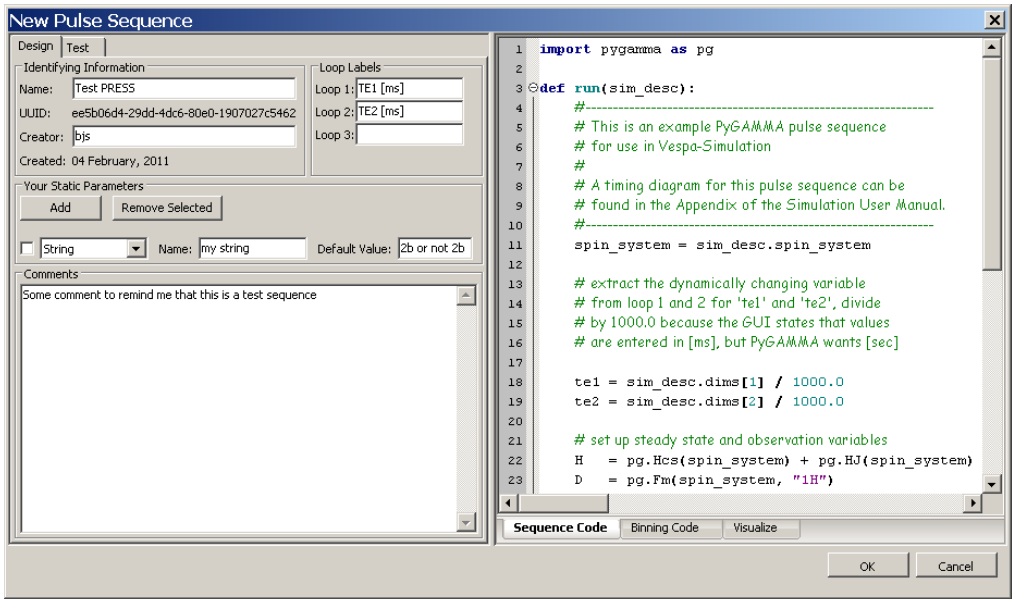
This is a text window (like the Sequence Code tab) in which PyGamma code can be pasted and/or edited. Simulation adds default binning code when the New Pulse Sequence dialog opens, but you can edit or delete it as you like. Again, details are in Appendix A.

Test Tab

When the user clicks on the Test tab, the settings in the Design tab are validated, and if passed, then the Test tab widgets are updated to reflect the pulse sequence design. If there are any missing fields or other errors in the Design tab, the user is prompted to fix these prior to switching to the Test tab.

Note: A similar validation takes place when the user hits the OK button. Only a validated pulse sequence can be saved into the database. However, the validation only checks to see if all necessary data is available in a reasonable format, NOT if it is functional PyGAMMA code.

An example is shown below of how settings in the Design tab are represented on the Test tab. Note that the test values for each loop have been entered and that the default value for the “my string” user parameter has been altered as well.



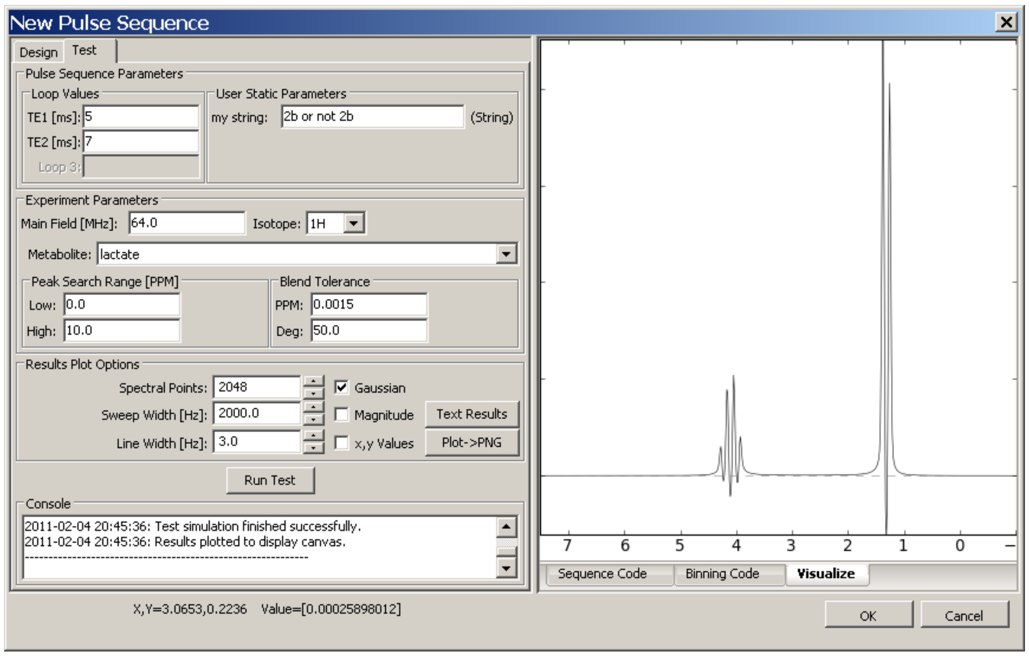
Identifying Information
Name: Test PRESS
UUID: ee5b06d4-29dd-4dc6-80e0-1907027c5462
Creator: bjs
Created: 04 February, 2011

