DMMS.R User Manual

www.abbott.com

Version Identification:

Version	Date	Description	Author
1.3.0	12/11/2023	Applies to product version 1.3.0	Warren Muse, Karl Bowden, Robin Liu

Table of Figures

Table of Tables

1 Overview

The Diabetes Mellitus Metabolic Simulator (DMMS) is a computer application designed to simulate Type 1, Type 2, or prediabetes Diabetes Mellitus (DM) in the human body. The application is available in two versions:

- The research version (DMMS.R), which is used by researchers to simulate the effects of various devices and treatment options on a group of subjects.
- The educational version (DMMS.EDU), which is used by clinicians to demonstrate the effects of various treatment options on an individual subject.

Both versions of DMMS share a common simulation model. The design of this model is based on studies of glucose/insulin metabolism in humans[1]. The data derived from these studies were used to create a set of virtual subjects representing the full range of human DM variability as seen in Type 1, Type 2, and pre-diabetes people. Each subject is defined by a set of individual parameters that determine metabolic responses to various lifestyle factors and treatment options. Once developed, the model was validated in clinical trials using cohorts of actual patients.

DMMS.R allows for fast and inexpensive study of clinical outcomes. While good *in silico* performance cannot guarantee good *in vivo* performance, it can often rule out ineffective approaches. DMMS.R allows the researcher to test the stability and robustness of control without the safety constraints necessary in clinical trials.

DMMS.R is currently only available in Microsoft Windows OS.

2 About this Manual

The DMMS.R User Manual is intended to serve as a complete reference manual for the DMMS.R application. The reader is assumed to have a basic working knowledge of diabetes treatment and research.

While the terminology used should be familiar to you, a glossary is provided to help clarify terms (see Section 14).

The manual is laid out progressively; each section and subsection builds on material previously presented. However, each section is designed to be as independent of others as practical. References to other sections are provided as needed. In addition, an index is included to allow quick access to important topics.

Section 3 (How DMMS.R Works) explores the inner workings of the application. While this material is important, it is not necessarily essential to be able to use the program.

Sections 10, 11 and 12 present simple walkthrough examples of how to setup and run various simulations. While these examples are by no means comprehensive, they do offer a complete beginning-to-end survey

[1] Dalla Man, C, Rizza RA, Cobelli C, Meal Simulation Model of the Glucose-Insulin System. IEEE Transactions on Biomedical Engineering, 2007, 54(10):1740-1749.

of the use of the application. If you are anxious to get started with using DMMS.R, and wish to defer learning the details until later, you can go right to the walkthroughs.

2.1 Dialog Minimize, Maximize, Dismiss Controls

This manual contains many figures showing message boxes and dialog boxes.

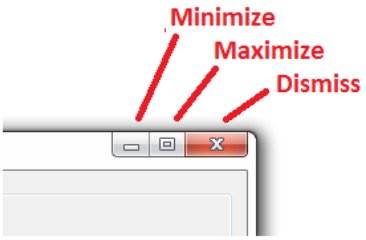


Figure 1: Dialog Sizing Controls: Windows Platform

3 How DMMS.R Works

DMMS.R has three basic parts:

- A database that contains metabolic characteristics of the populations of *in silico* subjects. This includes a set of parameters which are used to predict that subject's metabolic response to food, insulin, drugs, and exercise.
- A mathematical simulation model that applies a set of input data to a selected subject in the database. The output of the simulation model is a prediction of each subject's metabolic response state, including blood glucose, insulin levels, etc., over a given period of time.
- A user interface consisting of a set of computer screens (or dialogs) that enable you to interact with the database and simulation model[2].

Running a simulation involves the following steps:

- The user selects a set of *in silico* subjects from one of the five available subject groups, defines the length of the simulation and the time at which the simulation will start.
- The user defines a treatment protocol, including mealtime, duration and amounts, medication treatments with timing and dosing and exercise timing and duration.
- The simulation model takes the protocol definitions you have entered, extracts subject data from the database, and computes the metabolic response state for each of the subjects chosen for the study, over the length of the simulation.

[2] These dialogs contain several text boxes, radio buttons, drop-down lists, etc. that support viewing and entering specific data items. Although we often see these things described collectively as GUI "controls," that term may be ambiguous when describing a clinical simulation process. So, to avoid confusion, we will use the phrase "dialog controls" when referring to GUI controls. Labels on dialogs and dialog controls will be shown in italics when referenced in this manual.

- Output data from the model are displayed as a trend graph of blood glucose (BG), insulin-on-board (IOB), boluses applied, carbohydrate (CHO), meals consumed, and other information. Output data for each subject can also be saved as .csv files.

The user may repeat these steps as needed, using the same group of subjects, or different subjects.

Configuration (set up) for a simulation may be saved to a file for future use (see Section 8.1). The simulation may then be re-run, either through the DMMS.R user interface, or the command line interface. See Appendix B: Running Simulations from the Command Line Interface.

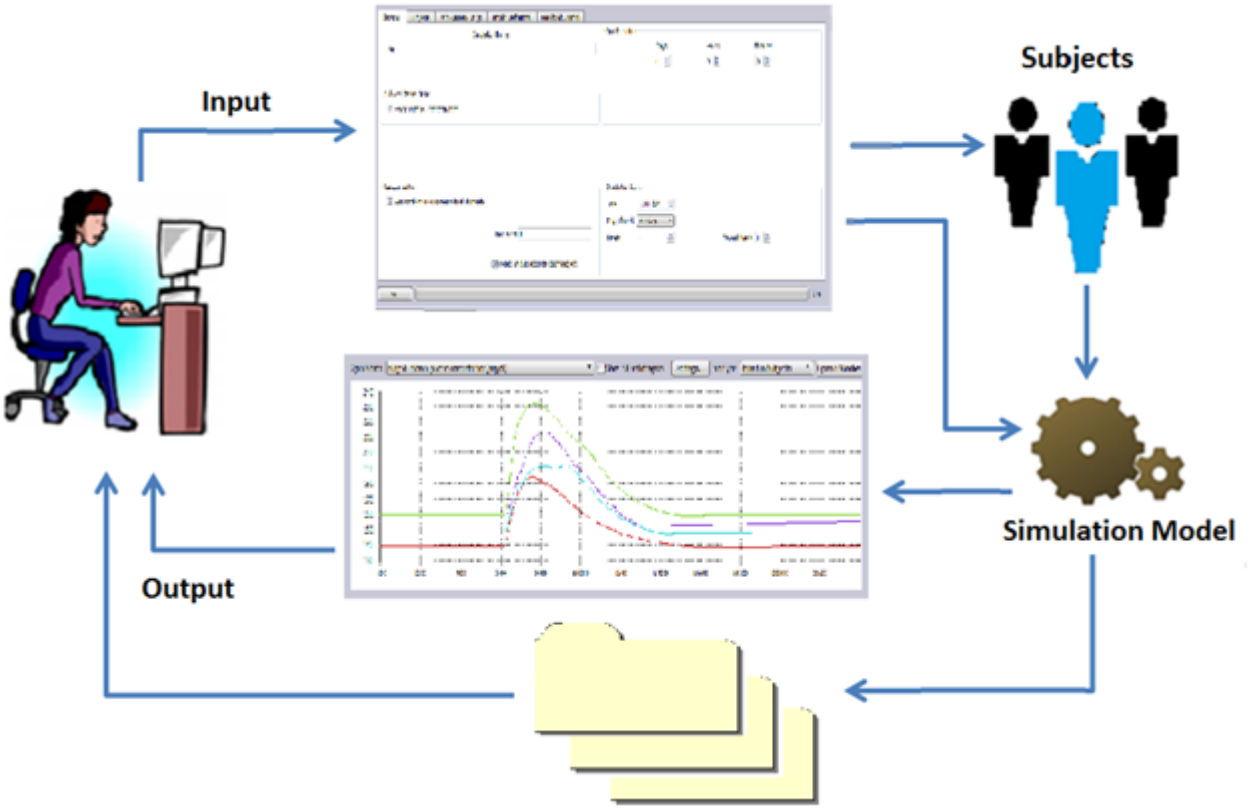


Figure 2: How DMMS.R Works

3.1 Subject Populations

DMMS.R supports simulations based on any one of 55 *in silico* (i.e., virtual) subjects. These subjects comprise 5 populations, categorized by disease-type and age:

Type	Age	Individual Subjects	"Average" Subject
Type 1	Adult (> 18 years old)	10	1
	Adolescent (14-18)	10	1
	Child (<14)	10	1
Type 2	Adult(>18)	10	1
Prediabetes	Adult (>18)	10	1

Totals		50	5

Table 1: Subject Populations

Each population has 10 subjects, each with unique metabolic responses. The three Type 1 populations have one additional subject representing the "average" response for the group.

Type 1 subjects are defined as being dependent on exogenous insulin.

Type 2 subjects have a fasting blood glucose level above 125 mg/dL or have a two-hour oral glucose intolerance above 200 mg/dL. These subjects show the presence of individualized endogenous insulin secretion.

Pre-diabetic subjects have a fasting blood glucose level of 100 – 125 mg/dL or have a two-hour oral glucose intolerance of 140-199 mg/dL.

3.2 The Model

DMMS.R is based on the compartmental model of the human metabolic system as developed by Dalla Man et al[1]. The model iteratively tracks the progress of hormones (e.g., insulin and glucagon) through the body and computes the *in-silico* subject's blood glucose level at each step. In addition to the *in-silico* subjects, the model includes:

- Sensors: These provide feedback from a human subject, by making measurements of the subject's metabolic state available in the same way that a real-world sensor (or an idealized sensor) would. For example, these may provide BG measurements by simulating a CGM or by providing an SMBG reading.
- Control Elements: These provide all the logic involved in closed loop control. They define the protocols for treating a subject and provide calculations for the amounts of treatments. These calculations are made based on signals from the sensor elements, or may be based on timing (e.g., meals, exercise). The control elements establish what input values (carbohydrates, injections of insulin or glucagon, etc.) are intended to be delivered to the subject (hereafter "intended subject inputs").
- Delivery Elements: These simulate devices that deliver the intended subject inputs described above. They may represent real-world devices, such as insulin pumps, or idealized devices. In some cases, they are not necessary – the "desired" input values prescribed by a control element may be unmodified by any delivery element before being provided to the subject (for example, this would typically be the case for a meal provider).

  Each delivery element will take the intended subject input and deliver the actual subject input.

Figure 3 shows these basic elements and the interactions among them. Additional detail is shown in Figure 4.

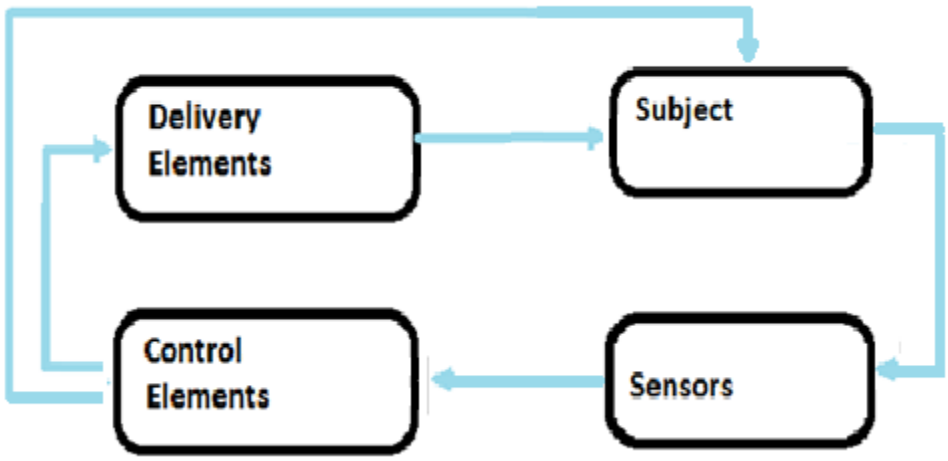


Figure 3: DMMS Model - a Bird's Eye View

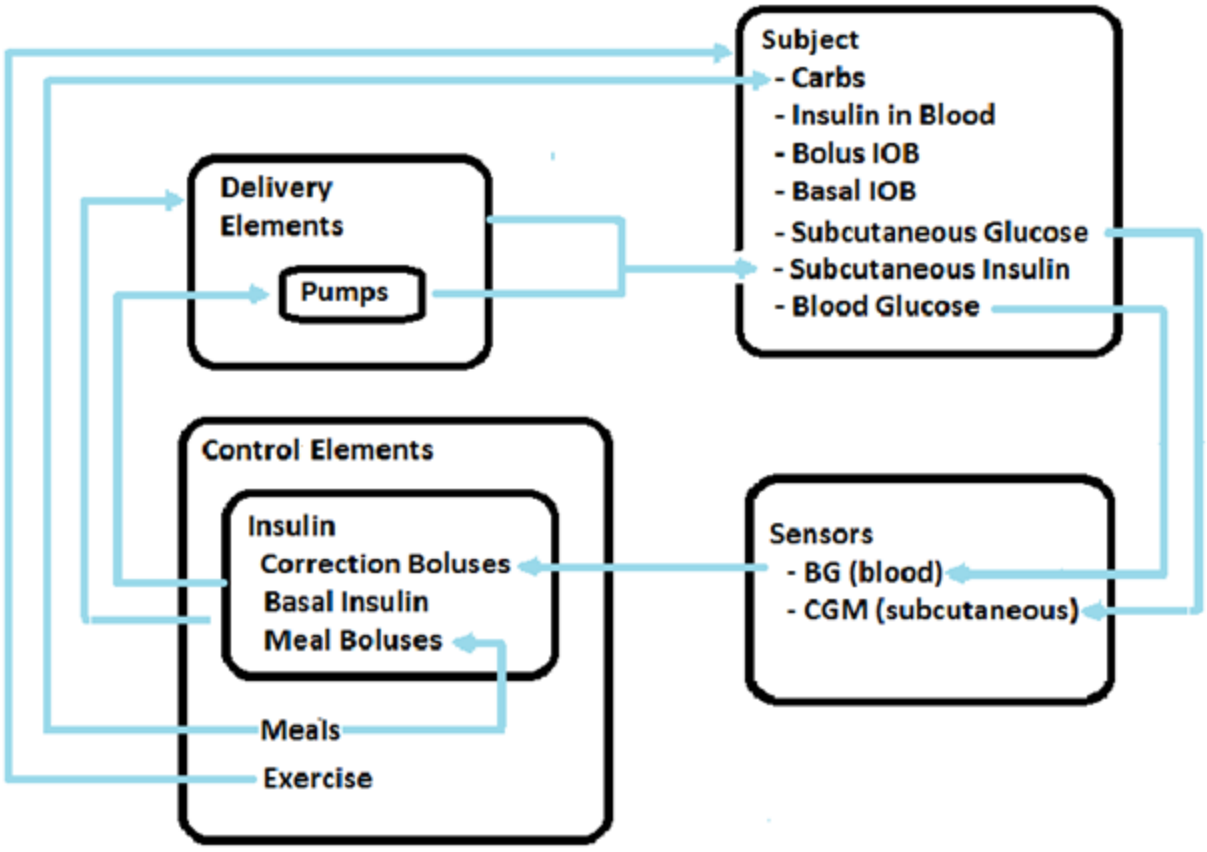


Figure 4: DMMS Model – a Closer Look

The DMMS.R user interface permits a great deal of flexibility. You may choose any subject, or group of subjects, and select and configure any number of sensors, control elements, and delivery elements. The

selection and configuration of control elements defines the treatment plan to be used in the simulation. See Section 6 for more detail:

- The Subjects: 6.4.2
- Sensors: 6.4.6.1
- Control Elements: 6.4.6.2
- Delivery Devices: 6.4.6.3

The simulation algorithm will activate each defined sensor, control element, and delivery device. Sensors and control elements will be activated in the order in which you listed them. To accommodate input and output signal dependencies, you may adjust the list order. (See Sections 6.4.6.1.3, 6.4.6.2.3 and Section 6.4.6.3.3)

Each sensor outputs a single signal with each iteration of the model. You may configure the name of this signal or accept the default name provided by DMMS.R. You can think of this configuration (the name of the signal, and the sensor that supplies it) as "defining" the signal. (See Section 6.4.6.1.5).

Many control elements do not use input signals, but for several others, you must choose one or more of the defined signals as input to the control. (See Section 6.4.6.2.4).

The ability to define the signal combined with the ability to assign that signal to one or more control elements gives you a great deal of control of how the DMMS.R model works.

3.3 Insulin Definition

Each complete insulin definition in DMMS is represented as an "insulin-release pair". An insulin-release pair is made up of:

- An insulin analog, defined entirely by the post-release Pharmacokinetic/Pharmacodynamic (PK/PD) characteristics of the insulin after it is "released" into either subcutaneous space or the bloodstream.
- A release description that specifies the timing and degradation associated with the insulin's movement from injection to the bloodstream. This includes:
    - A release profile that defines the timing of how the insulin analog is released into subcutaneous space or the bloodstream. Such a profile would not be included in the definition of a rapid acting insulin. For these insulins, when injected subcutaneously, the PK characteristics that determine how the insulin progresses to the bloodstream are completely defined by the compartmental model of the subcutaneous spaces. If injected directly to the bloodstream, such insulins are immediately available in there. In both types of injection, no release profile is needed to reflect additional delays.
    - A bioavailability percentage that determines how much of the injected insulin reaches the plasma when injected subcutaneously.

By separating these two concepts, release specification and PK/PD characteristics, DMMS provides a unique and powerful tool for specifying characteristics of an insulin. If this capability is not needed, however, you can simply choose one of the pre-configured insulin definitions available.

3.3.1 Predefined Insulin/Release Pairs

Currently there are two pre-defined insulin/release pairs available in DMMS.R:

- Normal Insulin. This insulin has the release and post-release PK/PD characteristics of a typical rapid-acting insulin and can be given as a meal bolus or as basal by continuous subcutaneous micro-bolus through a pump.
- Standard Long-acting Basal Insulin. This insulin has the release and post-release PK/PD characteristics of a long-acting insulin analogue currently used in basal treatment for Type 1 diabetes.
- Ultra rapid-acting Insulin. This insulin has the release and post-release PK/PD characteristics of an ultra-fast rapid-acting insulin analogue and can be given as a meal bolus.

3.3.2 Defining New Insulin/Release Pairs

You may define additional insulins by combining a specific set of post-release PK/PD characteristics with a specific slow-release profile and bioavailability. (See Section 6.4.4)

Currently, the distributed version of DMMS supports several release profiles, but only one insulin analog, defined by the post-release PK/PD characteristics of normal insulin. Ultra-fast acting insulins are modeled to pass quickly through subcutaneous space by using short-term release profiles in conjunction with delivery directly to the plasma. In this way, the release profile essentially models the insulin's passage through subcutaneous space on its way to the plasma.

The Epsilon Group can work with clients to study newly developed or proposed insulins using the DMMS simulation platform. Such simulations would be based on client definition of the PK/PD chara cteristics of the insulin. For contact information, see Section 15.

3.3.3 Insulin Release Profile

When insulin is injected into an *in-silico* subject, it is always provided to a designated "compartment" within the subject. For the DMMS, this will always be either a compartment representing subcutaneous space or one representing the plasma (the latter only applies under special circumstances).

Insulins designed to release over a span of time (i.e., "slow release") are defined to have a "release profile." The release profile defines how much of the insulin injected at one instant (typically in a single bolus) enters the designated compartment (either subcutaneous space or the plasma) at each point in time following the injection. For example, a slow-release insulin's release profile may specify that 1% of the dose is delivered in the first half hour, 4.5% in each of the next 23 hours, and 1% in the final half hour of a 24-hour period.

The choice of whether the "designated compartment" is a subcutaneous space, or the plasma is built into the release profile's definition and is determined by how the insulin is modeled and the data from which

the model was derived. The “Standard Long-Acting” release profile provided with DMMS is designed for release to the plasma, while the other profiles (“Intermediate Acting” and “Perfect Long Acting”) are designed for delivery to subcutaneous space.

Once any given amount of the insulin has been released to the designated compartment in accordance with the release profile, the post-release PK/PD characteristics of this portion will be identical to what they would have been had the insulin been defined with no release profile. Note that you may define an insulin to have no release profile by deselecting the “Slow-release” checkbox on the “Insulin Definitions” tab (see Section 6.4.4). See Figures 6 and 7 for illustration of this model.