Loop Labels
Loop 1: TE1 [ms]
Loop 2: TE2 [ms]
Loop 3:

Your Static Parameters
String Name: my string Default Value: 2b or not 2b

Comments
Some comment to remind me that this is a test sequence

```
import pygamma as pg

def run(sim_desc):
    #----------------------------------------------------------------
    # This is an example PyGAMMA pulse sequence
    # for use in Vespa-Simulation
    #
    # A timing diagram for this pulse sequence can be
    # found in the Appendix of the Simulation User Manual.
    #----------------------------------------------------------------
    spin_system = sim_desc.spin_system

    # extract the dynamically changing variable
    # from loop 1 and 2 for 'te1' and 'te2', divide
    # by 1000.0 because the GUI states that values
    # are entered in [ms], but PyGAMMA wants [sec]

    te1 = sim_desc.dims[1] / 1000.0
    te2 = sim_desc.dims[2] / 1000.0

    # set up steady state and observation variables
    H   = pg.Hcs(spin_system) + pg.HJ(spin_system)
    D   = pg.Fm(spin_system, "1H")
```



TE1 [ms]:	5
TE2 [ms]:	7
Loop 3:	
my string:	2b or not 2b (String)
Main Field [MHz]:	64.0
Isotope:	1H
Metabolite:	lactate
Low:	0.0
High:	10.0
PPM:	0.0015
Deg:	50.0
Spectral Points:	2048
Sweep Width [Hz]:	2000.0
Line Width [Hz]:	3.0

[✓ Gaussian] Magnitude x,y Values

```
2011-02-04 20:45:36: Test simulation finished successfully.
2011-02-04 20:45:36: Results plotted to display canvas.
------------------------------------------------------------
```

X,Y=3.0653,0.2236   Value=[0.00025898012]

Loop Values: These loops were defined in the Design tab. Any loops without a label are not included in the pulse sequence. The user must fill in a value for use in the test run for each loop.

User Static Parameters: User parameters were defined in the Design tab. They are initially populated with their default values, but may be altered for the test run as necessary.

Experiment Parameters: When the Run Test button is hit, a mini-Experiment will be run to test the user's pulse sequence. In order to properly run and display results the experiment needs values for Main Field [MHz], the isotope, one metabolite to be run (select from the list as sorted by isotope), and the binning parameters for Peak Search Range and Blend Tolerance (see Appendix A for more information on the standard blending algorithm)

Results Plot Options: These values only affect how the metabolite result is plotted to the Display Canvas tab in the notebook. Spectral Points are the number of points in the metabolite FID, Sweep Width defines the FID dwell time, Line Width [Hz] defines the broadening applied to the FID. Checking the Gaussian box applies a Gaussian lineshape, when it is not checked a Lorentzian lineshape is applied. Checking Magnitude plots magnitude data on the canvas, otherwise real data is plotted. Checking x,y Values will show in the lower left corner of the plot the x and y axis values of the location of the mouse as it moves across the canvas.

Text Results Button: Creates a text representation of the metabolite test results and displays them in a native text editor on your computer

Plot->PNG Button: Creates a PNG format image of the plot display and shows it in a native image viewer on your computer.

Run Test Button: Runs a test Experiment on the pulse sequence. The Start and End times should be reported in the Console window. Any additional exceptions that are raised should be reported between these messages.

Console: The place where text messages about each Test Run are printed.

Visualize Notebook Tab

Note: This tab can be moved and positioned in a variety of ways. Left click and drag the tab of the pane that you want to re-locate to the position that you want it.

The test metabolite results are reconstituted as a frequency domain spectrum as described in the Results Plot Options and plotted to this display tab. The Left mouse button can be used to draw a zoom box in both x and y directions. Multiple zooms can be performed. Left clicking once in place will zoom you all the way out to the maximum x-axis extent and fit the y-axis to approximately the min/max data range. Clicking and dragging on the Right mouse will draw a Span Cursor, two vertical cursors on the screen, filled in with light gray. These will stay in place between test runs as you vary loop and parameter values. Right clicking in place will turn off the Span Zoom region.

Edit: The first highlighted sequence is opened in a form similar to the New Sequence dialog. Note: Only the metabolite Name, and Comment are editable if the pulse sequence is “public” or referred to by one or more Experiments. 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.

If the sequence is editable, the existing values of the pulse sequence object are populated into the Design and Test tabs on startup. The name of the pulse sequence from the main dialog is shown in the dialog title. The pulse sequence setting can be edited and tested just like a New pulse sequence would be. Hitting OK saves any changes into the database. Cancel quits the dialog without saving changes.

6. Results Output

6.1 Results output into standard text editor

The Vespa-Simulation **View** menu lists commands that only apply to the active Experiment Tab. Select the **View→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. N.B. 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. The Vespa-Simulation **View** menu lists commands that only apply to the active Experiment Tab. Select the **View→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 saved 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 readable into PowerPoint.

The Vespa-Simulation **View** menu lists commands that only apply to the active Experiment Tab. Select the **View→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 is under the hood?

A.1.1 Vespa-Simulation Basic Concepts

This is a combination of logical concepts and constraints 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 spectral line has 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.

A.1.2 Experiments

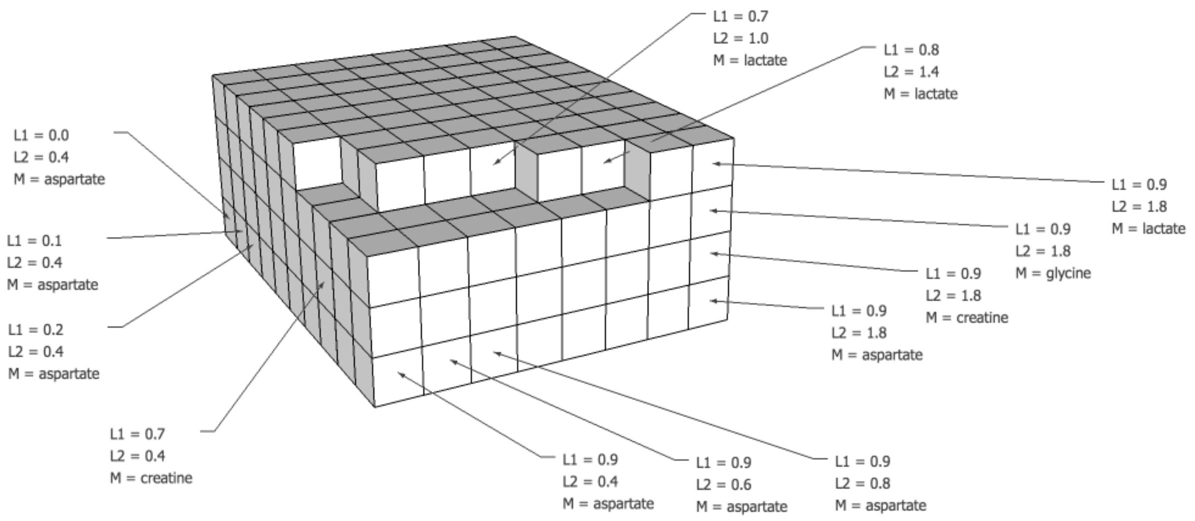
Experiments are the main focus of the Simulation application. An Experiment's *raison d'etre* is to run a set of simulations. This set of simulations is the experiment's *results space*.

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 metabolites 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.



Simulations themselves know nothing about one another and are agnostic to the order in which they're run. The existing Vespa-Simulation code is geared towards generating a regular results space that we iterate over in a very straightforward order. (More complex result spaces and iteration orders could be created provided you can dream up a GUI that allows users to describe that 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 discuss how *user defined* static variables (ie. they do not change as the loop variables are incremented) can also be passed into spectral simulations.

A.2 First Steps for Creating Your Own Pulse Sequences

A.2.1 Overview

This section contains a lot of information about how the PyGAMMA pulse sequences that you design in the Pulse Sequence Designer dialog work within the Vespa-Simulation application. There is a lot of information here, but the thing to keep in mind is that there are 5 very well documented examples following this section. Please take the time to read the "rules of the road" here. It should keep you from any rookie mistakes like not using the right name for the function that your PyGAMMA code goes inside. And then dig into some "learn by doing" afterwards.

The interface between Simulation and pulse sequence code changed in version 0.1.2 of Vespa. The new interface is not compatible with the old one. Pulse sequence code in prior versions won't run under the new interface without some changes.

We're written a practical guide to upgrading to 0.1.2. This document explains the details behind the change, such as --

- How Simulation Runs a Pulse Sequence
- Why We Changed the Interface
- What the New Interface Looks Like

A.2.2 How Simulation Runs Your Pulse Sequence (A Brief Review)

Each pulse sequence consists of two pieces of code -- the sequence code and the binning code. The sequence code is generally where we put PyGAMMA code that describes the simulation and generates the results. The binning code can subsequently be used to simplify these results (e.g. the combination of degenerate lines - hence, the name 'binning'). The binning step is optional.

A.2.3 The Interface Between Simulation and Your Pulse Sequences

Simulation imports your code as modules. Importing a module should be familiar to anyone who has used Python, and that's how Simulation uses your pulse sequence code. The sequence and binning code segments you provide are saved to temp files and then Simulation imports those files as two individual modules: one module for the sequence code and another module for the binning code.

This means that your sequence code is in its own namespace and your binning code is in its own separate namespace. It's as if they were in modules named my_sequence_code.py and my_binning_code.py.

Simulation calls the **run()** function in your code. Calling a function in an imported module should also be familiar to anyone who has used Python. In this case, you provide a function called run() in both your sequence and binning code. Those functions each accept a single parameter as described below.

Simulation passes a class instance to your code instead of a dictionary. Simulation passes an instance of a class that describes the simulation with a well-defined set of attributes.

The class contains attributes like field, peak_search_ppm_low, dims, etc. It also contains an attribute called spin_system that returns a spin system for the current simulation.

For a full list of the class attributes, examine the class definition (in vespa/simulation/src/simulation_description.py) or see section A.3.1 below.