Figure 5: Insulin Release to Subcutaneous Compartment and **Error! Reference source not found.** shows the release of insulin to the subcutaneous and plasma compartments.

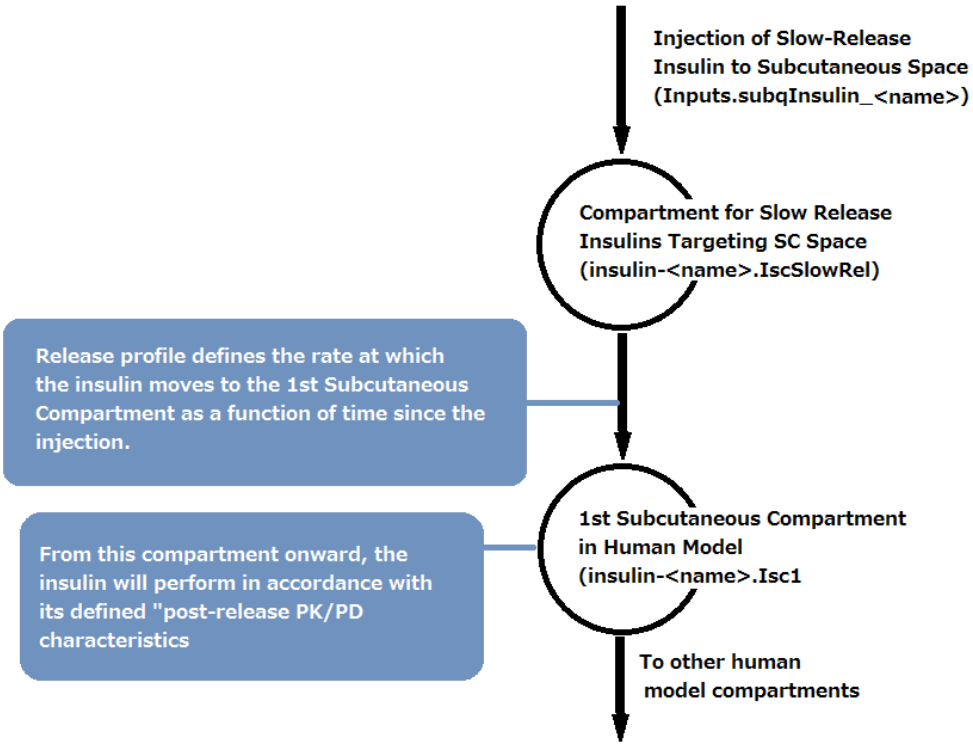


Figure 5: Insulin Release to Subcutaneous Compartment

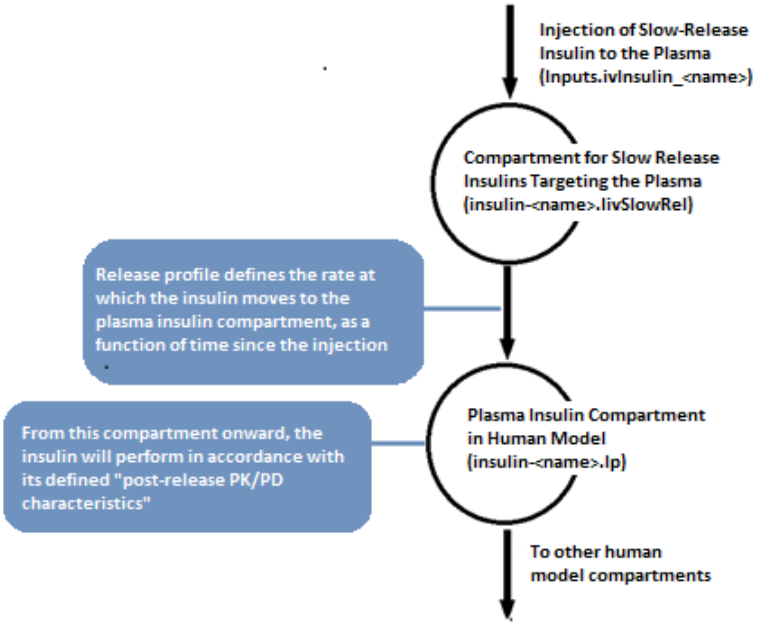


Figure 6: Insulin Release to Plasma

Note that when an insulin is defined as NOT using slow release, the 1st compartment in both diagrams is simply bypassed (i.e., the input signal goes directly to the "1st Subcutaneous Compartment" or to the "Plasma Insulin Compartment").

3.3.4 Insulin Bioavailability

The bioavailability specified in an insulin-release pair applies to both rapid acting and slow-release insulins, and always represents the fraction of the insulin that arrives in the plasma when delivered subcutaneously.

In the case of slow-release insulins with a release profile that is modeled to deliver to a subcutaneous compartment (see Figure 5: Insulin Release to Subcutaneous Compartment), the bioavailability-related loss (degradation) will take place between the 1st subcutaneous compartment and the "other human model compartments". In the case of slow-release insulins with a release profile modeled to deliver to the plasma (see Figure 6), the bioavailability-related loss is applied along with the release profile, as insulin moves out of the "compartment for slow-release insulins targeting the plasma."

For subcutaneous insulin infusion of insulins that do NOT use slow release as bioavailability losses are applied in the same manner as for the slow-release insulins with release profiles targeting subcutaneous space (as in Figure 5: Insulin Release to Subcutaneous Compartment).

For IV insulin infusion of insulins that do NOT use slow release, no bioavailability losses are applied.

3.4 Insulin on Board (IOB)

In general, Insulin On Board (IOB) refers to the amount of active insulin in the body. The DMMS.R simulation model calculates this value for the selected virtual subject as the simulation progresses.

The amount of IOB present at any given time is the result of:

1. The remaining insulin before the most recent meal bolus or correction bolus.
2. …plus the amount of the last meal bolus or correction bolus.
3. …minus the amount of insulin metabolized by the body since the last meal bolus or correction bolus.

A trace showing typical levels of IOB is shown in Figure 7.

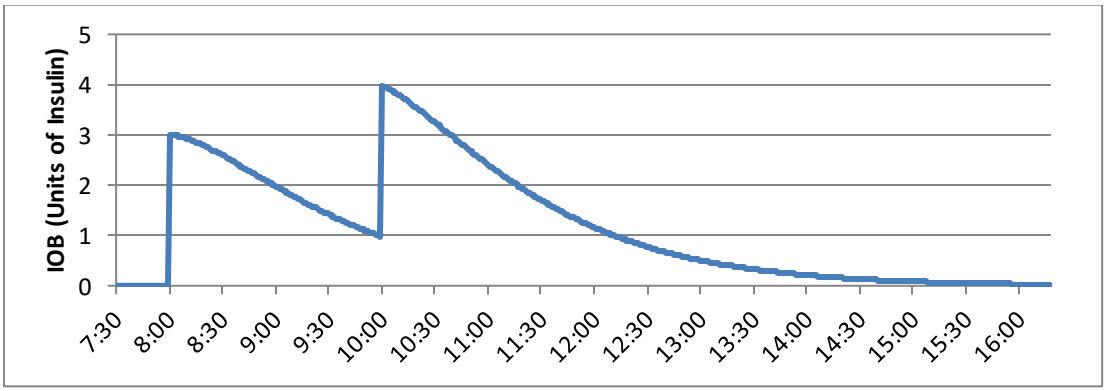


Figure 7: Insulin on Board (IOB)

In this example IOB spikes at 8:00 as the result of a meal bolus. IOB then declines until 10:00 when another bolus is given for a snack. After that, IOB decreases throughout the day. By 15:00 there is virtually no available insulin remaining.

DMMS.R calculates the IOB resulting from basal insulin as well as that from boluses. Each can be seen separately in the output.

3.5 Circadian Rhythm

The intrinsic circadian timekeeping system modulates many physiological systems, including daily rhythms in core body temperature, cortisol, and appetite. More importantly, it contributes to the variability in diurnal patterns of insulin sensitivity (S_I) in the early morning, and postprandially. This results in a lower S_I at breakfast than lunch and dinner and contributes to inter-and intraday variability.

This S_I reflects an equal adjustment of the time-varying pattern across the population (not per subject variation), in a stepwise line signal that varies three times a day (at 4 a.m., 11 a.m., and 5 p.m.).

This is found under the randomization options on the *General* tab of the main dialog (See Section 6.4.1).

3.6 Plugin Elements

In addition to the built-in elements provided, DMMS supports JavaScript and MATLAB® plugin elements. These may be sensor elements, control elements, or delivery elements. Plugins may modify the behavior of existing elements or may provide entirely new functionality. You may create plugins from scratch or modify existing plugins.

A JavaScript plugin element can be designed to call a Web Service (see section 9.1.5). Its purpose is to give you capability beyond the limitations of JavaScript, as the web service can be implemented in virtually any language.

The creation or modification of DMMS plugins requires a working knowledge of JavaScript or MATLAB programming. Use of MATLAB for plugins requires MATLAB to be installed on the same computer as the DMMS. Use of JavaScript, however, requires nothing in addition to the DMMS application.

For more detail on plugin element configuration, see Sections 6.4.6.1.5.5, 6.4.6.2.5.9, 6.4.6.2.5.10, and 6.4.6.3.6. For details on writing code for a plugin see Section 9.

3.7 Randomization

Randomization mimics the natural variation in empirical data or generated signals. DMMS employs the concept of randomization to make longer simulations more life-like. Randomization may be applied to carbohydrate intake estimation (carb counting) as well as to sensor measurements and delivery elements such as pumps (noise). In addition, randomization may be included in the design of JavaScript plugins.

In the case of carbohydrate estimation, you may specify randomization as a percentage. For sensors and delivery elements, you may elect to configure and apply Gaussian white noise to the signal. The General Configurable Sensor also supports an Auto Regressive Noise function.

Gaussian White Noise requires three parameters:

- A random seed: A seed for generation of the random number generated by the function. By using a different seed from one simulation to the next, you can, for example, establish the statistical ranges of outcomes resulting from the same scenario defined in the simulation's configuration. By running with the same seed, you can ensure repeatability.
- Noise sample time: The period (in minutes) at which the noise values are generated. Decreasing the sample time increases the total number of samples generated. DMMS includes a smoothing function to limit the difference between one sample and the next.
- Standard deviation: Standard deviation of the noise distribution. Controls the spread of the deviation. Higher standard deviation increases the magnitude of the noise signal. The distribution is centered on zero.

® MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

To simplify the control of simulations in which multiple sensors, control elements, or delivery elements use randomization, the DMMS provides a means of setting a random seed that will be used by all these components. Using this "common seed," you can change just one setting to alter the random sequence for every element that uses randomization. You can ensure repeatability by restoring the common seed to a previous value.

In addition, you can choose a setting that forces the common seed to be unique for each subject in the simulation. When you choose this, the seed you specify is a "base seed" used to compute a unique seed for each subject. This computed "subject-specific seed" will still be common to all elements for any given subject. With this feature, you can ensure better randomization across the population. For example, you will not see a specific pattern of noise associated with a given sensor, duplicated over the same time span for each subject in a simulation. You can still guarantee repeatability by choosing the same base seed.

There may be some cases where you would like to see what happens when each subject experiences the exact same timing of randomness. When this applies, you can avoid making the seed unique for each subject, but still use the "common seed" feature. This would allow you to see, for example, how all subjects respond when they all get correspondingly high and low readings from their SMBG meters at the same times.

These randomization options are set on the *General* tab of the main dialog (See Section 6.4.1).

4 Installation and Licensing

This manual assumes you already have DMMS.R installed on your computer. If you are unsure as to whether you have the application installed, go to section 6 and follow the instructions.

The Epsilon Group requires that each installation of DMMS.R be licensed.

For instructions for installation and licensing of DMMS.R on a Windows device, see Guide to Installing TEG Software for Windows. To determine the version of DMMS.R installed on your computer, see Section 13.

5 Files and File Locations

5.1 File Access

DMMS provides access to various files via the standard file dialogs for the Windows platform. The reader is assumed to be familiar with retrieving and saving files using these dialogs on their computer.

5.2 Files Written during Installation

During the installation process, the DMMS.R installer utility will copy two sets of files into two separate locations of your local hard drive:

- **Program Files.** Files written here are intended to be accessed only by the application. You should not remove, rename, or modify these files.

- **User Files.** Each user has a different User Files location. These files may be modified, and you may add new files to this area. On installation, these files will be a subset of files written into the application files area.

On the Windows operating system the paths to these two locations, by default, are:

- **Program Files:** "C:\Program Files\The Epsilon Group\DMMS.R\…."
- **User Files:** "C:\Users\<User Name>\DMMS.R\…."

The person handling the installation may specify an alternative location for the application files area. While running DMMS.R the user may elect to read/write User Files to a different location.

See Guide to Installing TEG Software for Windows for more detail. While using DMMS, you may direct file output to a folder of your choice.

5.3 Subject Parameter Files

Subject parameter files are written to the "…\data folder" in both the Application Files location and the User Files location when DMMS is installed. This data is intended for your reference only; modification of these files has no effect on the application. On a Windows platform, the subject parameter files are written to "C:\Users\<User Name>\Documents\DMMS.R\data," when DMMS is installed.

For each subject group (pre-diabetic adult; Type 1 adult, adolescent, child; Type 2 adult) the following data is written to the file:

Item	Description	Units	Applies to
Name	Subject identifier (e.g., adolescent#004)	None	all
Gb	Basal (fasting) blood glucose	mg/dL	all[3]
BW	Body weight	Kg	all
Age	Subject's chronological age	Years	all
Optimal Daily Basal Insulin		Units	Type 1 only
CR	Carbohydrate ratio	g/unit of insulin	Type 1 only
CF	Correction factor	mg/dL/Unit of Insulin	Type 1 only
OGTT2	Results of a 2-hour oral glucose tolerance test	mg/dL	Type 2 and Pre-diabetics only

Table 2: Subject Parameters

5.4 Configuration Files

The data that you enter into DMMS.R may be saved in configuration files. This allows you to repeat a previous simulation, perhaps with a few minor changes, without reentering all the data.

[3] For Type 1 subjects, the amount calculated using Optimal Daily Basal Insulin; For Type 2, the actual blood glucose "reading" for this subject.

In general, configuration files are created at run time, with a name that you specify (see Section 8.1). By default, on a Windows platform, these files are written to "C:\Users\<User Name>\Documents\DMMS.R\config."

You may change these default locations. On installation, two example configuration files (config_T1_3meals_mealBoluses.xml and config_T2_3meals_metformin.xml) will be written to the default location.

Configuration files may be saved and loaded using the first three sub-items under *File* on the main menu bar (See Section 8.1).

While configuration files may be modified by an experienced user using a text editor, it is strongly recommended that to make any required configuration changes, use this process:

1. Load the configuration (Rx plan) into the application.
2. Make any required changes using the user interface.
3. Save the Rx plan back to the file.

5.5 JavaScript and MATLAB Script Files

DMMS supports user-definition of "plug-in" feedback elements using JavaScript or MATLAB. On installation, example implementation files for both JavaScript and MATLAB (the MATLAB files have similar names, but with a "_matlab" extension) and example setup files for MATLAB are written into your installation "config" directory. If you create any feedback elements using JavaScript or MATLAB, you must put the associated implementation and setup files in this directory.

For more detail, see Section 9 and the JavaScript and MATLAB subsections under the individual configuration sections for sensors, controls, and delivery elements.

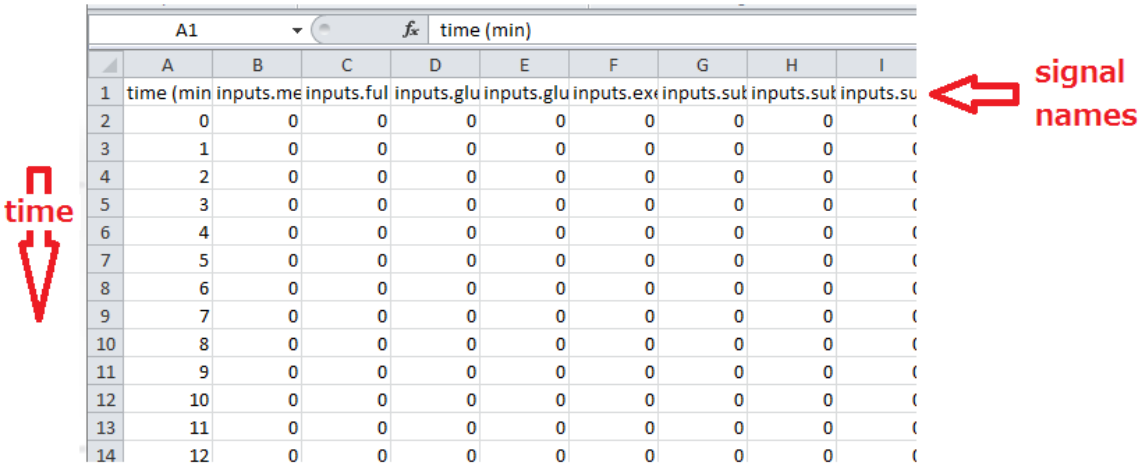
5.6 Simulation Output Files

Data output from DMMS.R may be saved for future reference and analysis. By default, data output files are written to, and read from, the "…/results" folder in your User Files location. You may, however, change the path to this location (See Section 6.3.1.1).

Simulation files may be accessed using sub-items under File on the main menu bar.

5.6.1 Signal History Files

By default, each time you run a simulation, a .csv (comma separated values) file is generated for each subject. These files contain a minute-by-minute snapshot of the subject's metabolic state data. For a list of the values saved in this file, along with their units, see Appendix A: Output Data Definition.



	A	B	C	D	E	F	G	H	I
1	time (min	inputs.me	inputs.ful	inputs.glu	inputs.glu	inputs.ex	inputs.su	inputs.su	inputs.su
2	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0
8	6	0	0	0	0	0	0	0	0
9	7	0	0	0	0	0	0	0	0
10	8	0	0	0	0	0	0	0	0
11	9	0	0	0	0	0	0	0	0
12	10	0	0	0	0	0	0	0	0
13	11	0	0	0	0	0	0	0	0
14	12	0	0	0	0	0	0	0	0

Figure 8: Signal History File

5.6.2 MATLAB Signal History Files

In addition to, or in place of, the .csv signal history files, the DMMS.R can save the minute-by-minute metabolic state data for each subject to a MATLAB data file (with .mat extension). This can be done automatically, via choices you make in the configuration file (see section 6.4.7) or can be done manually through the DMMS.R's menus (see section 7.7.2).

5.6.3 Plot Files

You may save Signal Trace Plots, CVGA Plots, or Summary Trace Plots to scalable vector graphics (.svg) files or portable document format (.pdf) files. Each plot will be saved in the same form in which it is displayed. See Section 6.3.1 and Section 7.1.1 through 7.3.

5.6.4 Metrics Files

You may save individual subject or population metrics derived from simulations to pdf files. See Sections 6.3.1, 7.4 and 7.5.

6 Using DMMS.R

Using DMMS.R consists of 4 basic steps:

1. Selecting a subject, or subject group, and defining the time span of the simulation.
2. Configuring sensor, control, and feedback elements
3. Running the simulation
4. Analyzing the results

Each of these steps is supported by the application's user interface.

6.1 Getting Started on Windows

To begin, navigate to the Epsilon Group folder in your App Menu (Windows 8 and 10) or Program Menu (Windows 7). You should see something like Figure 9:

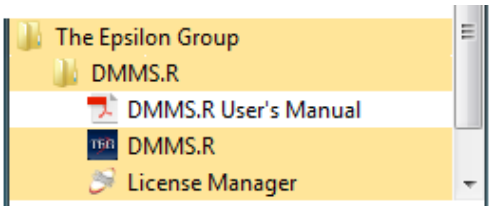


Figure 9: DMMS.R in Program Menu

Double-click the DMMS.R application icon[4]. This will cause the main dialog to appear as in Figure 10.