The same object is passed to both the sequence and binning code, so it's easy to "pass" a variable created in the sequence code to the binning code. Just assign it to an attribute on the object. For instance, to make the transition table matrix available to the binning code, add this to your sequence code:

```
sim_desc.mx = PyGAMMA.TTable1D(ACQ.table(sigma0))
```

This demonstrates a larger point: once the simulation description object is passed to your code, Simulation doesn't use it. Your code is free to manipulate it as you see fit. Not only can you add attributes and methods, you can delete and overwrite them too.

Simulation passes 8 bit strings. All strings passed to your code in the simulation description are UTF-8 encoded 8 bit strings. If you don't know what this means, you can probably just ignore it. Specifically, it means that the strings are safe for PyGAMMA. (see http://scion.duhs.duke.edu/vespa/gamma/wiki/PyGammaAndPythonStrings)

Your code returns results via a **return** statement. Your code (sequence or binning, as explained below) should return a 3-tuple of lists (or other iterables) of floats that represent the ppm, area, and phase values. The phrase *"...(or other iterables)..."* means that the elements of the 3-tuple can be lists, tuples, PyGAMMA.DoubleVector objects, numpy arrays, etc. They don't even have to be of the same type. For instance, this is a valid set of results:

```
return ( [0, 0, 0], numpy.zeros(3), PyGAMMA.DoubleVector(3) )
```

The tuple elements must be the same length. If they're not, Simulation discards your results and raises a ValueError.

You can return results from the sequence or binning code. Since not everyone will want to run a binning step, we've made it easy to skip. If your sequence code returns a 3-tuple of results as described above, Simulation won't call your binning code. If your sequence code returns None(or doesn't have a return statement at all), then Simulation will call your binning code which must return the 3-tuple of results.

Results must contain only Python **float**, **int** or **long** objects. The type of every element in the ppm, area and phase lists must be float,int or long. One can't return, for example, Python complex numbers, PyGAMMA complex numbers, or ctypes.c_float objects.

If this rule is violated, Simulation discards your results and raises a ValueError.

A.3 Creating a Pulse Sequence without Extra Parameters

A.3.1 How to create a "One-Pulse" pulse sequence

An important thing to remember in pulse sequence design is that regardless of how many looping variables are defined, each spectral simulation (calculation) receives a standard set of pulse sequence parameters as described below.

To achieve this, an object called "sim_desc" (the simulation description) is created to store these common (and any other) parameters. A new sim_desc object is created for each Simulation within an Experiment object (ie. You can not use this object to "pass messages" between simulations). Each sim_desc object is sent to a function that executes the PyGAMMA spectral simulation that it describes. On completion of each simulation, your code returns lists of results (area, ppm, and phase values). Simulation adds start/finish time stamps and stores the results in the database.

The 14 standard parameters and one user defined parameter are stored as attributes of the sim_desc object, and are:

'vespa_version' – (string) version number of the Vespa-Simulation program in string format

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

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

'peak_search_ppm_high' – (float) upper end of 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 (specified in 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)

and

'user_static_parameters' – (list) static parameters defined by the user in the GUI that are stored in this list as strings in the order that they are presented in the GUI (see below). Note: In this One-Pulse experiment there are no user defined parameters so the list would be empty.

Via the attribute "spin_system", the sim_desc object provides a PyGamma spin_system object constructed from the field, isotopes, chemical shifts and j-coupling values. This is only for your convenience and you're welcome to use the original values any way you please.

The One-Pulse Example

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

```
import PyGAMMA as pg

def run(sim_desc):
    #-----------------------------------------------------------------
    # This is an example PyGAMMA pulse sequence
    # for use in Vespa-Simulation
    #
    # A timing diagram for this pulse sequence can be
    # found in the Appendix of the Simulation User Manual.
    #----------------------------------------------------------------
    spin_system = sim_desc.spin_system

    # set up steady state and observation variables
    H   = pg.Hcs(spin_system) + pg.HJ(spin_system)
    D   = pg.Fm(spin_system, "1H")
    ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
    ACQ = ac

    # excite and acquire the data
    sigma  = pg.sigma_eq(spin_system)
    sigma0 = pg.Iypuls(spin_system, sigma, "1H", 90.0)

    # instantiate and save transition table of simulation results
    # note. this step copies the TTable1D result from the ACQ into
    #       a TTable1D object in the sim_desc object. Thus, when
    #       we return from this function and the ACQ variable gets
    #       garbage collected, our copy of the results in not affected
    sim_desc.mx = pg.TTable1D(ACQ.table(sigma0))
```

The first thing to note is that other than the “spin_system” attribute, this pulse sequence does not make use of any of the parameters in the sim_desc object. 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).

In this example the first line of code (ignoring comments) defines the Hamiltonian, in this case consisting simply of chemical shift and J coupling terms. The second through fourth lines define the detection and acquisition operators. The fifth line defines an equilibrium density matrix. The sixth line applies an ideal 90 degree pulse to the density matrix and returns the resulting density matrix. The final line applies the acquisition operator to the final density matrix and returns a transition table. For more details on PyGAMMA and GAMMA objects consult the PyGAMMA documentation.

Note. The final line of code demonstrates the one “output” code requirement if the user plans on using the standard ‘binning_code’ provided by Simulation as the default. In that case, the user must create and fill a transition table attribute called “mx” in the sim_desc object.