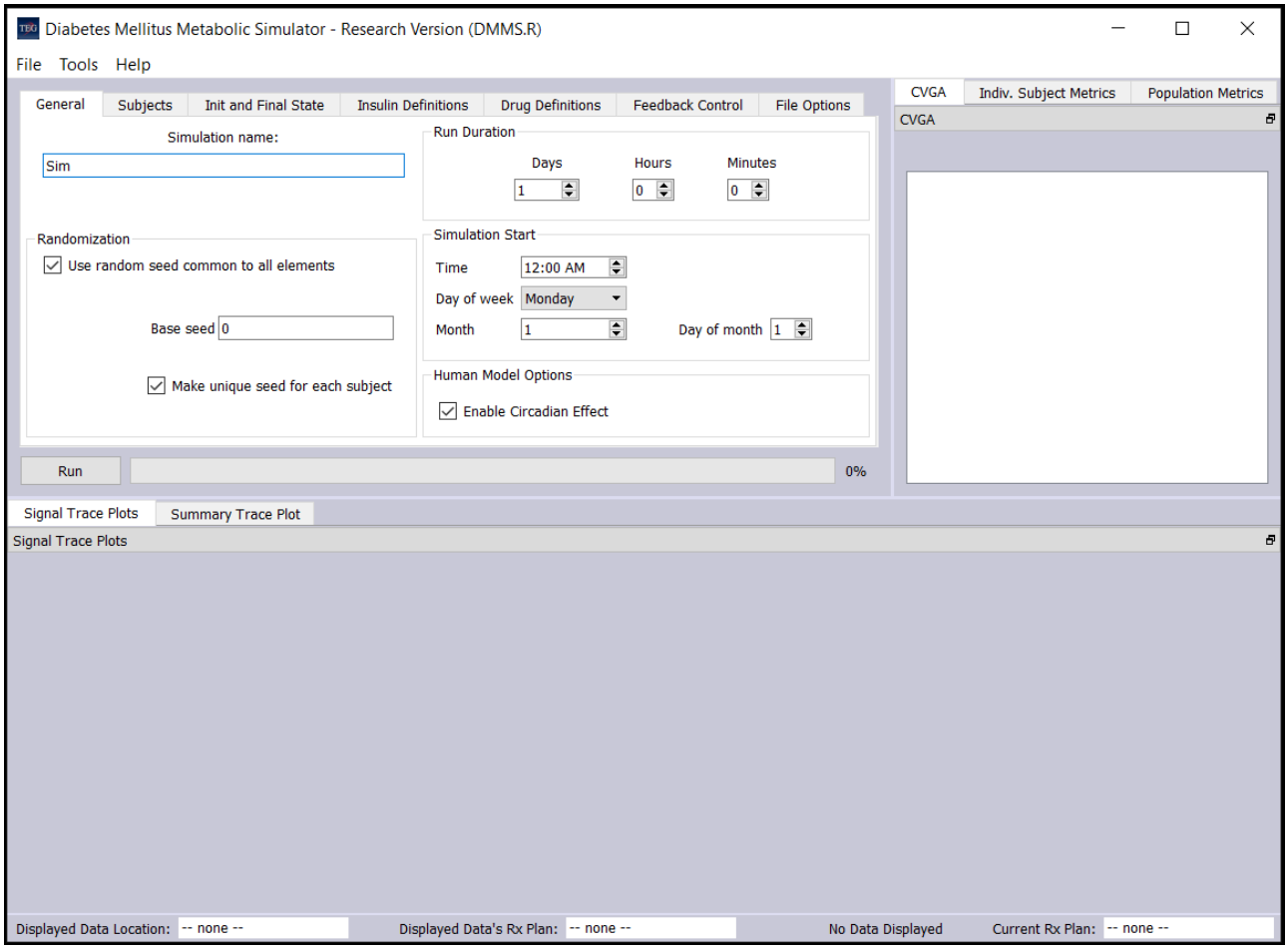


Figure 10: DMMS.R Dialog on Startup (Windows)

[4] The appearance of the start menu may vary slightly on your computer. However, if you cannot locate the DMMS.R application icon or User's Manual, or if the dialog in Figure 9 does not appear, the program has not been correctly installed. Please contact The Epsilon Group for assistance (See Section 15 for contact information).

The dialog display contains five distinct functional areas:

- A top-level menu
- Tabs to allow you to configure a simulation (duration, subject selection, Rx plans, control elements, etc.).
- A button and progress bar to run and monitor the simulation.
- The output metrics display area.
- The output plot display area

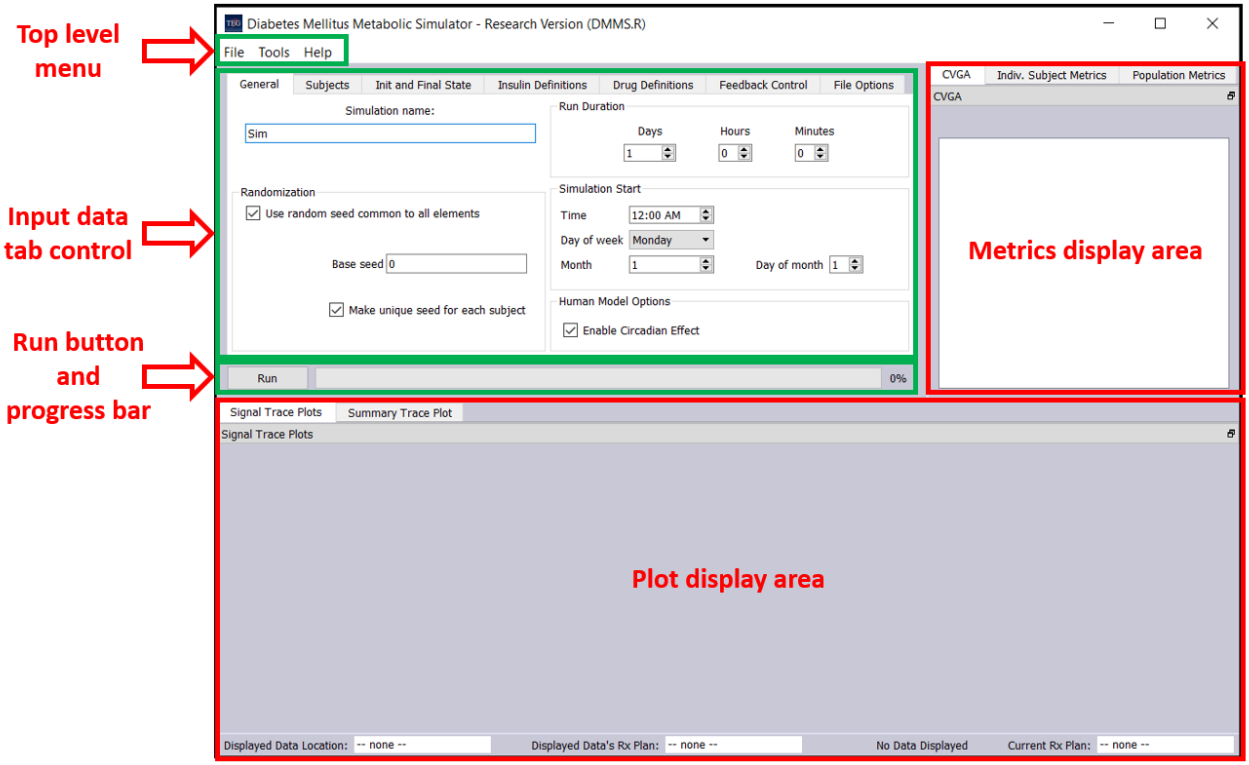


Figure 11: DMMS.R Dialog Sections (Windows)

In Figure 11, the areas outlined in green contain the dialog controls that allow you to make the inputs you need to design your simulation; the red-outlined areas contain the displayed output.
The dialog display contains five distinct functional areas:

- Top level menu
- Tabs for simulation configuration (Duration, Subject selection, Rx plans, Control elements, etc.)
- Run button and progress bar
- Output metrics display area
- Output plot display area

6.2 General User Interface Features

6.2.1 Dependencies among Dialog Controls

In general, each dialog control is independent of any other. There are a few instances, however, where dialog controls may be at least partially inter-dependent:

- Some dialog controls are not enabled, or not even visible, unless some other dialog control (typically a check box) has a specific setting (e.g., is checked).
- Allowable input to one dialog control may be dependent on the setting of another.

You may open the tabs and choose options on them in any order and may freely change your choices at any time prior to running the simulation. However, it is usually "safest" to work through the tabs from left to right. This is most applicable to the insulin definitions (which are often referenced from configured feedback control elements. See Section 6.4.4) and, within the feedback control tab, when setting up sensors whose output signals may be referenced by control elements (See Section 6.4.6.1.4).

6.2.2 Tooltips

While the purpose and function of most of the application's dialog controls are obvious, a few may require a short explanation. DMMS.R supports this requirement with tooltips. To reveal the tooltip for a dialog control, simply allow the cursor to hover over it for a second or two, and a tooltip, if one is available, will appear (see Figure 12 below).

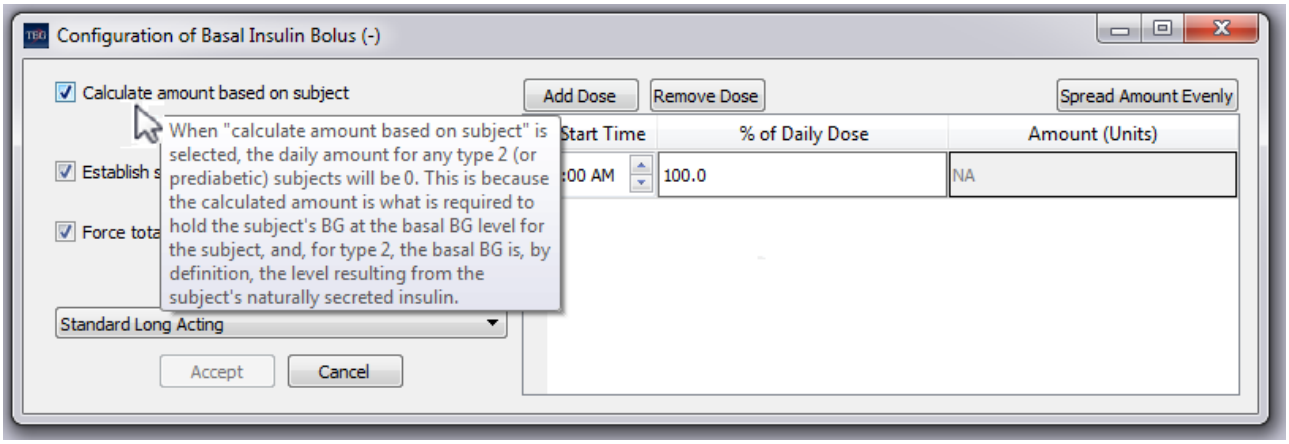


Figure 12: Tool Tip

6.2.3 Dockable Display Elements

Some of DMMS's GUI display elements are dockable; they can be moved from their default positions as needed. For example, the CVGA, Individual Metrics, and Population Metrics displays are all contained in the same tab dialog control (See "Metrics display area" in Figure 11 above). By default, only one of the three is visible. However, you can "un-dock" each and move it to another area of your screen to show two, or even all three, at the same time.

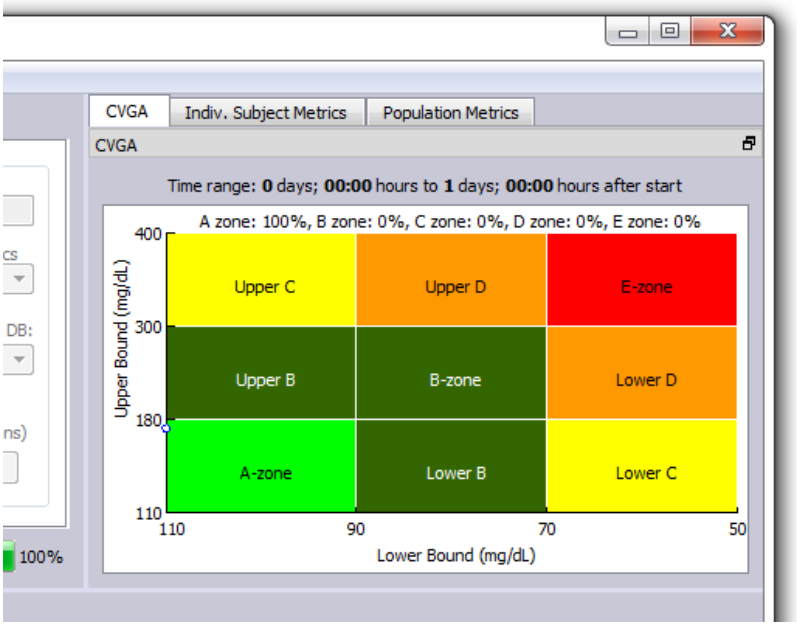


Figure 13: Docked Displays

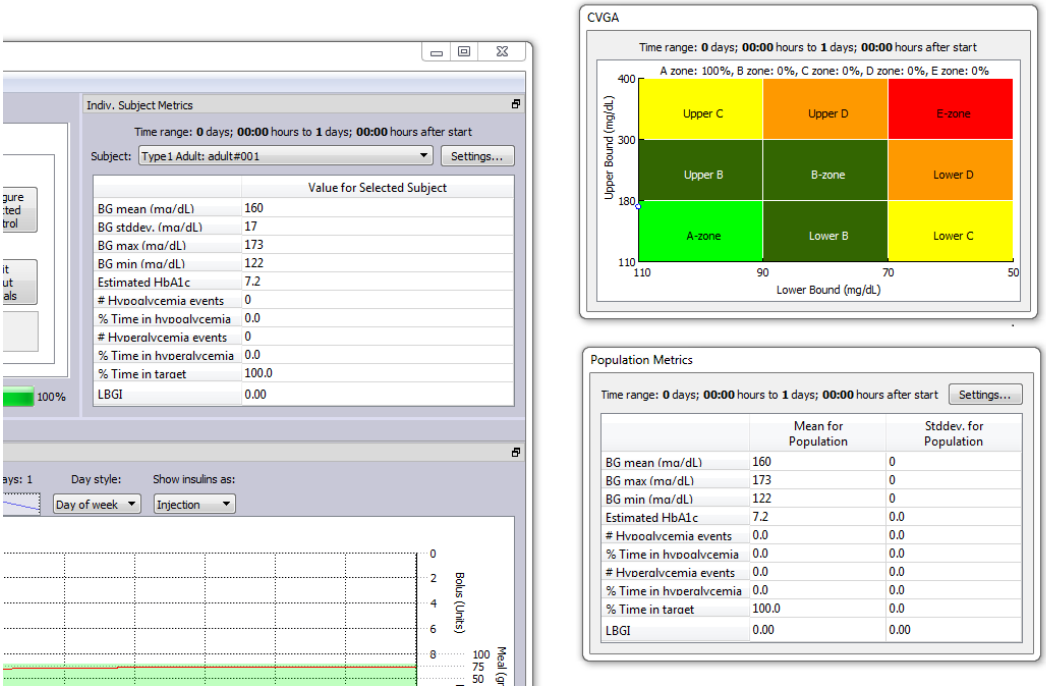


Figure 14: Un-docked Displays

To “un-dock” a display element, depress the left mouse button on its title bar and drag it to another area of your screen. To “re-dock” a display element, double click on its title bar.

Another feature of these dockable elements is that you can re-order the tabs within the tab control container by dragging with the mouse.

Dockable display elements also include the Summary Trace Plot and the Signal Trace Plot (See "Plot display area" in Figure 11 above).

6.3 Top Level Menu

On a Windows computer, DMMS.R's top-level menu contains three entries: *File, Tools*, and *Help*. The top-level menu is part of the main DMMS.R GUI as in Figure 15:

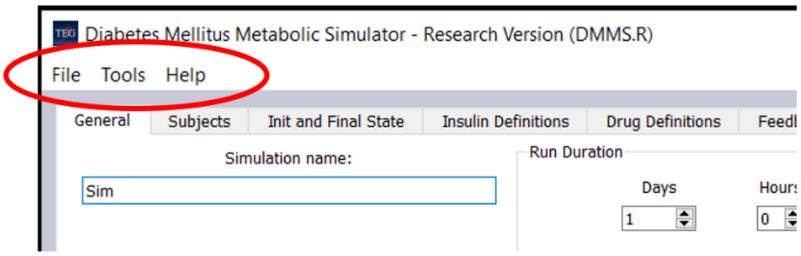


Figure 15: : Top Level Menu on Windows

6.3.1 File

Selections under the File menu allow you to save and retrieve configuration data (Rx plans) as well as the data files that DMMS generates.

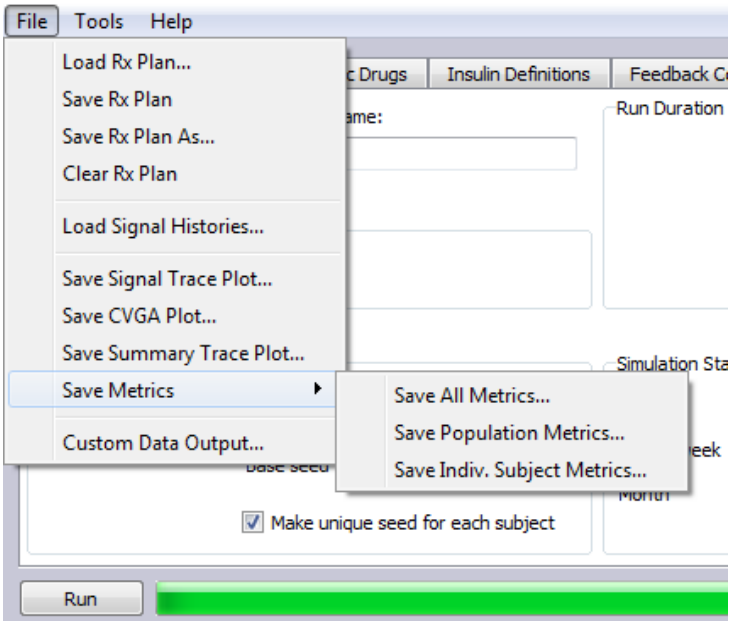


Figure 16: File Menu

For information on the use of these menu items, see Sections 7.7 and 8.1.

6.3.1 Tools

The Tools menu contains three sub-items:

6.3.1.1 Settings

This option will display a dialog to permit you to define the default and current file directory for configuration files and data output files.

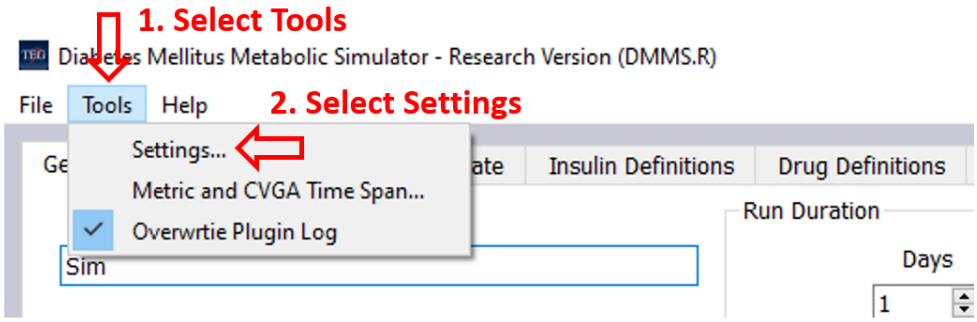


Figure 17: Selecting Settings

In the *Setting Set Selection* section, the user has two options to set their setting, Default and Current. By selecting the Default settings, the default directory will be shown in the *Default Configuration File Directory* and the *Default Results Directory*. For the first installation, the default directory will be *C:\Users\<User Name>\Documents\DMMS.R.*

To change the default directory, simply click the *Browse* buttons to select a preferred directory for both configuration and results folder, then click the *Save Defaults* button to save your change(s).

If you want to work on a different directory but not willing to change the default settings, you can select Current Settings in the *Setting Set Select* section. By selecting this option, you will be able to select a preferred current working directory without the need to change the default directory. You are also allowed to switch back to the default directory in the current settings by simply clicking the *Copy From Defaults* button. See Figures 18 and 19.

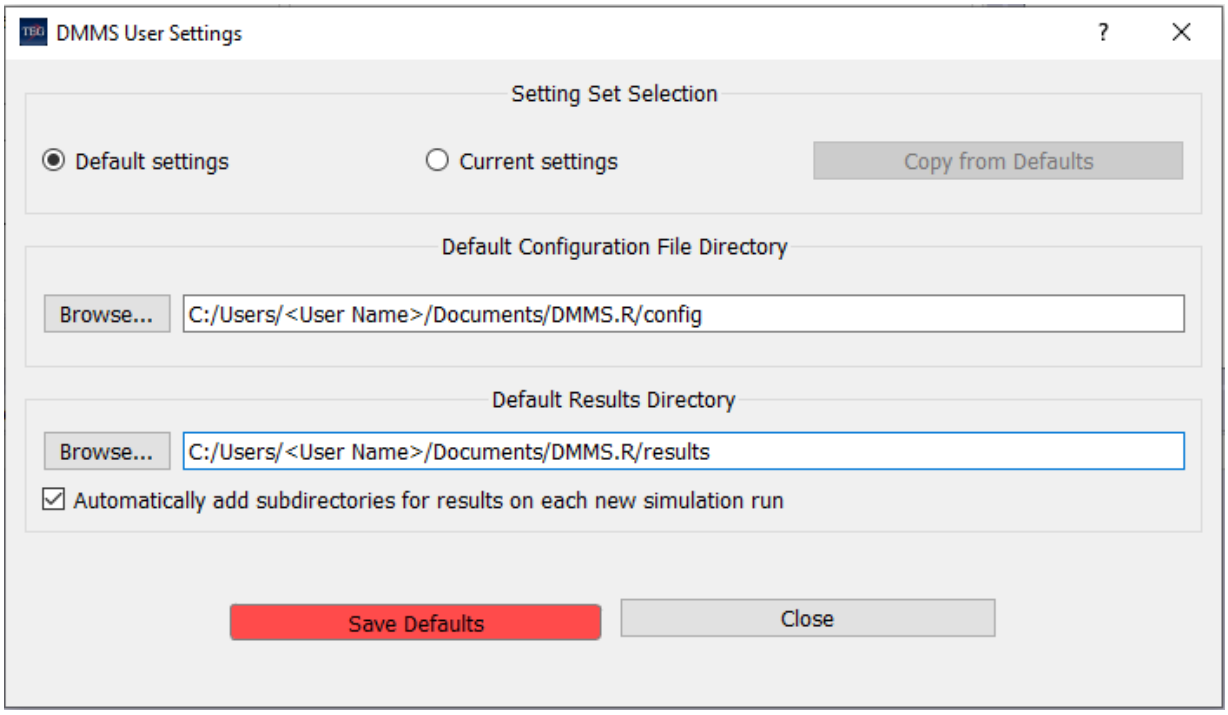


Figure 18: User Settings Window for Default settings

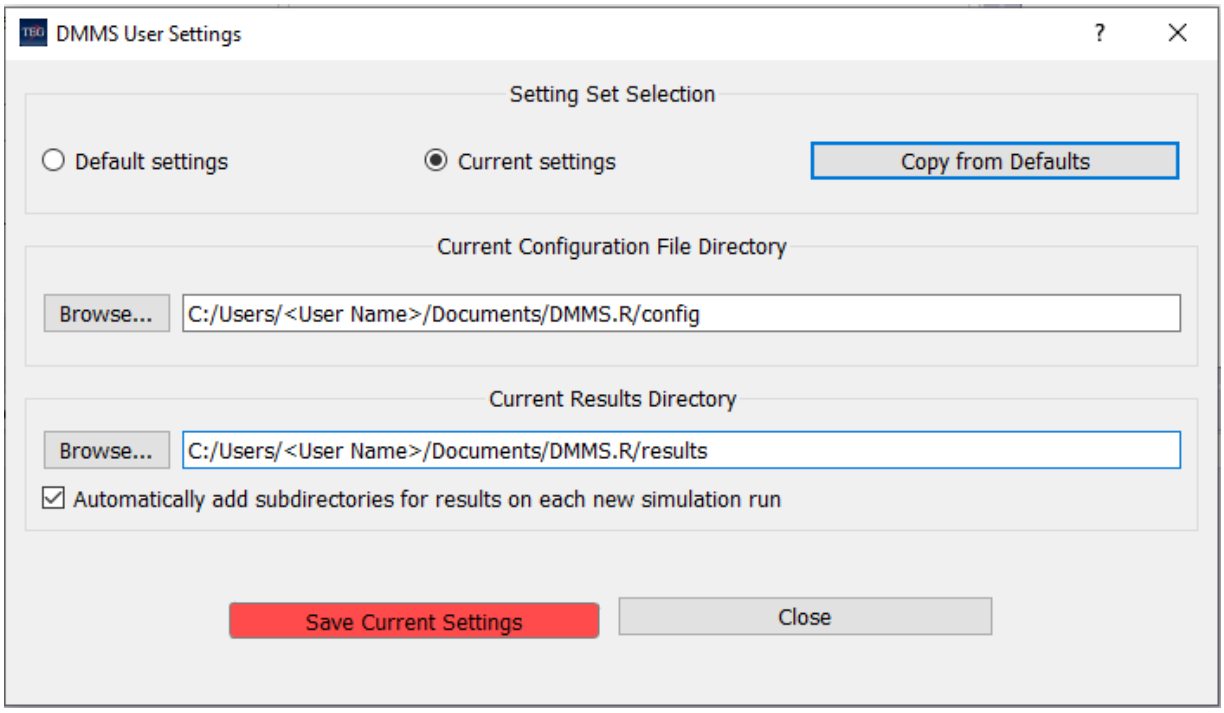


Figure 19: User Settings Window for Current settings

Note that these settings are persistent: if you were to run subsequent simulations, they would all write their results files into the root directory that you most recently defined, even if that definition occurred in a previous working session.

If the "*Automatically add subdirectories…*" checkbox is checked, each simulation run results in a new folder being created under the "results" root. If it is unchecked, the results from multiple simulations will always be written in the same folder overwriting any previously generated data files.

As an example, specify a *Default Results Directory* named "…Users/<your user name>/Documents/DMMS.R/myResults." Now run 3 simulations, retaining the default simulation name of "Sim", and leave the "Automatically add subdirectories…" box checked. In the "…myResults" folder, you should see 3 folders named Sim, Sim1, Sim2. You would find the output files for your first simulation under Sim, the second under Sim1, and the third under Sim2 Now uncheck the checkbox and run another simulation. This time the output files are written under Sim; any files previously saved to this folder will be overwritten.

6.3.1.2 Metric and CVGA Time Span

You may limit the time span used to calculate the Metrics and CVGA data. Select *Metric and CVGA Time Span…* to access the dialog explained in section 7.6.

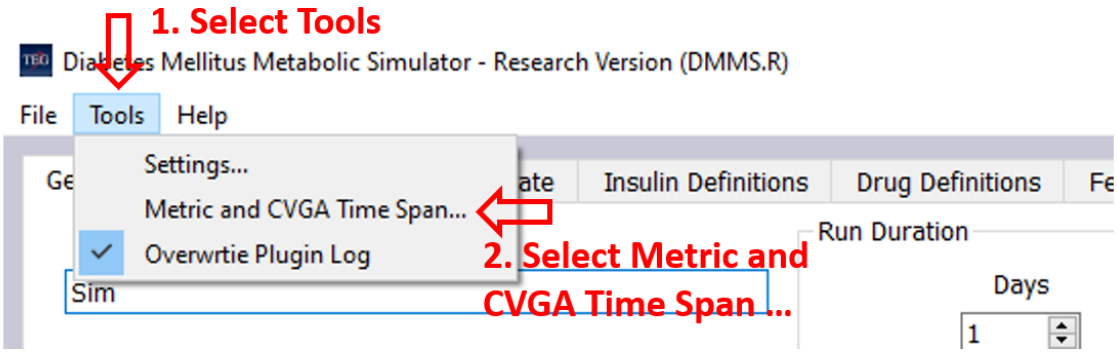


Figure 20: Selecting the Metric and CVGA Time Span

6.3.1.3 Overwrite Plugin Log

You can select either keeping one copy of the previous plugin log file if any (default) or generate a new plugin log file for every simulation with an incremental index attached at the end of the file name.

When the checkable menu item *Overwrite Plugin Log* is checked, the plugin log will be saved in the same folder as the plugin used during the simulation, and will be overwritten for every simulation run; when it is unchecked, a new plugin log will be generated for every simulation run and be saved in the same folder as the plugin used during the simulation.

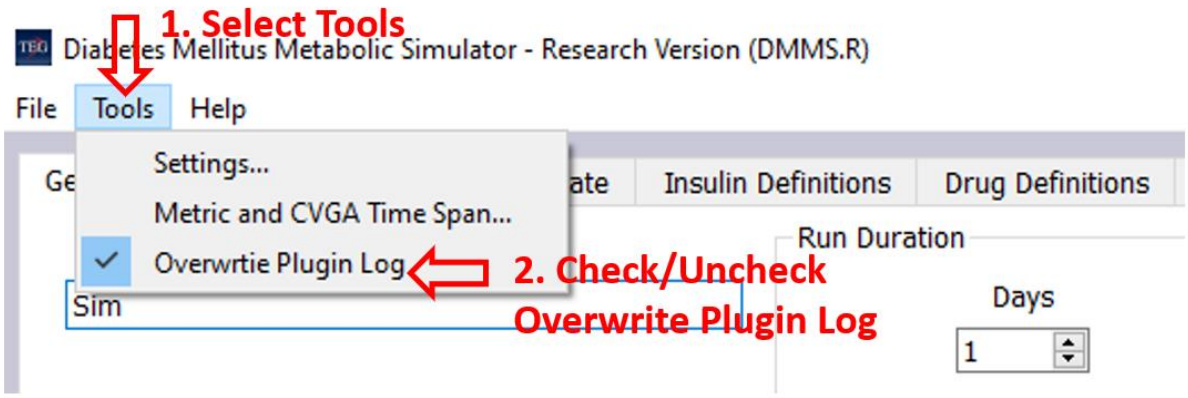


Figure 21: Check/Uncheck overwrite plugin log

6.3.2 Help

The Help menu item contains direct link to open the user manual and the information about the version of DMMS.R installed.

6.3.2.1 Open User's Manual

You can open the DMMS.R User's Manual directly through the application. To open the user's manual, click the *Help* drop down menu, then select Open User's Manual, as shown in Figure 22, the user's manual should open directly.

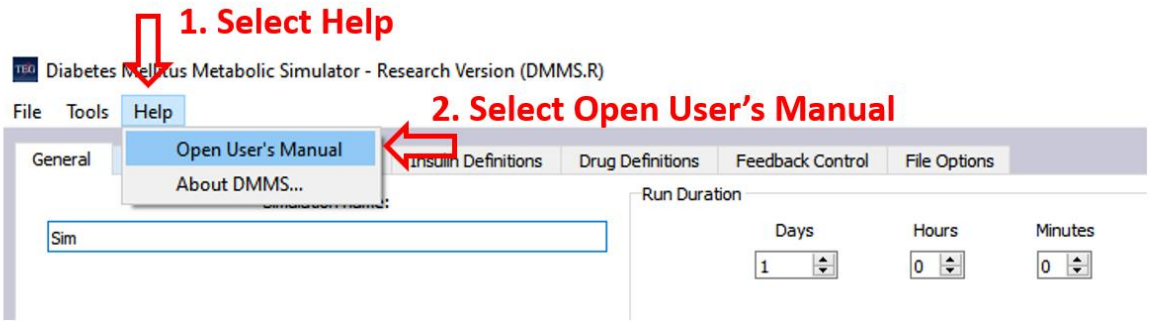


Figure 22: Open User's Manual

6.3.2.2 About DMMS…

You can learn about the application information by selecting "*About DMMS"*. Please see Section 13 for more information.

6.4 Input Data Tab Control

The application presents a tab control with these tabs:

Tab Label	See Section…	Page
General	6.4.1 (Defining General Simulation Options)	36
Subjects	6.4.2 (Selecting the Subject Group)	36
Subject Initial State	6.4.3 (Configuring Subjects' Initial State	39
Insulin Definitions	6.4.4 (Defining Insulin)	40
Anti-diabetic Drugs	6.4.5 (Drug Definition	44
Feedback Control	6.4.6 (Configuring Feedback Control)	43
File Options	6.4.7 (Configuring Options for Saving Signal Histories)	82

Table 3: Input Data Tab Control

6.4.1 Defining General Simulation Options

The General tab allows you to control the length of the simulation and the time at which it will start.

It also contains dialog controls for defining the name of the simulation (any output files will be written to a folder using this name) and for controlling randomness in the simulation.

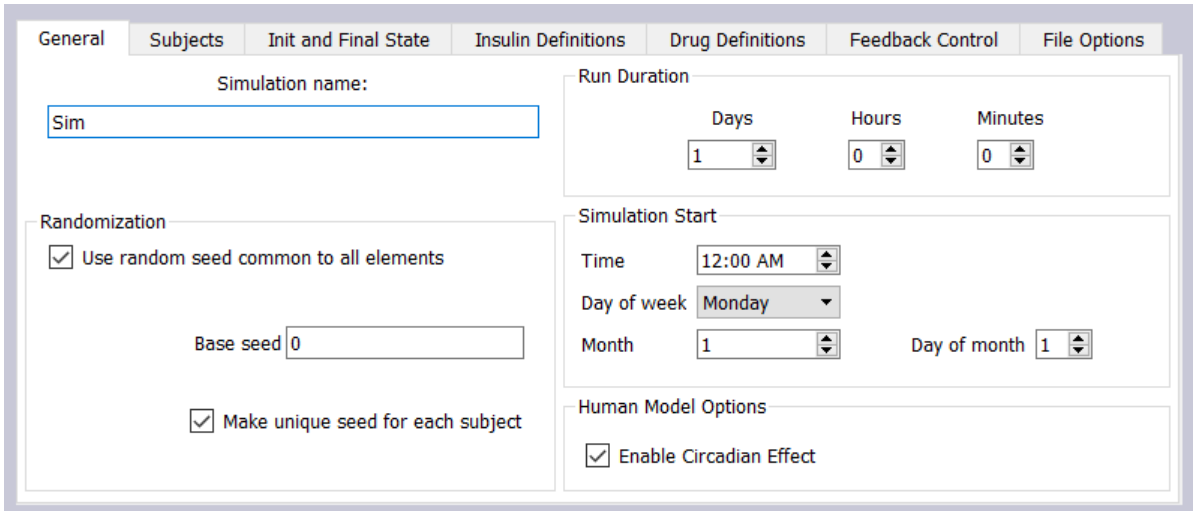


Figure 23: General Input Data Tab

Dialog controls in the *Randomization* group box allow you to set parameters affecting every random process within the simulation. When *Use random seed common to all elements* is checked, the seed specified on this tab will override any random seed set in the configuration of an individual sensor, control element, or delivery element. These parameters are explained in section 3.7.

To enable the circadian effect mentioned in section 3.7, simply click the *Enable Circadian Effect* check box in the *Human Model Options* section.

6.4.2 Selecting the Subject Group

The Subjects tab permits you to choose the subject or subjects to include in the simulation. There are three steps involved:

1. In the *Available Subjects* group box, choose the subject population to work with. *The Subject* list box will be filled with subjects from that population.
2. Choose one or more subjects by highlighting them individually or by pressing *Select All*
3. Press *Add* to include your selected subjects in the simulation.

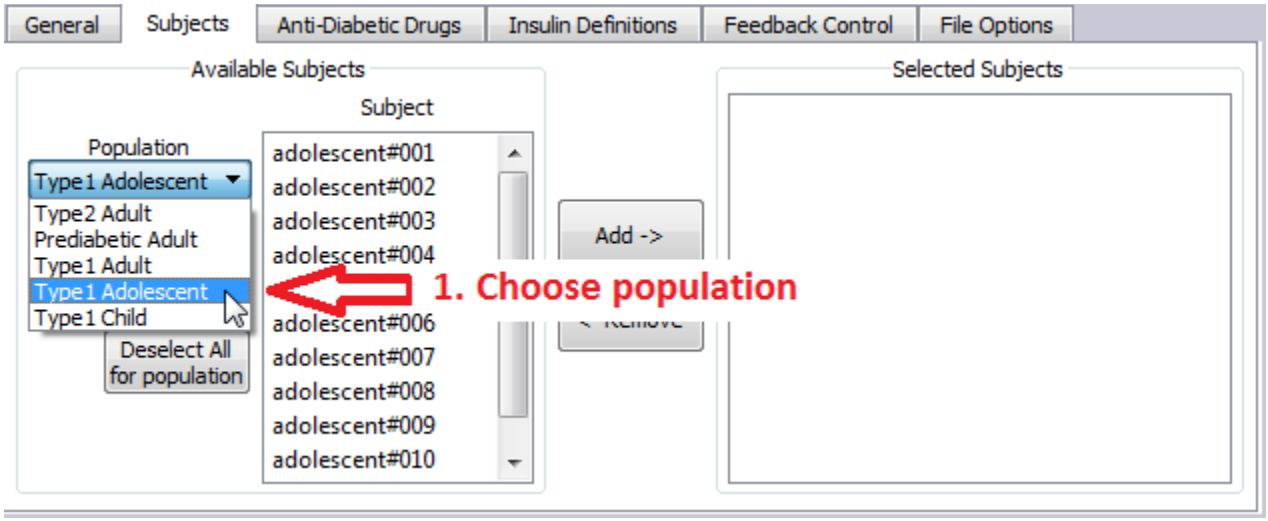


Figure 24: Selecting a Subject Population

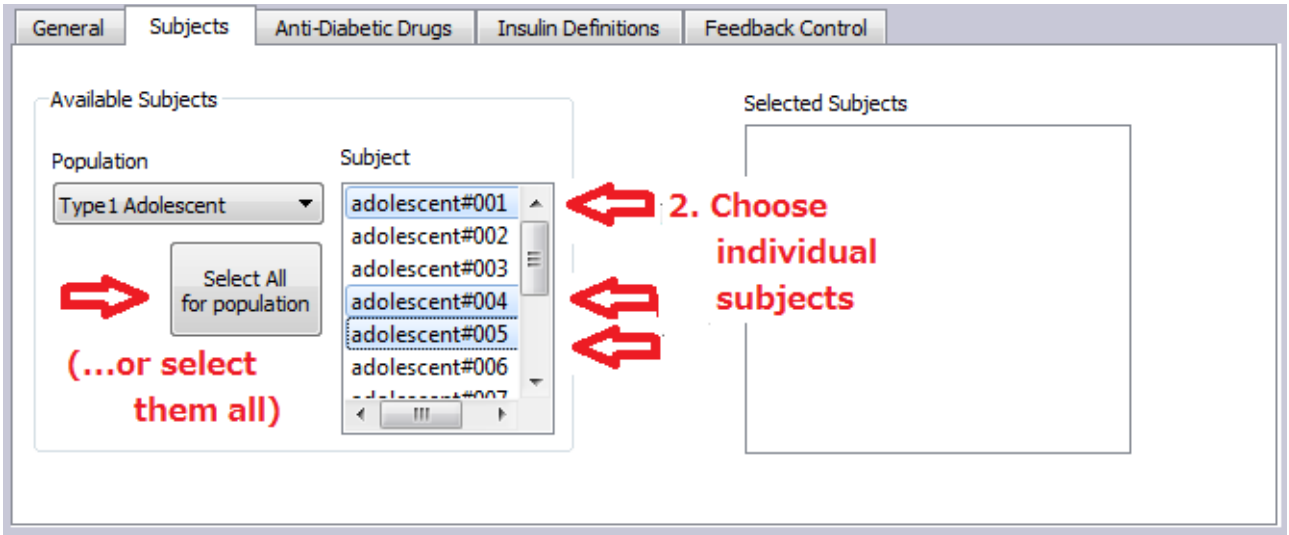


Figure 25: Selecting Subjects

Within the *Available Subjects* group box, the *Deselect All* button will clear any subjects you have selected from the available subjects list. Note that this will not affect the Selected Subjects group box on the right.