Note. In the final line, we have to explicitly create a new TTable1D object and copy the simulation results from the TTable1D in the ACQ variable. This is done by default if the TTable1D to be copied is passed into the initialization of the object. We copy this information because otherwise we would only have a reference to the ACQ object’s results. When we return from the function, the ACQ object is ‘garbage collected’ and then our reference is broken.

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:

```
import PyGAMMA

def run(sim_desc):
    area   = PyGAMMA.DoubleVector(0)
    ppm    = PyGAMMA.DoubleVector(0)
    phase  = PyGAMMA.DoubleVector(0)
    field  = sim_desc.field
    nspins = sim_desc.nspins
    tolppm = sim_desc.blend_tolerance_ppm
    tolpha = sim_desc.blend_tolerance_phase
    ppmlow = sim_desc.peak_search_ppm_low
    ppmhi  = sim_desc.peak_search_ppm_high

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

    return (ppm, area, phase)
```

This code expects that an attribute named “mx”, that is a PyGAMMA transition table, already exists in the sim_desc object. 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 of the binning 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, your code must **always** return the three equal length lists representing ppm, area and phase.

A.3.2 A “One-Pulse” pulse sequence that does NOT use binning code

Here is the PyGAMMA code that is in the sequence_code string for the One-Pulse No Binning sequence

```
import math
import PyGAMMA as pg

def run(sim_desc):
    #-------------------------------------------------------------------------
    # This is an example PyGAMMA pulse sequence for use in Vespa-Simulation.
    # It demonstrates how results can be returned directly by the sequence
    # code as opposed to being returned by the binning code. When the
    # sequence code returns results, the binning code is never invoked.
    #
    # A timing diagram for this pulse sequence can be found in the Appendix
    # of the Simulation User Manual.
    #-------------------------------------------------------------------------
    spin_system = sim_desc.spin_system

    # set up steady state and observation variables
    H   = pg.Hcs(spin_system) + pg.HJ(spin_system)
    D   = pg.Fm(spin_system, "1H")
    ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
    ACQ = ac

    # excite and acquire the data
    sigma  = pg.sigma_eq(spin_system)
    sigma0 = pg.Iypuls(spin_system, sigma, "1H", 90.0)

    # instantiate transition table of simulation results
    mx = pg.TTable1D(ACQ.table(sigma0))
    # Calculate results
    mx_indices = mx.Sort(0, -1, 0)
    normal = 0.5 * 2**(sim_desc.nspins - 1)

    ppms = []
    areas = []
    phases = []

    for index in mx_indices:
        # ctmp is a complex number
        ctmp = mx.I(index)

        ppms.append(-mx.Fr(index) / (2.0 * math.pi * sim_desc.field))
        areas.append(math.hypot(ctmp.Rec(), ctmp.Imc()) / normal)

        if ctmp.Imc() or ctmp.Rec():
            phase = (-180.0 / math.pi) * math.atan2(ctmp.Imc(), ctmp.Rec())
        else:
            phase = 0

        phases.append(phase)

    phases = [(phase * -1.0) for phase in phases]

    return (ppms, ares, phases)
```

Note that the lines in yellow further process the original One-Pulse sequence in order to extract the transition lines from the PyGAMMA simulation and then process them so that they are appropriately passed back to the Simulation program. Note also that some of this code is PyGAMMA (mx.Sort, etc.), some is straight Python (math.pi, math.atan2, etc.).

The final line of code creates a tuple with three iterable objects (lists in this case, but it could also be tuples or other iterable objects) that contain the ppm values, areas and phase values for all lines. These lists MUST have the same length. These are the results values that are saved to the database.

The fact that your sequence code returns something other than None tells Simulation not to call the binning code.

A.3.3 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:

```
import PyGAMMA as pg

def run(sim_desc):
    #-----------------------------------------------------------------
    # This is an example PyGAMMA pulse sequence
    # for use in Vespa-Simulation
    #
    # A timing diagram for this pulse sequence can be
    # found in the Appendix of the Simulation User Manual.
    #-----------------------------------------------------------------
    spin_system = sim_desc.spin_system
    # extract the dynamically changing variable
    # from loop 1 and 2 for 'te1' and 'te2', divide
    # by 1000.0 because the GUI states that values
    # are entered in [ms], but PyGAMMA wants [sec]

    te1 = sim_desc.dims[1] / 1000.0
    te2 = sim_desc.dims[2] / 1000.0

    # set up steady state and observation variables
    H   = pg.Hcs(spin_system) + pg.HJ(spin_system)
    D   = pg.Fm(spin_system, "1H")
    ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
    ACQ = ac
    sigma0 = pg.sigma_eq(spin_system)

    # excite, propagate, refocus and acquire the data
    sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 90.0)
    Udelay = pg.prop(H, te1*0.5)
    sigma0 = pg.evolve(sigma1, Udelay)
    sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 180.0)
    Udelay = pg.prop(H, (te1+te2)*0.5)
    sigma0 = pg.evolve(sigma1, Udelay)
    sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 180.0)
    Udelay = pg.prop(H, te2*0.5)
    sigma0 = pg.evolve(sigma1, Udelay)

    # instantiate and save transition table of simulation results
    # note. this step copies the TTable1D result from the ACQ into
    #       a TTable1D object in the sim_desc object. Thus, when
    #       we return from this function and the ACQ variable gets
    #       garbage collected, our copy of the results in not affected
    sim_desc.mx = pg.TTable1D(ACQ.table(sigma0))
```

The first thing to note is that this pulse sequence utilizes the "spin_system" variable and also the sim_desc object for the Loop1 and Loop2 values in the `"te1 = sim_desc.dims[1]"` and `"te2 = sim_desc.dims[2]"` lines. There are no user-defined static parameters. Similarly to the example above a transition table attribute 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.4 Creating a Pulse Sequence with Extra Parameters

A.4.1 The "PRESS-CP with Variable R-groups" Pulse Sequence

Here is the PyGAMMA code that is in the sequence_code string for the PRESS-CP with Variable R-groups" sequence:

```
import PyGAMMA as pg

def run(sim_desc):
```

```
spin_system = sim_desc.spin_system

# extract the dynamically changing variable from loops 1, 2 and 3, divide
# 'te1' and 'te2' by 1000.0 because the GUI states that values are entered
# in [ms], but PyGAMMA wants [sec]

te1     = sim_desc.dims[1] / 1000.0
te2     = sim_desc.dims[2] / 1000.0
rgroups = int(sim_desc.dims[3])

# extract user static parameter values from the control dictionary. They
# are inserted into a list in the order that they are shown in the GUI.

pulse_type = int(  sim_desc.user_static_parameters[0])
ang90      = float(sim_desc.user_static_parameters[1])
pd90       = float(sim_desc.user_static_parameters[2])
tauR       = float(sim_desc.user_static_parameters[3])

pd180   = pd90 * 2.0
ang180  = ang90 * 2.0
offhz   = 0.0

# set up steady state and observation variables
H   = pg.Hcs(spin_system) + pg.HJ(spin_system)
D   = pg.Fm(spin_system, "1H")
ac  = pg.acquire1D(pg.gen_op(D), H, 0.000001)
ACQ = ac

# apply excitation pulse and propagate to first 180 pulse
sigma0 = pg.sigma_eq(spin_system)
sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 90.0)

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

# apply first 180 pulse and propagate to CP train start
sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 180.0)

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

sigma1 = sigma0

# apply the Carr-Purcell refocussing pulse train
if pulse_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(spin_system,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(spin_system, sigma0, H, "1H", offhz, pd90,  ang90)
            sigma0 = pg.Sypuls(spin_system, sigma1, H, "1H", offhz, pd180, ang180)
            sigma1 = pg.Sxpuls(spin_system, sigma0, H, "1H", offhz, pd90,  ang90)
            Udelay = pg.prop(H, tauR)
            sigma0 = pg.evolve(sigma1,Udelay)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

            sigma1 = sigma0

# propagate to second 180 pulse
Udelay = pg.prop(H, te2*0.5)
sigma0 = pg.evolve(sigma1, Udelay)

# apply second 180 pulse and propagate to data acquisition
sigma1 = pg.Iypuls(spin_system, sigma0, "1H", 180.0)

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

```
    # instantiate and save the transition table of simulation results
    # note. this step copies the TTable1D result from the ACQ into
    #       a TTable1D object in the sim_desc object. Thus, when
    #       we return from this function and the ACQ variable gets
    #       garbage collected, our copy of the results in not affected
    sim_desc.mx = pg.TTable1D(ACQ.table(sigma0))
```

The pulse sequence makes use of the “spin_system” attribute. The first seven lines of code (ignoring comments) are good examples of how to access the sim_desc object attributes for all three loop parameters and some user-defined static parameters. Note that the object attribute name for user-defined parameters is ‘user_static_parameters’ and that they are ordered into a list in the order they are arranged in the GUI. Thus, the alpha/2 pulse duration is set by the line:

```
      ‘pd90 = float(sim_desc.user_static_parameters[2])’
```

since this variable was the third one listed in the GUI. 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.

A.5 Creating a Pulse Sequence with an RF Pulse WaveForm

A.5.1 A “PRESS” sequence that uses a ‘real’ RF pulse read in from a file

A typical application might be to use one or more user defined pulses in a pulse sequence. Though various ways of accessing pulses in the VeSPA database for use in pulse sequences is described elsewhere a simple method that PyGAMMA provides is to read the complex values for a given pulse from file. The following code, closely resembling the above PRESS sequence code but using real pulses for the 180 pulses, illustrates how to accomplish this. In particular a user_static_parameter is used to specify the name and path of the file containing the pulse values:

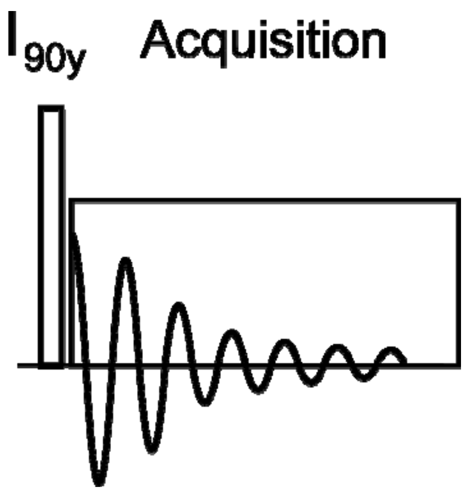
```
import PyGAMMA as pg

def run(sim_desc):
    #-----------------------------------------------------------------
    # This is an example PyGAMMA pulse sequence
    # for use in Vespa-Simulation
    #
    # A timing diagram for this pulse sequence can be
    # found in the Appendix of the Simulation User Manual.
    #-----------------------------------------------------------------
    spin_system = sim_desc.spin_system