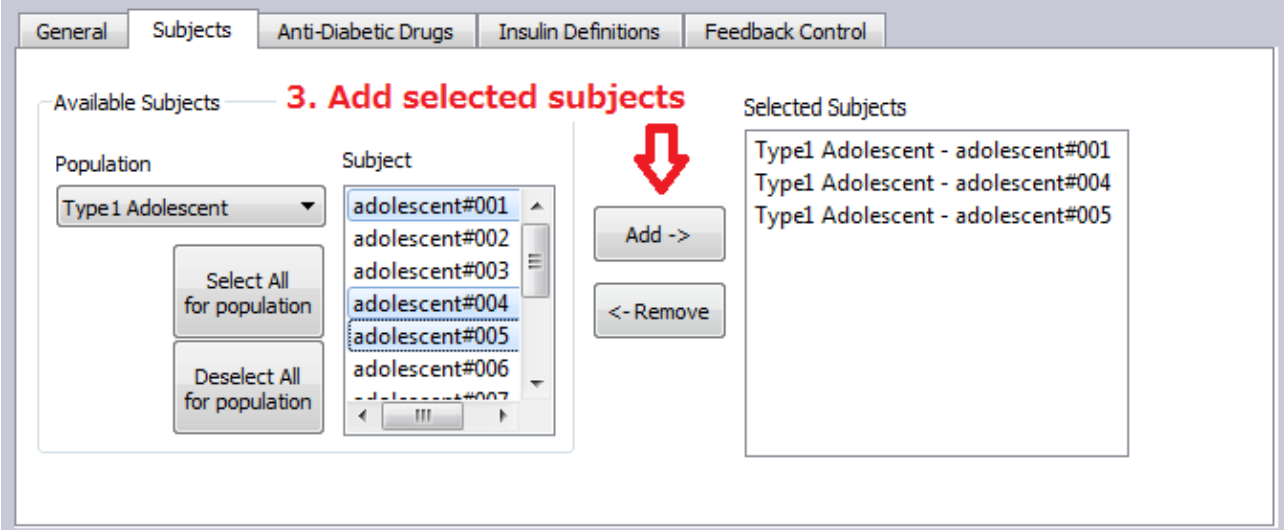


Figure 26: Adding Subjects

You may remove any Selected Subjects by highlighting them and pressing the *Remove* button.

Note that it's possible to create a group of subjects from various populations. Just repeat the steps shown above for each desired population, in turn.

6.4.3 Configuring Subjects' Initial State

The *Init and Final State* tab lets you define how the subjects' states are initialized, and whether the final state of each subject is saved to a file. Saving the final states to a file will allow you to use these states as a starting point for subsequent simulations.

6.4.3.1 Specifying an Initial BG

The simplest way to establish the subjects' initial state is to force their initial BG to a specified value. This is done by checking the *Force initial BG* checkbox and entering the desired concentration in the corresponding data entry field. Note: Currently, this feature is only supported for Type 1 subjects – if Type 2 subjects are included in a simulation with *Force initial BG* selected, the DMMS.R will display an error when you attempt to run the simulation.

6.4.3.2 Loading and Saving Subjects' Complete States

You can also use a comma separated value (CSV) file to specify the complete initial state you would like for each of the subjects in the simulation. This file will have one row of values for each subject, and will have a subset of the columns found in the signal history files (see section 5.6.1 and Appendix A: Output Data Definition). The subset includes all signal names beginning with "subj." or with "insulin-". If user-defined insulins are included in the simulation, there will also be columns for concentration data and IOB for each of these.

To specify a CSV file for initializing the subjects' states, use the *Load each subject's full initial state from a file* checkbox, and the controls immediately below it to type or browse for the full path of the file to be

used. The file can be specified by either an absolute or relative path. If a relative path is selected, the path is considered to be relative to your default results directory (see section 6.3.1.1).

The easiest way to create a CSV file for use in initializing subjects' states is to first run a simulation with *Save each subject's full final state to a file* checked. The resulting file will be of the needed format and will include all the desired columns. Note that the set of user-defined insulins (if any) must be identical to those configured for the simulation used to load this file. Also, the simulation used to load the file should only include subjects that are also in the simulation used to save the file.

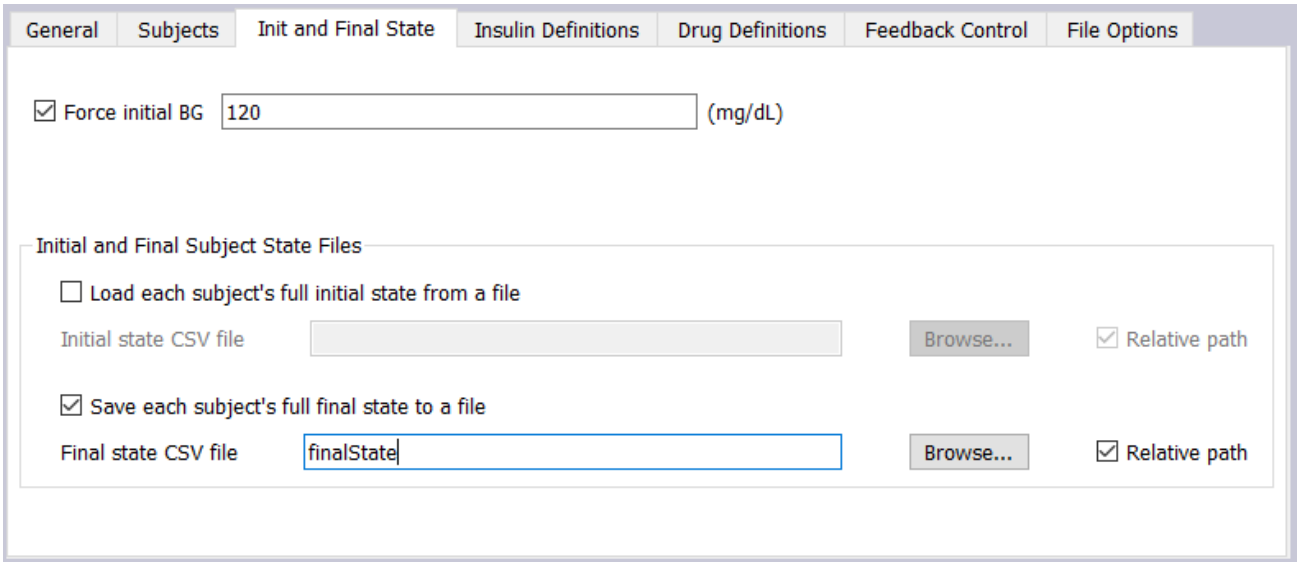


Figure 27: Configuring the Initial States of Subjects

6.4.3.3 Limitations on Loading a Subject's Initial State

Files used to initialize a subject's state have some limitations related to their representation of slow-release insulins. These files can only represent the time of the most recent slow-release bolus and the amount of insulin yet to be released from the slow-release compartment (see section 3.3.3). The rate at which such an insulin is released depends on the entire history of boluses. For example, the portion of the "unreleased" insulin that came from a bolus a couple of hours ago may be released more rapidly to the plasma than the portion that remains from a much older bolus. When initializing from a saved state, the DMMS.R will assume that all of the unreleased insulin came from the most recent bolus.

Consider a final state saved from a simulation with multiple slow-release boluses still on board at the end. If you use this final state to initialize a new simulation, insulin may not be released to the plasma at a rate identical to what would have been seen in the first simulation had it been allowed to continue. This situation is minimized, however, by the fact that most of the unreleased insulin at the start of the 2nd simulation will be associated with the most recent bolus.

A similar situation exists for glucose tablets delivered near the end of a simulation. The final state saved from such a simulation only includes information about the most recently delivered glucose tablet. This limitation should have a minimal effect since glucose from tablets appears in the plasma so rapidly.

6.4.4 Defining Insulins

The Insulin Definitions tab allows you to define any insulin analogs you plan to use in your simulation.

Because these insulins are often referenced from configured feedback control elements, you'll typically want to define them prior to defining feedback controls. For more detail on insulin analogs, insulin/release pairs, etc., see Section 3.3 and the relevant entries in the Glossary (Section 14).

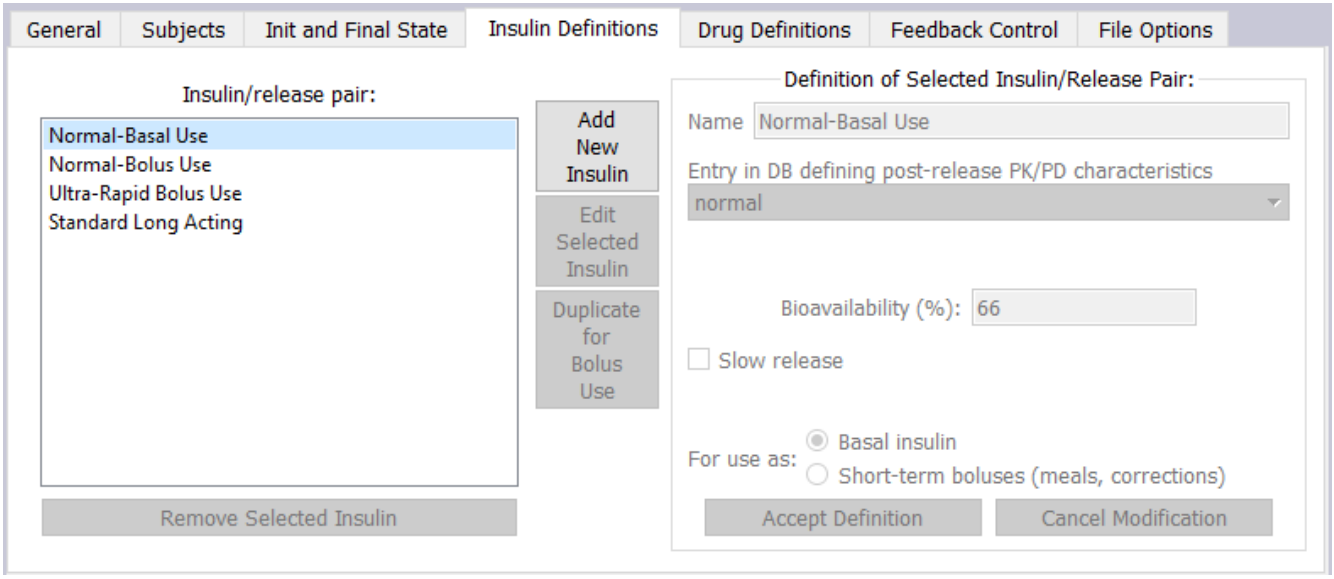


Figure 28: Insulin Definitions Tab

The Insulin Definitions tab allows you to define a new insulin/release pair by selecting the desired post-release PK/PD characteristics from a drop-down (currently "normal" is the only choice), and then selecting the bioavailability and, if applicable, the release profile with which it is to be combined.

When the *Slow-release* checkbox is selected, one of the pre-defined release profiles from the DMMS.R database may be incorporated into the insulin/release pair. When this is left unchecked, the "release" component of the insulin/release pair is instantaneous. In this case, for example, if the insulin is delivered as a bolus to the subcutaneous space, the entire bolus becomes present in that space as soon as it is delivered.

Each insulin/release pair can be used for either bolus (meal or correction) or basal purposes. This allows independent tracking of the amount of each insulin used for each role, within each compartment, as the simulation progresses. In addition, the assigned "use" of any insulin/release pair will limit which control elements can be associated with it. The "Normal-Basal Use" and "Standard Long Acting" insulins are used as basal insulins; "Normal-Bolus Use" and "Ultra-Rapid Bolus" insulins are used, as you might suspect, for non-basal boluses. If you add a new insulin, you may define it to be for either a basal-use or bolus-use.

Parameter	Possible values	Input to Model?
Name	any character string	no
Use	Basal	no
	Bolus	no
Insulin type	Slow release	yes
	Fast release	yes
Post-release PK/PD characteristics	Any available database entry	yes

Table 4: Insulin Definition Parameters

Note that the "Name" and "Use" parameters have no effect on the operation of the model; they are used only to identify the data in the output.

For more on insulin analogs, insulin/release pairs, and related topics, see Section 3.3 and the relevant terms in the Glossary (Section 14).

6.4.4.1 Adding a New Insulin

To add a new insulin analog, press the *Add New Insulin* button. The screen should look like this:

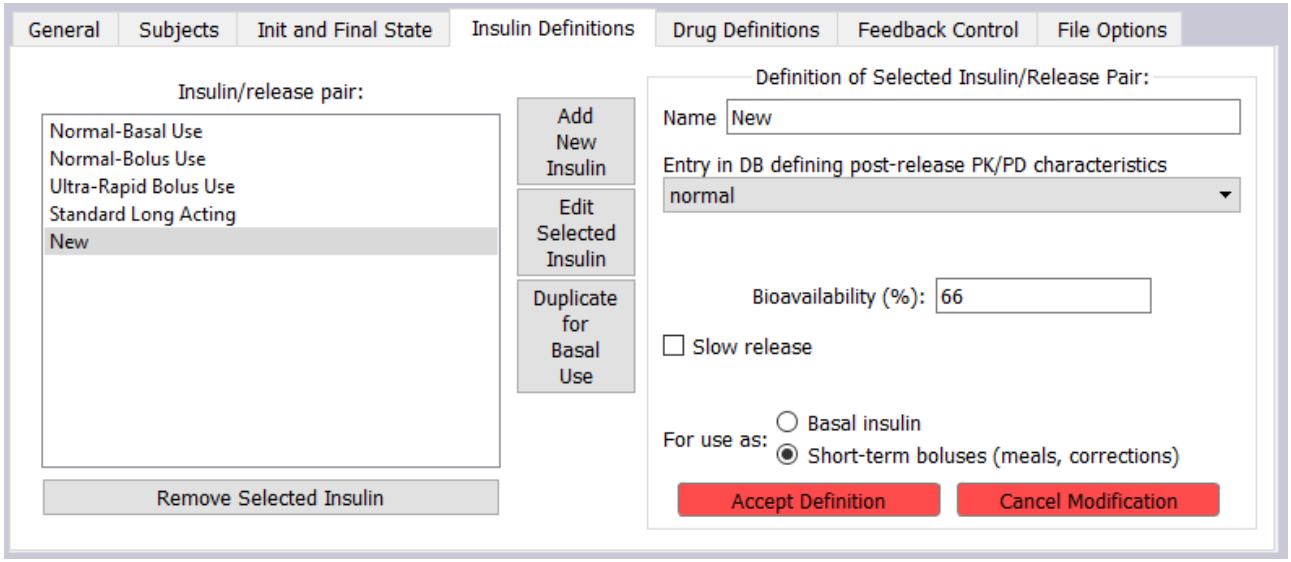


Figure 29: Adding a New Insulin

You can define your new insulin/release pair definitions by making the desired entries on the right side of the screen. Press *Accept Definition* to enter this definition into the program database.

6.4.4.2 Editing a Previously Added Insulin

You may also edit any previously defined insulin/release pair (except for the default ones) by selecting its name in the list and pressing *Edit Selected Insulin*.

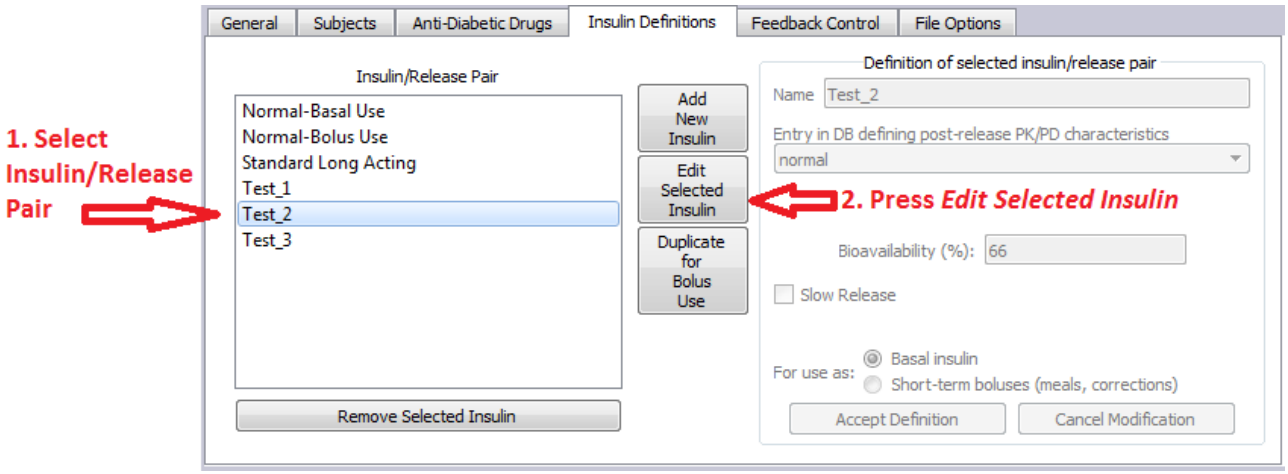


Figure 30: Selecting an Insulin to Edit

Then make the required changes on the right-hand side of the dialog. Editing is enabled when you add a new insulin. Here we have added three new insulins. After adding insulin Test_3, we may want to go back and make changes to insulin Test_2. However, the edit controls for Test_2 are now disabled. To enable them, press *Edit Selected Insulin*.

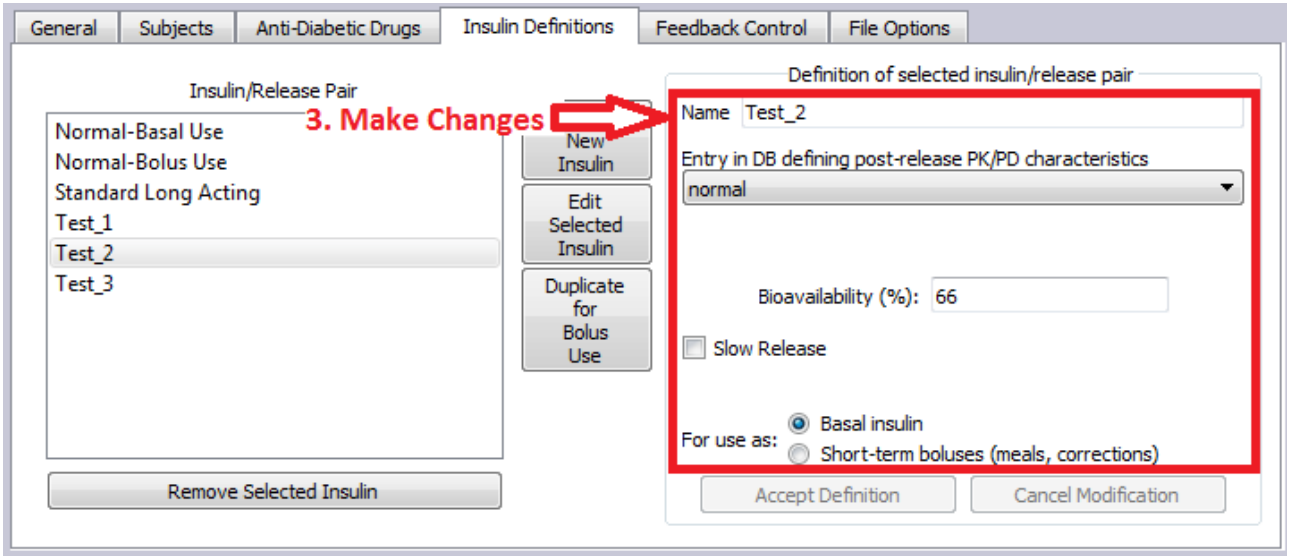


Figure 31: Editing a Previously Added Insulin

6.4.4.3 Duplicating an Insulin for Alternative Use

You may duplicate any defined insulin such that the result is the same in all respects except for the assigned use. To duplicate a basal-use insulin, select it and press *Duplicate for Bolus Use.* To duplicate a bolus-use insulin, select it and press *Duplicate for Basal Use.*

6.4.4.4 Removing a Previously Added Insulin

You may remove a previously added insulin/release pair by selecting it in the list and pressing R*emove Selected Insulin.*

6.4.5 Drug Definition

The Drug Definition tab allows you to define the drug that will be used in the feedback control. See Section 6.4.6.2.5.9 for Configuring drug dosing.