    # extract the dynamically changing variable
    # from loop 1 and 2 for 'te1' and 'te2', divide
    # by 1000.0 because the GUI states that values
    # are entered in [ms], but PyGAMMA wants [sec]
    #
    # evolution after 90 before first 180 in msec and
    # divide by 1000 so PyGAMMA TE is in msec

    te1 = float(sim_desc.dims[1]) /1000.0
    te2 = float(sim_desc.dims[2]) /1000.0

    # extract user static parameter values from the control dictionary. They
    # are inserted into a list in the order that they are shown in the GUI.

    pulsestep     = float(sim_desc.user_static_parameters[0])
    pulse180file  = str(  sim_desc.user_static_parameters[1])

    # set up a container and read pulse values into it. You could also
    # read a file using Python code and subsequently inset values into
    # the PyGAMMA row_vector container. Then create a “time axis” array
    # with a time value for each point in the pulse vector

    pulse = pg.row_vector.read_pulse(pulse180file,
                                     pg.row_vector.ASCII_MT_DEG)
    ptime = pg.row_vector(pulse.size())
    total = pulse.size() * pulsestep

    for j in range(pulse.size()):
        ptime.put(pg.complex(pulsestep, 0), j)

    # create the pulse waveform and composite pulse objects from the
    # file and pulse sequence information

    pwf  = pg.PulWaveform(pulse, ptime, "TestPulse")
    pulc = pg.PulComposite(pwf, sys, "1H")

    # note that below we have to now account for the time of
    # the pulse in our propagation intervals in order to have
    # our TE calculate correctly.

    H = pg.Hcs(sys) + pg.HJ(sys)
    D = pg.Fm(sys)
    Udelay1 = pg.prop(H, 0.5*(te1-total))
    Udelay2 = pg.prop(H, 0.5*(te1-total + te2-total))
    Udelay3 = pg.prop(H, 0.5*(te2–total))
    ac = pg.acquire1D(pg.gen_op(D), H, 0.001)
    ACQ = ac
    sigma0 = pg.sigma_eq(sys)

    sigma1 = pg.Iypuls(sys, sigma0, 90.0)
    sigma0 = pg.evolve(sigma1, Udelay1)
    Ureal180  = pulc.GetUsum(-1)
    sigma1 = Ureal180.evolve(sigma0)
    sigma0 = pg.evolve(sigma1, Udelay2)
    sigma1 = Ureal180.evolve(sigma0)
    sigma0 = pg.evolve(sigma1, Udelay3)