Currently, Metformin is the only anti-diabetic drug available for treatment in the simulator. The post-release PK/PD for this drug was modeled to reflect a steady-state effect; the drug's effect, as modeled, is consistent over time, based on the scheduled daily dosage. *The Epsilon Group is prepared to model other drugs at clients' request by using the plugin, where the affected areas are defined by configuring an instance of the plugin drug. Please reference how to configure plugin elements in Section 6.4.6.*

To add the available drug model, select the drug in the *Available drug models* section then click the *Add ->* button, the selected drug will be listed under *Included drug model instances* section. By selecting each drug model, you can also see the influence area of that drug under the *Selected drug's areas of influence*.

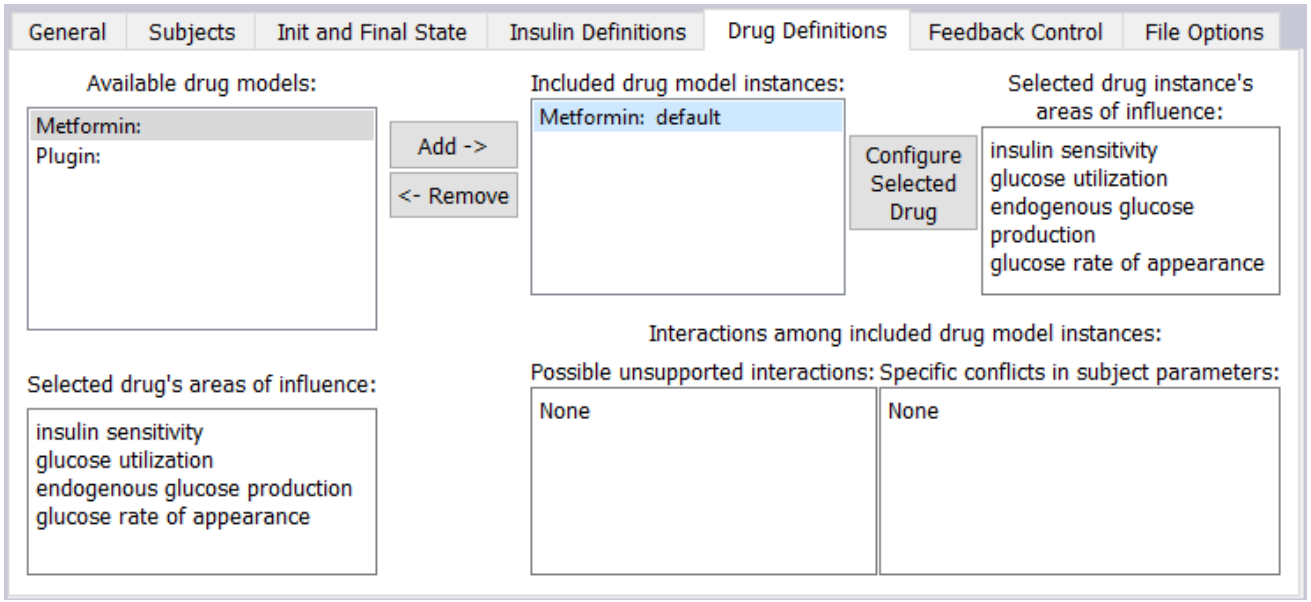


Figure 32: Add available drug model

6.4.6 Configuring Feedback Control Elements

The *Feedback Control* tab permits definition of sensor, control, and delivery devices. Each of these is explained in the following subsections.

6.4.6.1 Sensors

The sensors subtab is shown in Figure 33 below. Because the output signals from sensors are often referenced by other control elements, you will typically want to get these configured first.

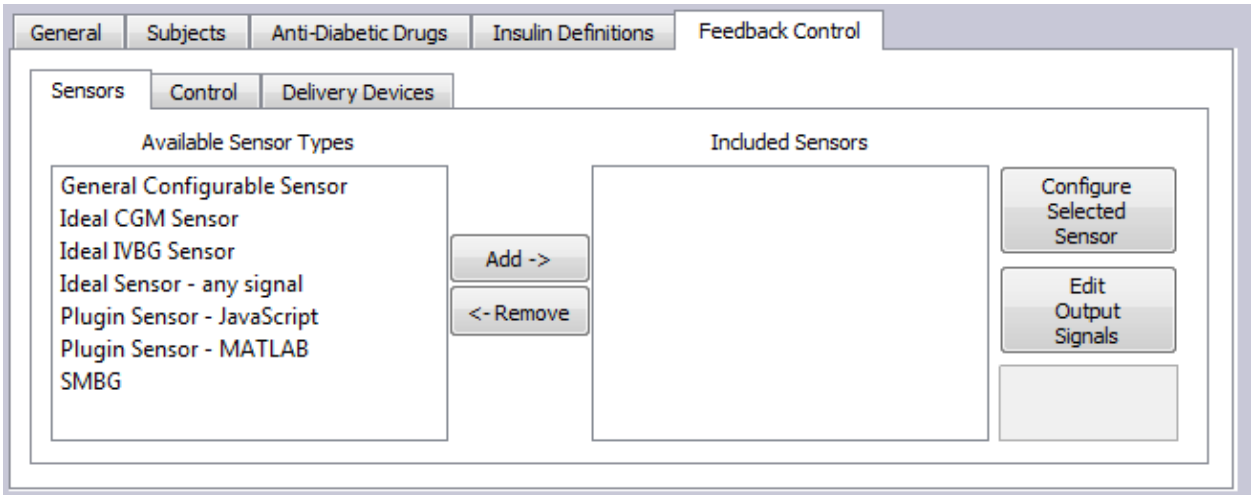


Figure 33: Sensors Subtab

6.4.6.1.1 Adding a New Sensor

You may include any available sensors in your simulation. Highlight the desired sensor and press *Add.*

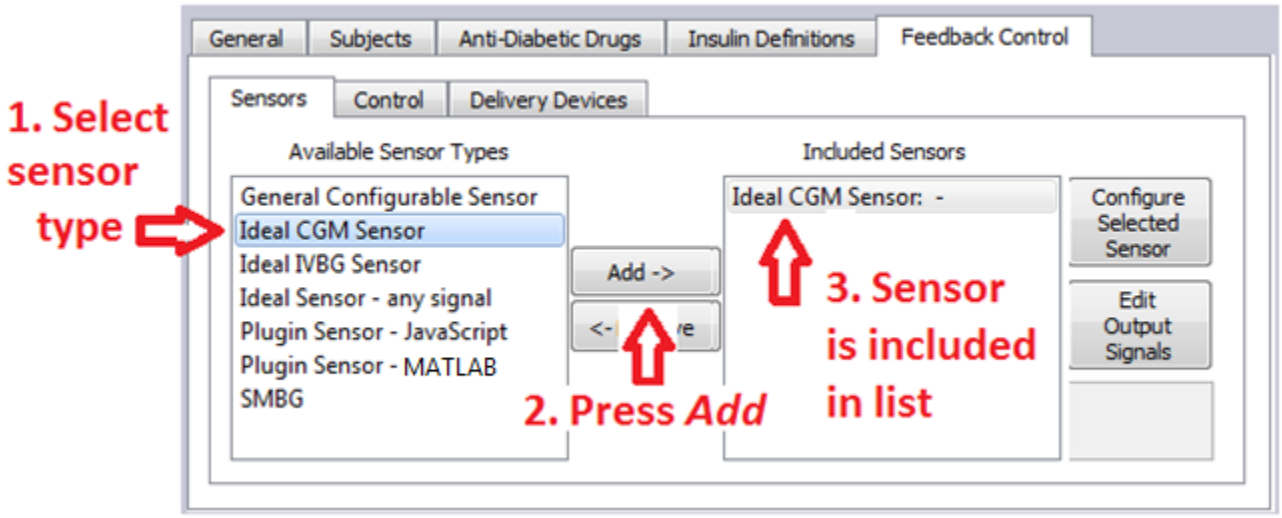


Figure 34: Adding a New Sensor

You may add additional sensors of most sensor types. These additional sensors may be configured differently. The application does not allow duplicate Ideal CGM Sensors or Ideal IVBG Sensor types as they cannot be reconfigured; there is no point in having duplicate identical sensors.

Notice that the identifier of the included sensor has two parts. The first part is its type (in this case "SMBG"). The second part is its name, initially just a hyphen ("-"). If you were to add additional sensors of the same type, a number is automatically added to the name so that it remains unique (see **Error! Reference source not found.**).

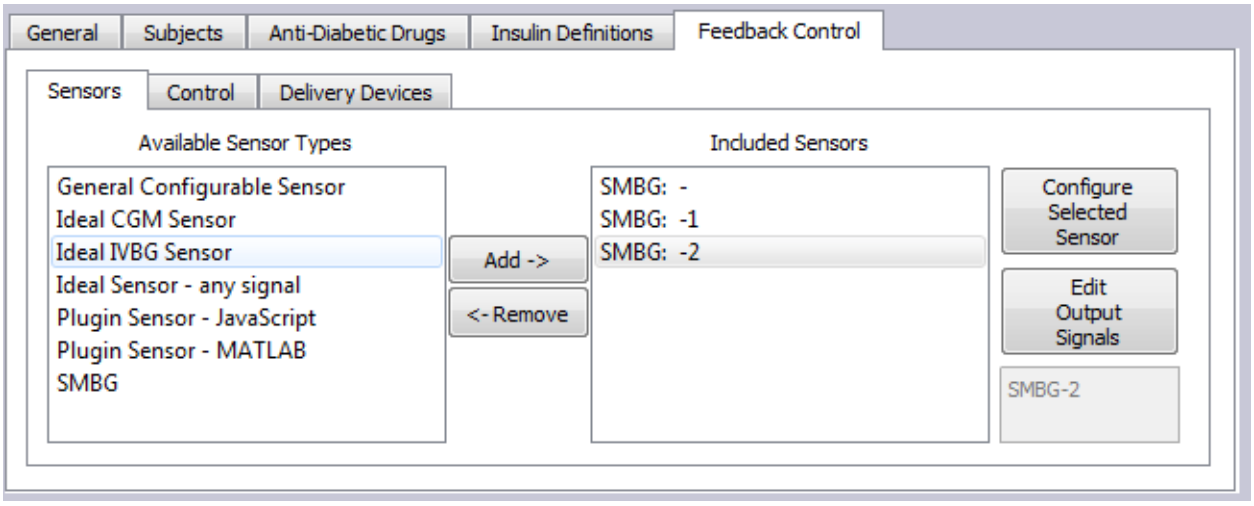


Figure 35: Multiple Sensors of the Same Type

The hyphen-and-number syntax is intended as a temporary place holder; you will probably want to change the sensor name to something more descriptive. (See Section 6.4.6.1.3)

The "ideal sensors" (Ideal BG Sensor, Ideal CGM Sensor, and Ideal Generic Sensor) are preconfigured to deliver signals without any noise or latency (lag). If this is not the behavior you want, select the General Configurable Sensor, and configure it to meet your requirements, or select a suitable Plugin Sensor.

6.4.6.1.2 Removing a Sensor

To remove an included sensor, simply select the sensor to be removed and press the Remove button.

6.4.6.1.3 Changing the Name or List Position of a Sensor

You may change the position of a sensor, in the list of included sensors, by selecting it and choosing *Move Up or Move Down* as desired.

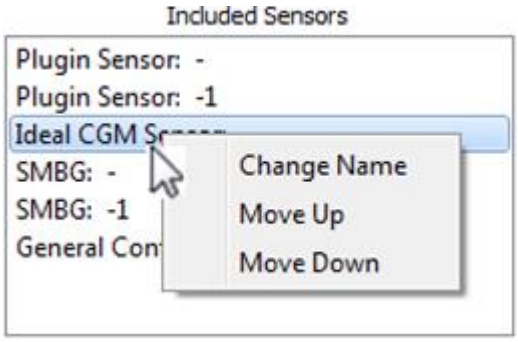


Figure 36: Changing the Name of a Sensor

Changing the position of the sensor in the list changes the order of execution within each iteration of the model: sensors are activated in list order. This feature allows input/output signal dependencies among sensors and can be especially useful with plugin sensors.

Choosing *Change Name* will produce the following pop-up dialog box:

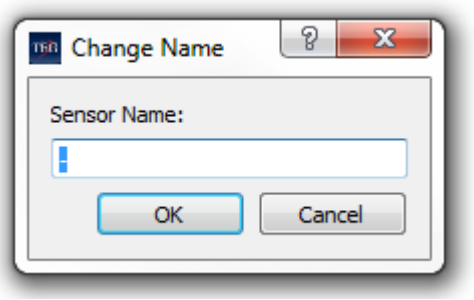


Figure 37: Entering the New Sensor Name

Note that the sensor type is not part of the name and cannot be changed. Also, the program will not allow you to duplicate the name of another sensor of the same type. For example, if in the example above, we try to change the name of SMBG sensor -2 to -1, we get this warning:

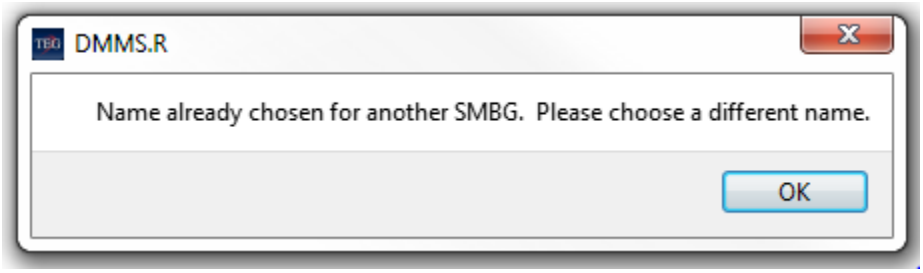


Figure 38: Duplicate Sensor Name Warning

This feature prevents having 2 or more sensors with the same identifier.

6.4.6.1.4 Changing the Name of a Sensor's Output Signal

In addition to changing the name of a sensor, you may change the name of the signal the sensor produces.

To do this, select the sensor from the *Included Sensors* list and press *Edit Output Signal*. Using our example, we will choose SMBG Sensor -2. The following dialog should appear:

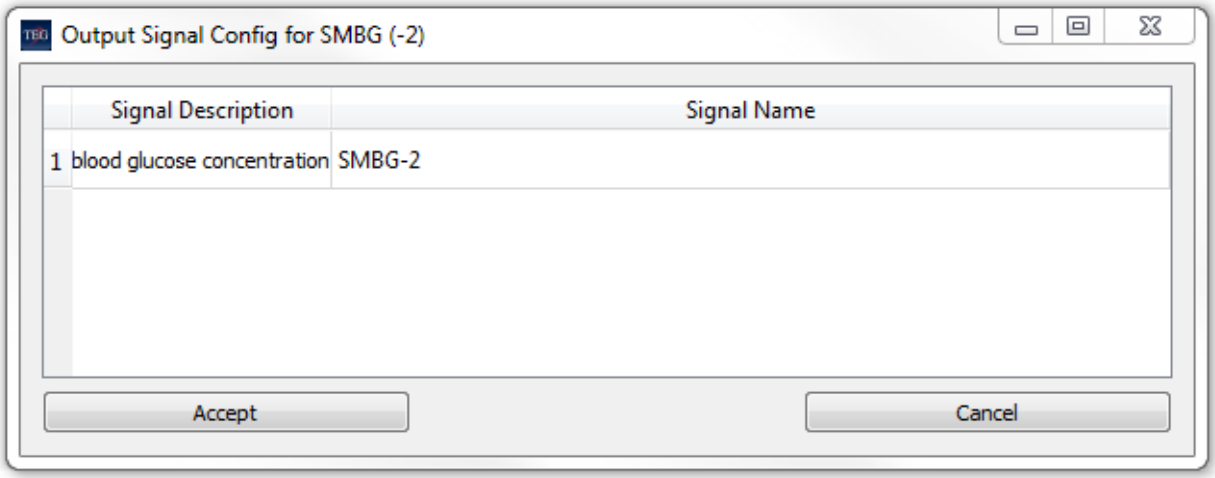


Figure 39: Sensor Signal Name Dialog

Note that the default signal name is qualified with the name of the sensor (in this case "-2". This is to ensure it will be unique with respect to the signals of any other included sensor (SMBG -1, for example).

Double-click the current signal name to open the box for editing:

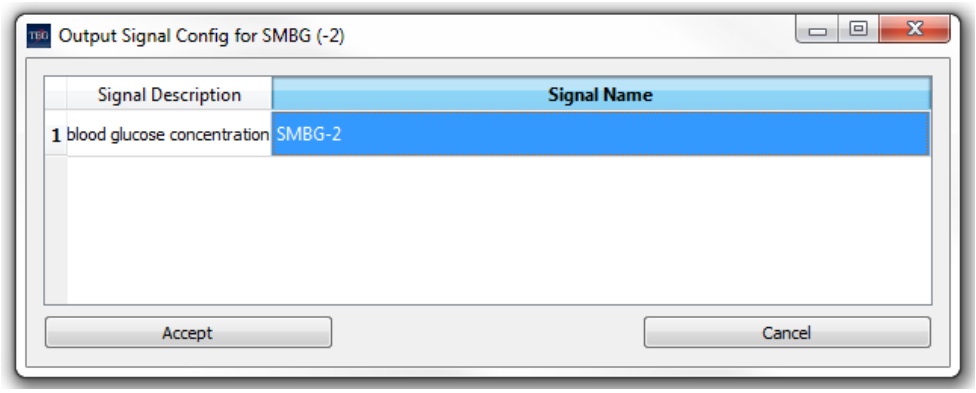


Figure 40: Changing the Signal Name

As with the sensor name, the application will not allow you to duplicate an existing signal name.

6.4.6.1.5 Configuring a Sensor

Configuring a sensor of any of the configurable sensor types begins with the steps outlined in this section. If your list of *Included Sensors* does not contain a sensor of the desired type, you must first add it to the list. Then select the sensor itself and press *Configure Selected Sensor*. When you do this, a dialog box will appear.

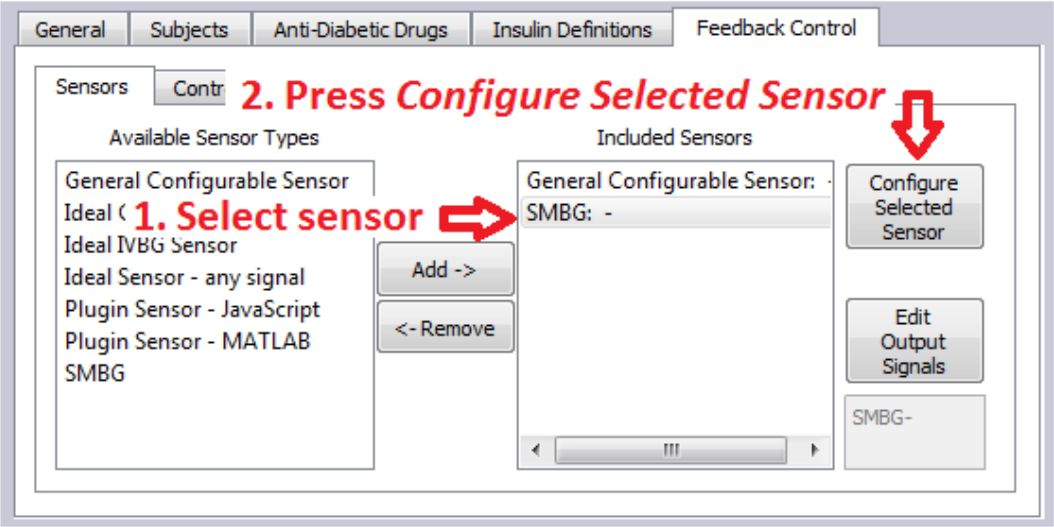


Figure 41: Configuring a Sensor

The configuration options for each of the available sensor types vary. Some sensors have many configuration parameters while others have none.

6.4.6.1.5.1 *Configuring an Ideal CGM Sensor*

The Ideal CGM Sensor is defined to always provide the subcutaneous (interstitial) blood glucose concentration from the subject's current state, with no noise, lag, or any other error.

The Ideal CGM Sensor has no attributes that can be configured.

6.4.6.1.5.2 *Configuring the Ideal BG Sensor*

The Ideal BG Sensor provides the plasma glucose concentration based on the subject's current state, with no noise, lag, or any other error.

The Ideal BG Sensor has no attributes that can be configured.

6.4.6.1.5.3 *Configuring the General Configurable Sensor*

The General Configurable Sensor, as you might infer from the name, is highly configurable. Selecting this sensor, and pressing *Edit Selected Sensor* will produce this dialog:

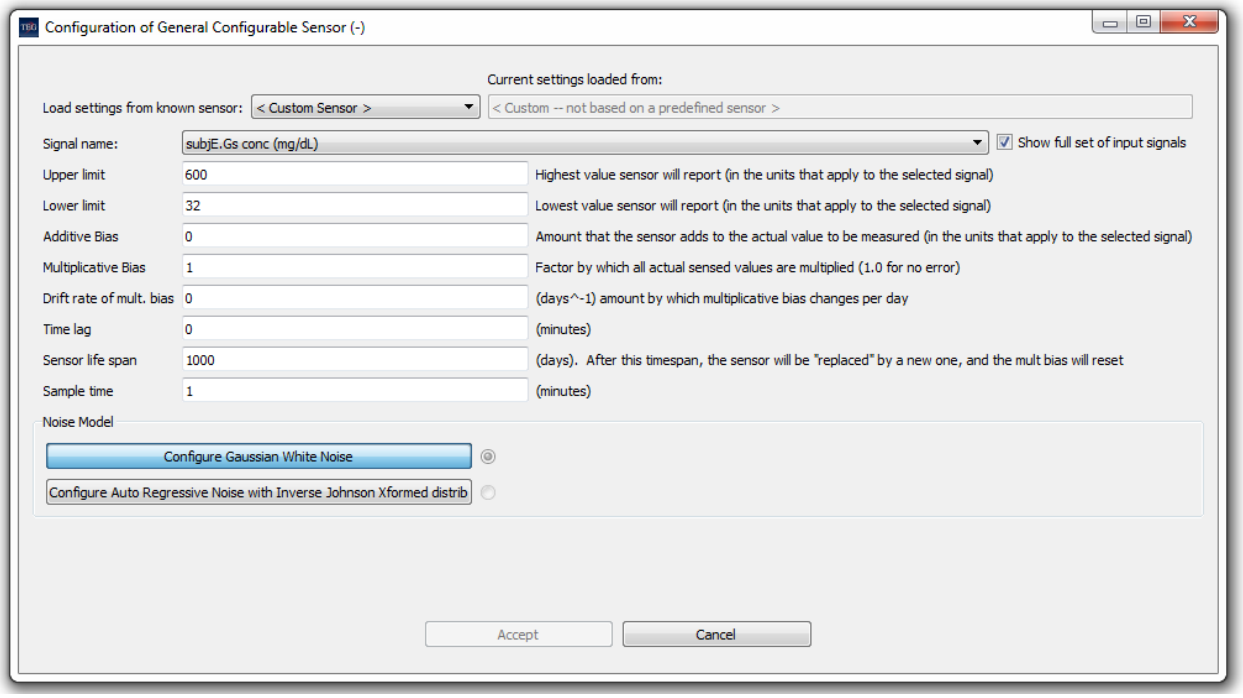


Figure 42: Configuring the General Configurable Sensor

You may edit this sensor to specify:

1. A pre-defined sensor from which to load configuration values
2. The name of the signal this sensor will supply[5]
3. The range of output reported
4. A bias to add to the value reported by the signal
5. A bias with which to multiply the value reported by the sensor
6. A drift rate
7. A time lag
8. The number of days the sensor is to be active
9. The sample time in minutes
10. Gaussian white noise be applied to this sensor
11. Auto regressive noise be applied to this sensor

If you choose to apply Gaussian white noise to the filter, you will be able to specify:

1. A random seed.
2. Noise sample time
3. Standard deviation

[5] Definitions for these signals are given in Appendix A: Output Data Definition. Note that when *Show full set of input signals* is *not* checked, the list shows more "friendly" (i.e., less cryptic) names for these signals. In this situation, tooltips are presented that relate friendly and cryptic names.

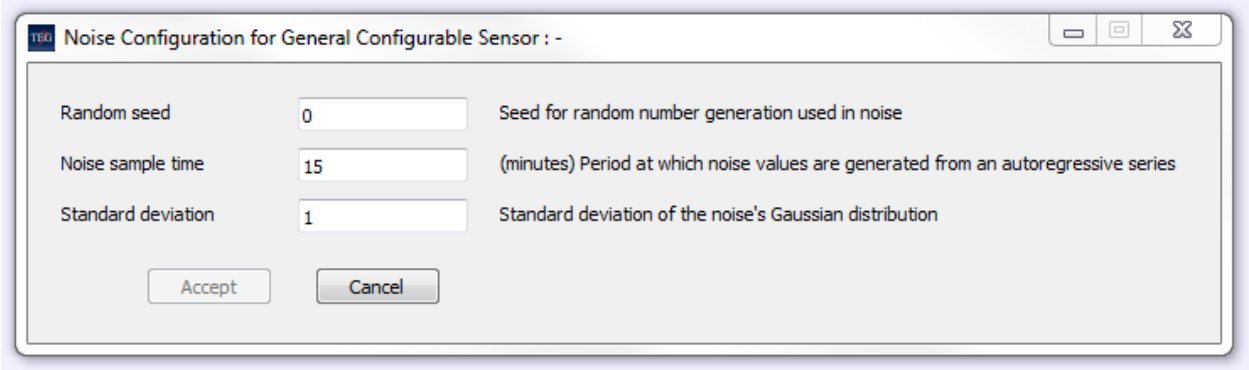


Figure 43: Configuring Noise for the Standard Sensor

For more detail on Gaussian white noise, see Section 3.7.

Choosing to configure Auto Regressive noise will allow entry of additional configuration:

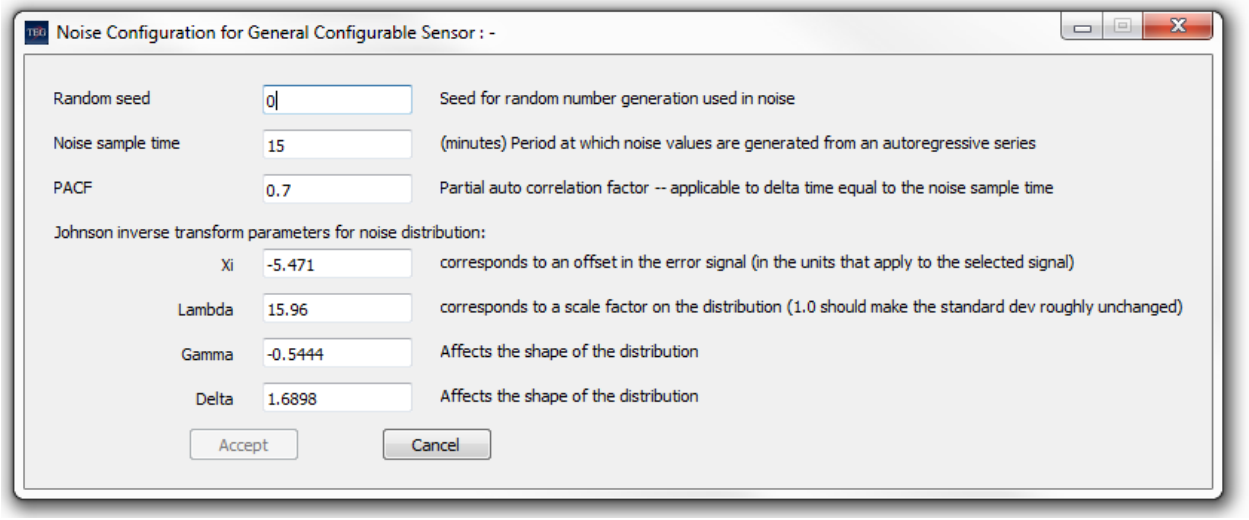


Figure 44: Configuring Noise with Auto Regression

1. Random seed
2. Noise sample time
3. Partial auto-correlation factor (PACF)
4. Error signal offset (Xi)
5. A scale factor (Lambda)
6. Distribution shape factor (Gamma)
7. Distribution shape factor (Delta)

6.4.6.1.5.4 *Configuring the Ideal Sensor – any signal*

The Ideal Sensor is defined as a sensor that always provides the configured subject state value from the subject's current state, with no noise, lag, or any other error.

Configuring the Ideal Sensor consists of selecting the subject state.

Activating the Signal Name drop down will show a list of some subject state variables from which you can choose (see footnote 5, page 49). These signals will be prefixed with subj, subjE, and insulin.

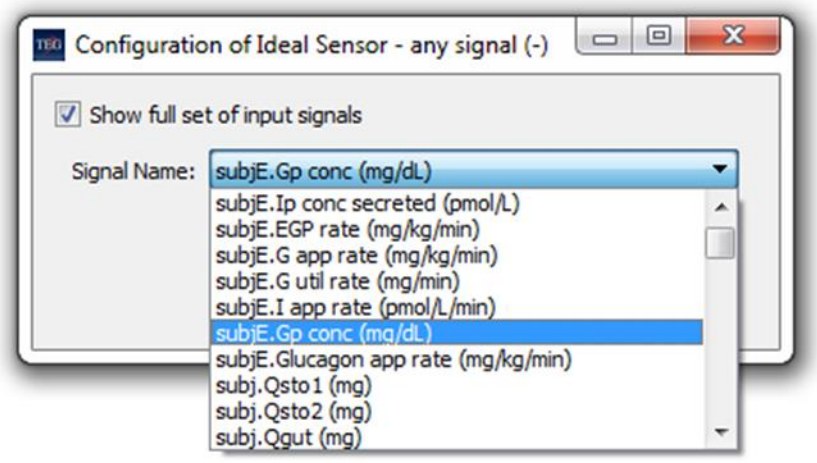


Figure 45: Expanded List of Signal Name List for the Ideal Sensor

To reduce the size of the list of available signals to include only those most commonly used, uncheck the *Show full set of states* checkbox[5].

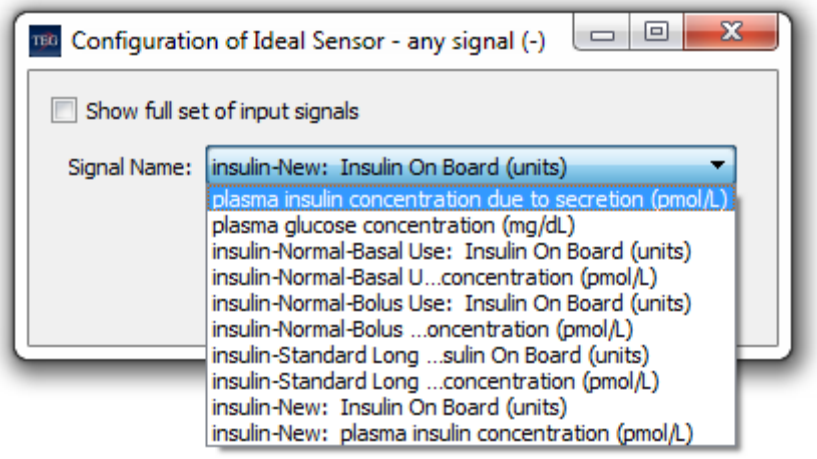


Figure 46: Reduced Signal Name List for the Ideal Sensor

6.4.6.1.5.5 *Configuring a Plugin Sensor*

On the Windows version of DMMS.R, you may configure a plugin sensor written in either JavaScript or MATLAB. Example files, "samplePluginSensor.js" and "samplePluginSensor_matlab.m", are provided in the configuration files directory (see Section 5.4).

In addition to specifying the path to the JavaScript or MATLAB file you want to use, you must also indicate the subject state (or states) that the sensor will monitor, and the path to the JavaScript or MATLAB file the sensor will use. Like the General Configurable Sensor, you may choose from a full list of subject states or from a shortened list, depending on the state of the *Show full set of input signals* checkbox. Unlike the General Configureable Sensor, you are not limited to a single subject state variable: you may add as many as you need.

The location of the JavaScript or MATLAB plugin file may be specified by either an absolute path or a path that is relative to your default configuration directory (see section 6.3.1.1). This is controlled by a pair of radio buttons. In either case, the file path/name may be entered directly into the applicable edit box, or selected by pressing the Browse… button. Using relative paths will allow your configuration files to be shared more easily with users who may want to perform simulations in a different directory location on their own computers. Note that the DMMS.R will always display both the relative and absolute path of the plugin file, regardless of which choice is made via the radio buttons.

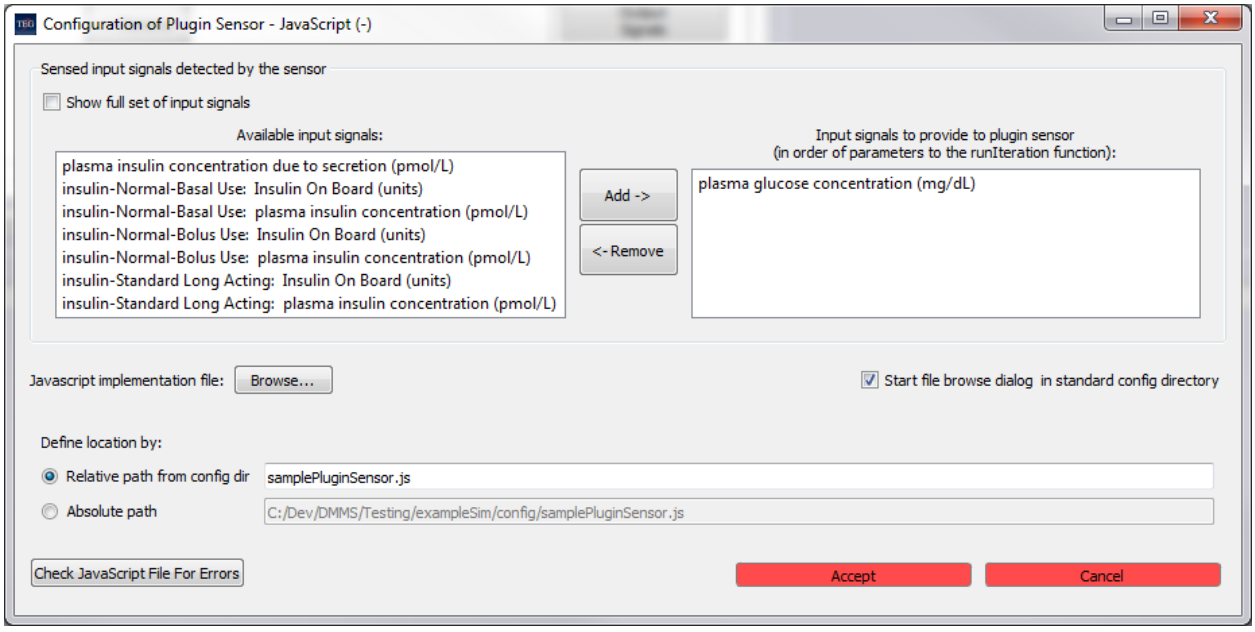


Figure 47: Configuring a JavaScript Plugin Sensor

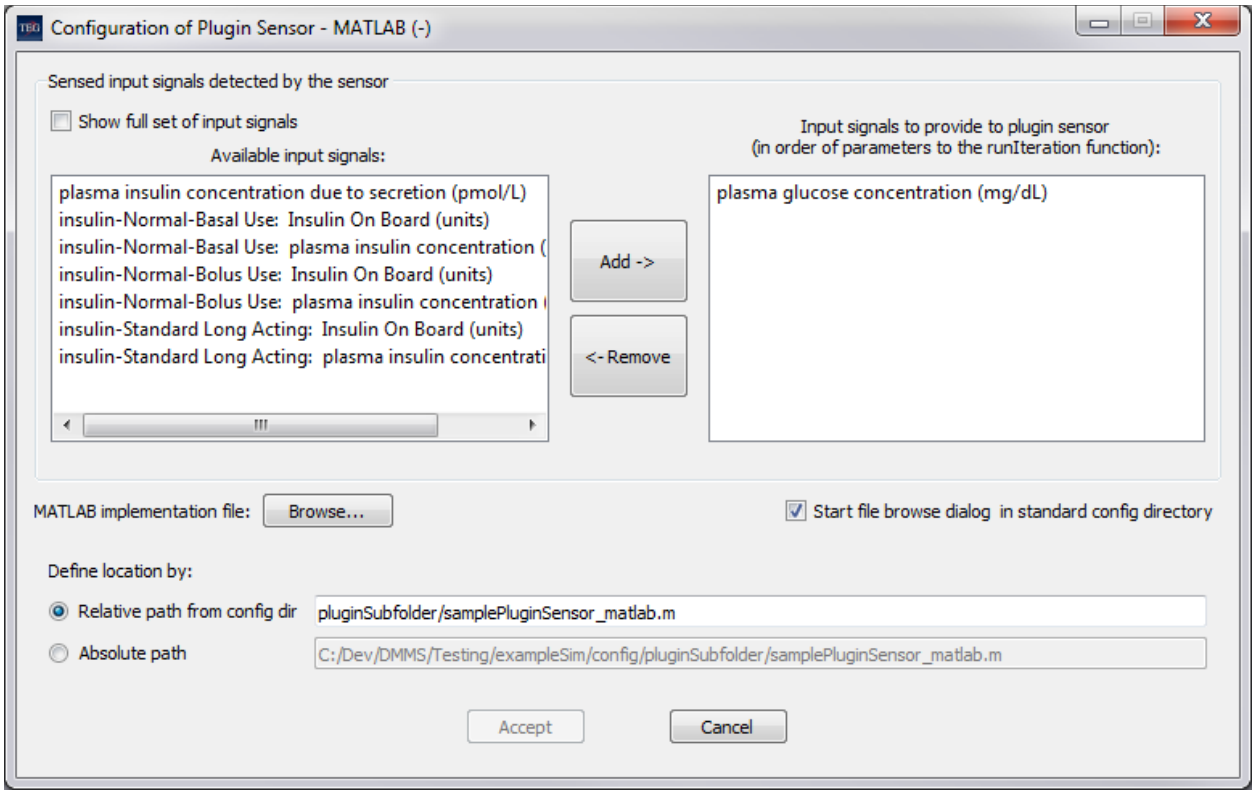
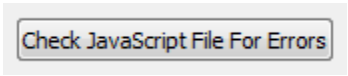


Figure 48: Configuring a Plugin Sensor with MATLAB

As you can see from Figure 47 and Figure 48, the JavaScript and MATLAB configuration dialogs are very similar. However, DMMS.R does provide a means to check the syntax of your JavaScript file:



This can be handy in detecting and identifying errors in the script before running the simulation.

6.4.6.1.5.6 *Configuring the SMBG Sensor*

Configuration of the SMBG Sensor is supported by the following dialog:

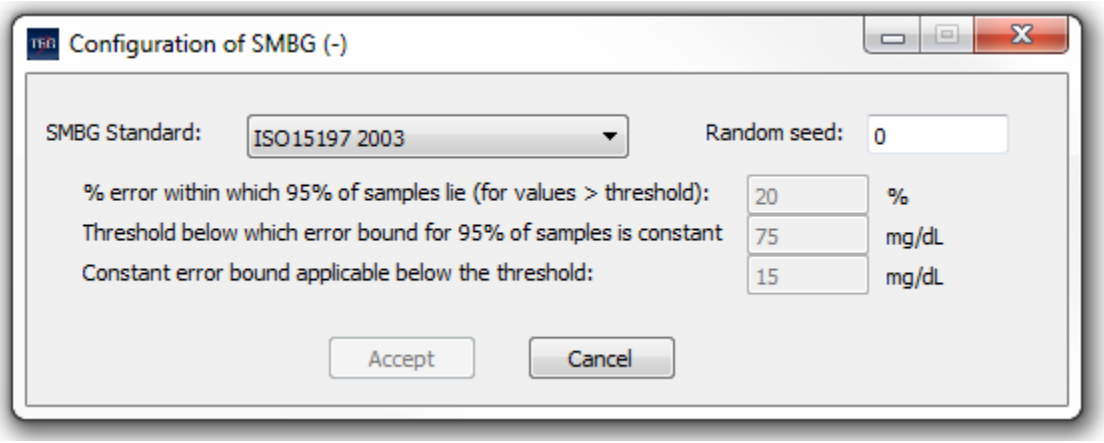


Figure 49: Configuring the SMBG Sensor

The SMBG Sensor provides plasma blood glucose concentration data from the subject's current state, with Gaussian white noise applied to every sample. The magnitude of the noise is defined as the maximum permitted by one of the ISO specifications for SMBG (the ISO15197 2003 or ISO15197 2013 specification).