    # instantiate and save the transition table of simulation results
    # note. this step copies the TTable1D result from the ACQ into
    #       a TTable1D object in the sim_desc object. Thus, when
    #       we return from this function and the ACQ variable gets
    #       garbage collected, our copy of the results in not affected
    sim_desc.mx = pg.TTable1D(ACQ.table(sigma0))
```

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

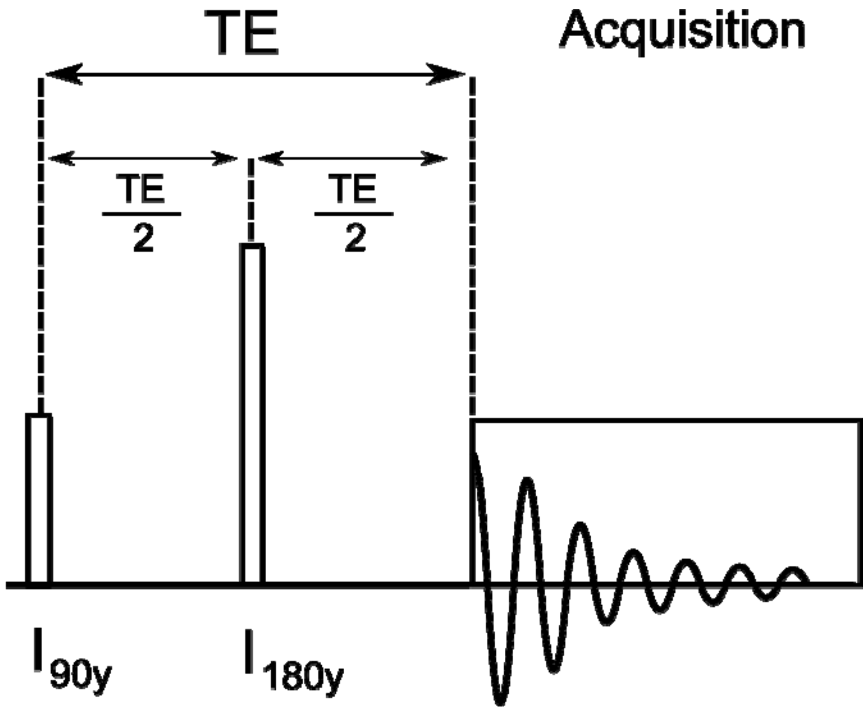
none

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

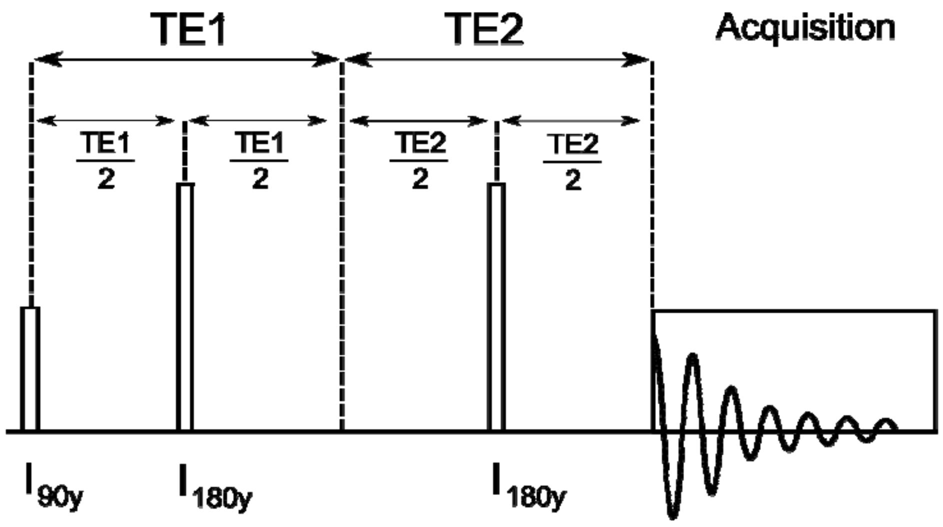
none

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

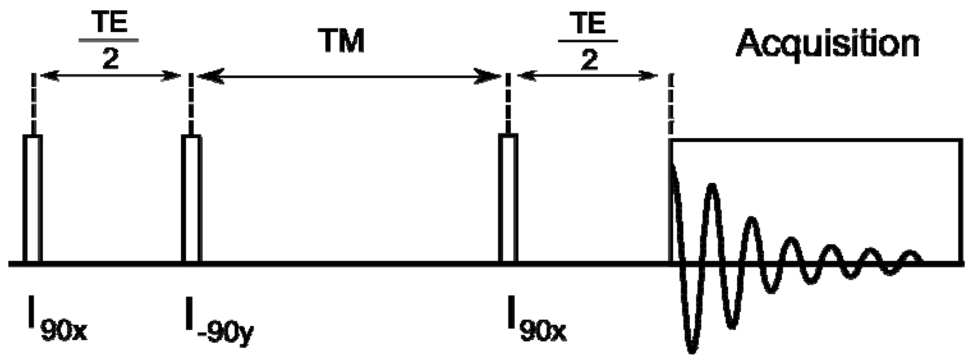
none

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

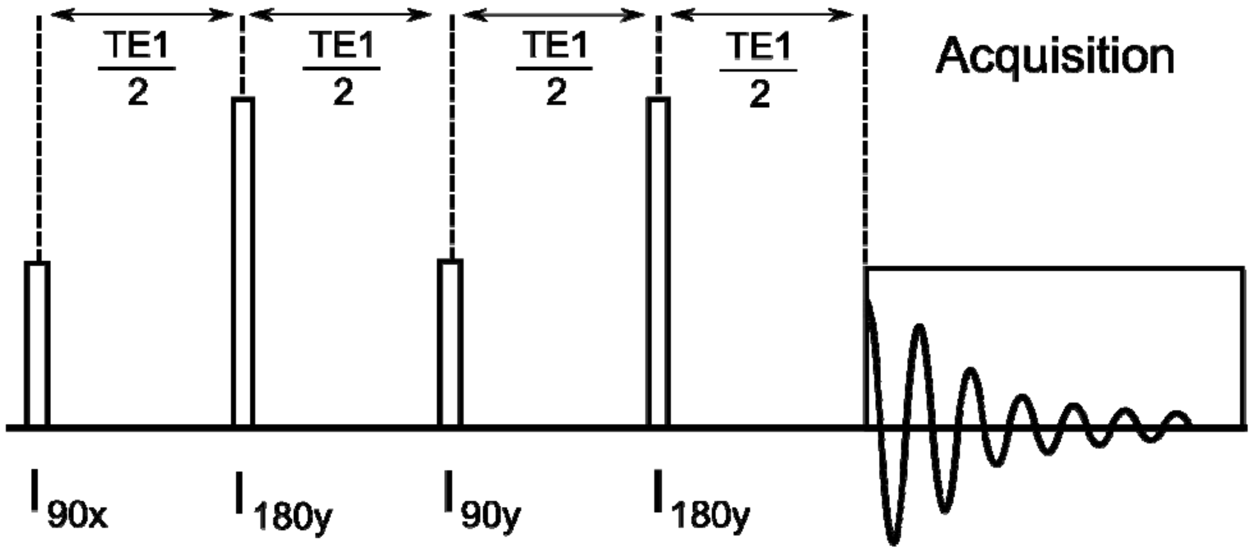
none

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.