You may select which specification to apply from within the SMBG configuration dialog.

Noise characteristics for the 2003 SMBG sensor are defined as follows:

- For blood glucose concentrations above 75 mg/dL, the error magnitude within which 95% of all sensor samples lie is 20% of the actual blood glucose concentration (e.g., for true blood glucose concentrations of 200mg/dL, 95% of all sensor samples would be between 160 and 240mg/Dl.).
- For blood glucose concentrations at or below 75 mg/dL, the error magnitude within which 95% of all sensor samples lie is 15 mg/dL (e.g., for true blood glucose concentrations of 60mg/dL, 95% of all sensor samples would be between 45 and 75mg/dL).

Noise characteristics for the 2013 SMBG sensor are defined as follows:

- For blood glucose concentrations above 100 mg/dL, the error magnitude within which 95% of all sensor samples lie is 15% of the actual blood glucose concentration (e.g., for true blood glucose concentrations of 200mg/dL, 95% of all sensor samples would be between 170 and 230mg/dL).
- For blood glucose concentrations at or below 100 mg/dL, the error magnitude within which 95% of all sensor samples lie is 15 mg/dL (e.g., for true blood glucose concentrations of 80mg/dL, 95% of all sensor samples would be between 65 and 95mg/dL).

6.4.6.2 Control Elements

The *Control* subtab permits you to define the control elements to be used in your simulation.

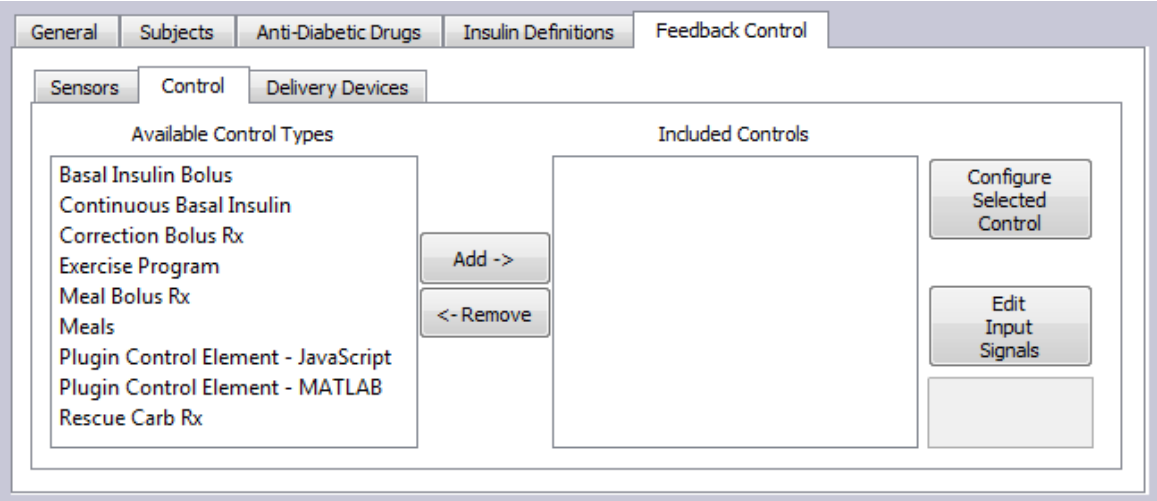


Figure 50: The Control Subtab

6.4.6.2.1 Adding a New Control Element

The process for adding a new control element is very similar to that for adding a new Sensor. (See Section 6.4.6.1.1)

6.4.6.2.2 Removing a Control Element

The process for removing a control element is very similar to that for removing a Sensor. (See Section 6.4.6.1.2)

6.4.6.2.3 Changing the Name or List Position of a Control Element

The process for changing the name or list position of a Control is very similar to that for changing the name or list position of a Sensor. (See Section 6.4.6.1.3)

6.4.6.2.4 Selecting a Control Element's Input Signals

To configure input signals for a control, select the control from the *Included Controls* list and press the *Edit Input Signals* button. If the control has selectable input signals, a dialog will appear to permit you to select them. The Correction Bolus Rx, for instance, has signals that can be configured via this dialog:

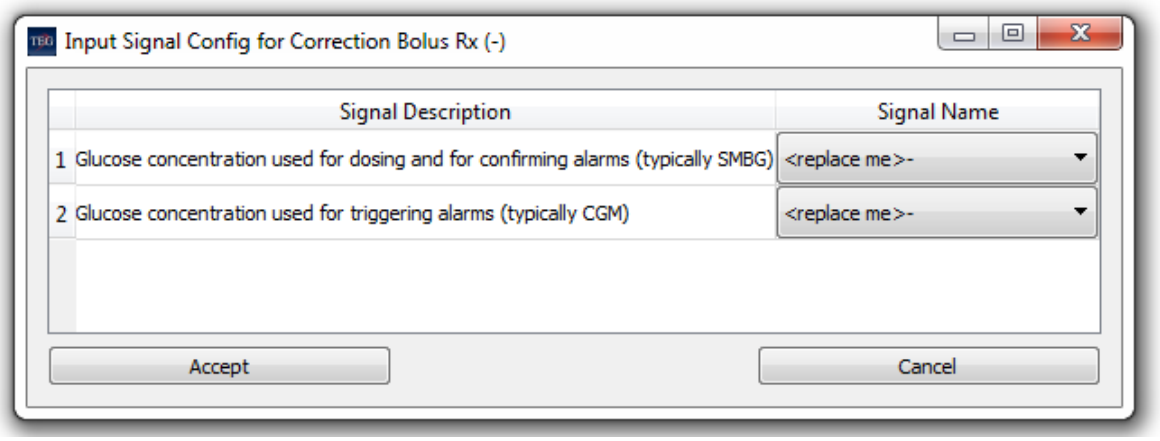


Figure 51: Selecting Control Element Input Signals

The *Signal Descriptions* listed in the dialog are defined by the particular control element you are configuring. The available items in the *Signal Name* dropdowns result from the sensor configurations you have defined. The phrase "<replace me>" in the dialog shown is a placeholder to remind you to enter an actual Signal Name.

The input signals required for each control element are listed in Table 5:

Control Element	Input Signals
Basal Insulin Bolus	none
Continuous Basal Insulin	
Correction Bolus Rx	Dosing BG (typically SMBG)
	Triggering BG (typically CGM)
Exercise Program	none
Meal Bolus Rx	
Meals	
JavaScript Plugin	Defined in plugin
MATLAB Plugin (Windows platform only)	
Rescue Carb Rx	Blood Glucose Concentration

Table 5: Input Signals for Each Control Element

6.4.6.2.5 Configuring a Control Element

The process for configuring control elements is very similar to that for sensors (See Section 6.4.6.1.5).

6.4.6.2.5.1 *Insulin Delivery*

Some control elements have configuration dialog boxes with a checkbox for selecting whether the insulin is to be delivered subcutaneously or directly to the plasma. Because the far more typical choice is subcutaneous, the default setting is unchecked. Checking the checkbox will cause the insulin to be delivered directly to the plasma.

Note that direct-to-plasma delivery is generally intended to only be used in conjunction with:

- an appropriate delivery mechanism
- …or with an insulin release profile that simulates the insulin becoming available in the bloodstream in accordance with realistic timing. (See Section 3.3.3)

6.4.6.2.5.2 *Configuring the Basal Insulin Bolus Control Element*

The Basal Insulin Bolus Control Element configuration dialog is shown below:

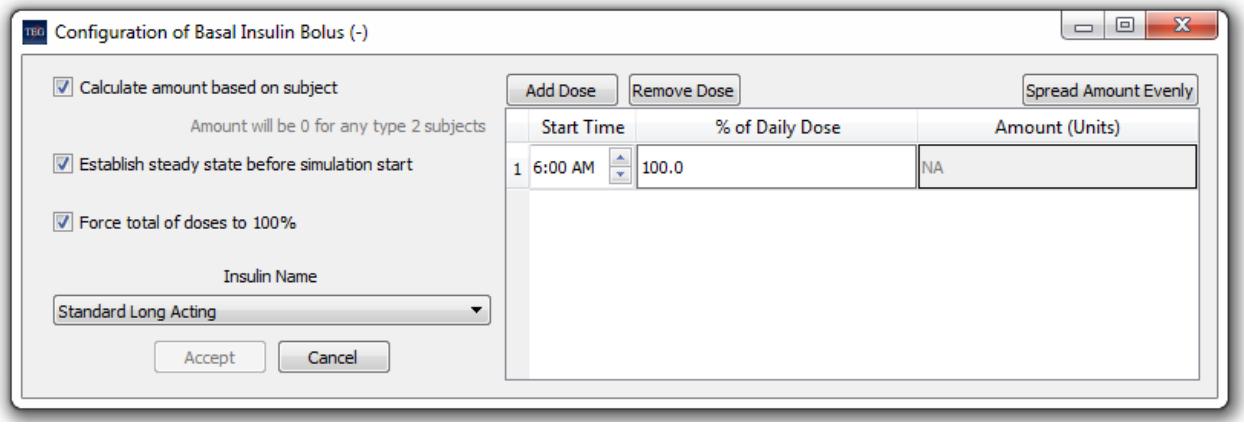


Figure 52: Configuring the Basal Insulin Bolus Provider

The Basal Insulin Bolus Provider, by default, provides a single bolus at 6:00 am. You can, of course, change both the time of the bolus and add new boluses at different times. Here we have added another dose and changed the time at which each dose is administered:

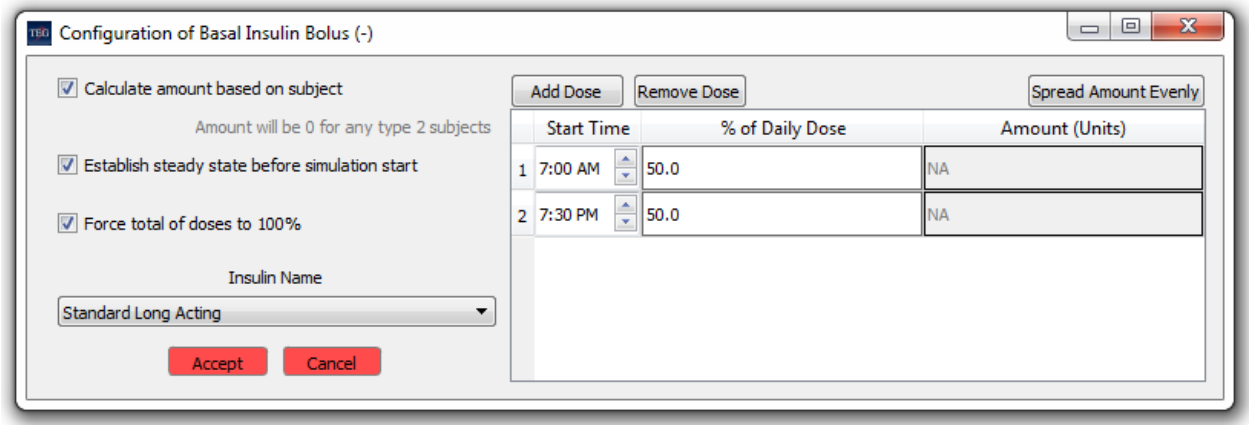


Figure 53: Configuring the Basal Insulin Bolus Provider (2)

Notice that the *% of Daily Dose* for each bolus is split evenly. We can change that as well:

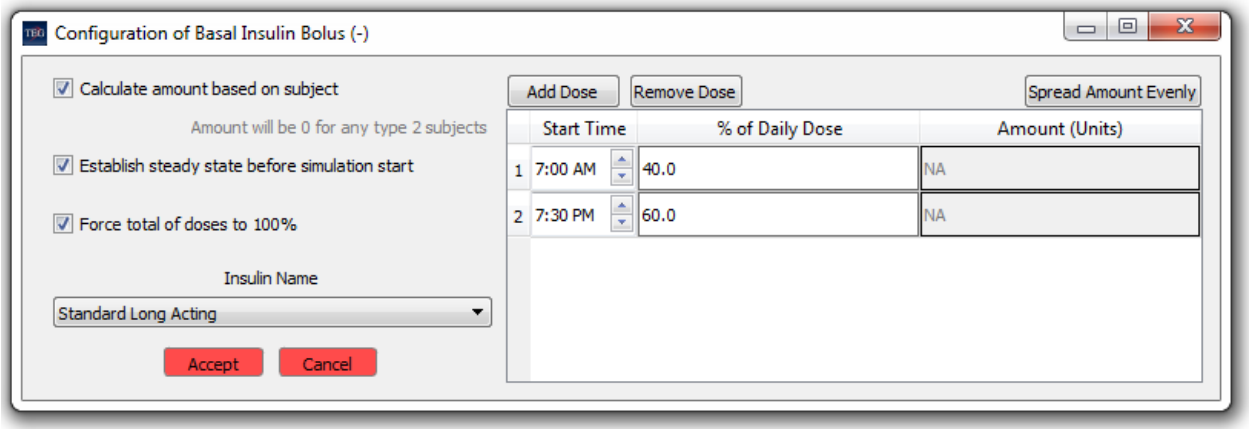


Figure 54: Configuring the Basal Insulin Bolus Provider (3)

Pressing the *Spread Amount Evenly* button will equalize the dose between/among boluses.

If the *Force total of doses to 100%* checkbox is checked, any change in a single dose will force any other boluses to adjust so that their total is equal to the daily amount.

If you do not want the application to calculate the total daily amount, but would rather enter it explicitly, unchecked the *Calculate amount based on subject* checkbox. This will cause an additional edit box is enabled to allow you to enter the desired daily dose:

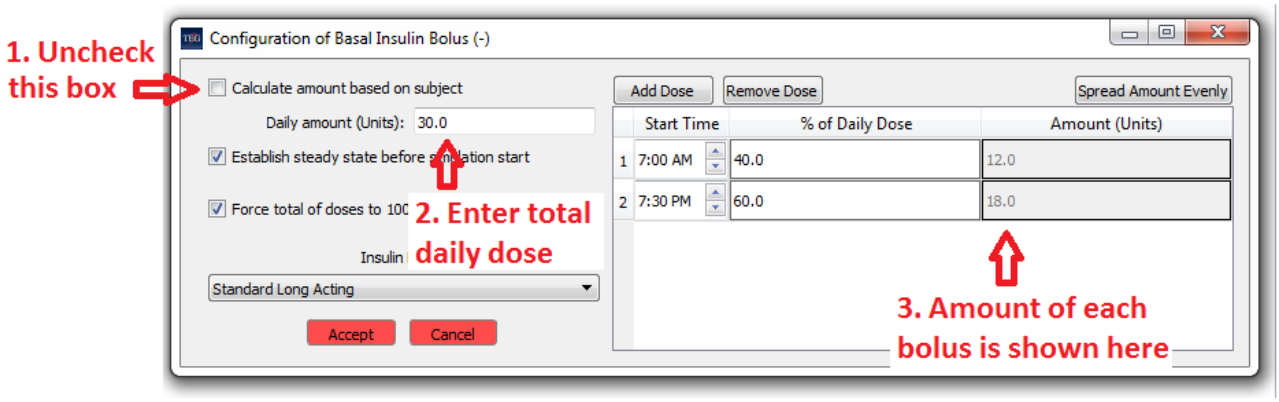


Figure 55: Configuring the Basil Insulin Bolus Provider (4)

When you do this, the amount of each individual bolus is calculated and displayed.

You may also specify the insulin to be used:

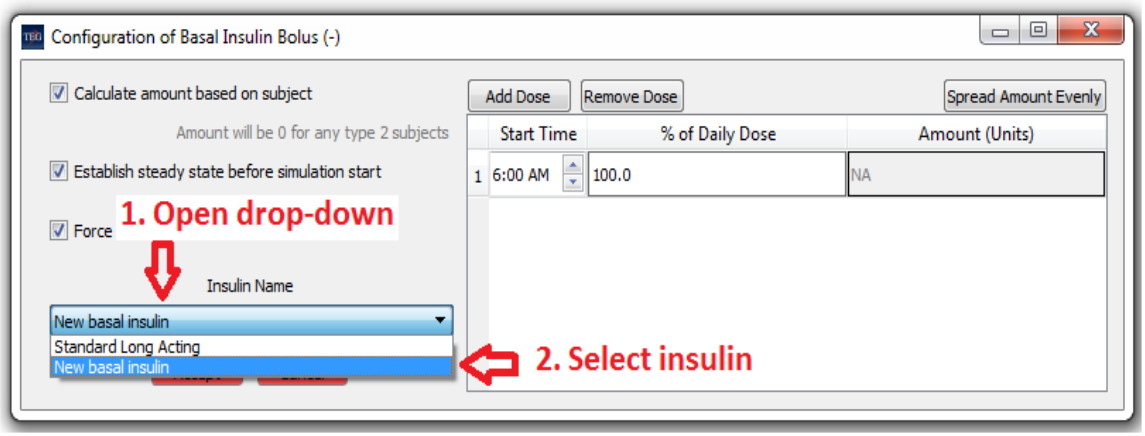


Figure 56: Configuring the Basal Insulin Bolus Provider (5)

When *Establish steady state before simulation start* is selected, the DMMS.R will ensure that, at the beginning of the simulation, the amount of the chosen insulin in each of the subject's compartments (plasma, liver, etc.) will equal the maximum amount that would ever be reached for the associated time of day. Note that the time to reach this "steady state" condition is completely determined by the duration of the release profile for the insulin. To reach steady state, the DMMS.R performs simulations on each subject for a "warmup period" during which the basal insulin is provided according to the defined schedule. During this warmup period, other control elements do not operate, and signal history data is not recorded or graphed. Configurations which specify an initial state or initial BG value for subjects (see section 6.4.3) are not compatible with this functionality and are not permitted by the DMMS.R.

When *Establish steady state before simulation start* is not selected, the amount of the chosen insulin in each compartment will be 0 at the start of the simulation. In this case, the simulation can be used to see how a subject approaches steady state starting with the initial dose.

Note that when *Establish steady state before simulation start* is not selected, a virtual subject's initial "Normal-Basal Use" insulin concentration will be at the level that would hold that subject's blood glucose concentration at their normal fasting level.

6.4.6.2.5.3 *Configuring the Continuous Basal Insulin Control Element*

You may configure the Continuous Basal Insulin Control element by specifying the insulin to deliver and the dosage.

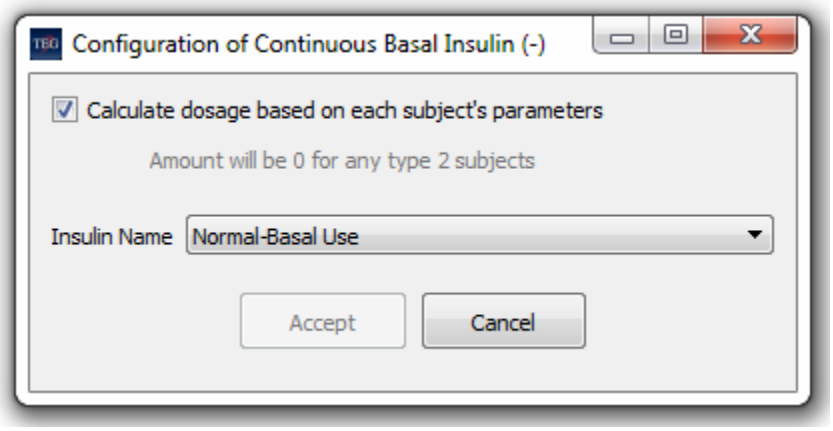


Figure 57: Continuous Basal Insulin Configuration

By default, the dosage is calculated based on the subject's parameters stored in the database. You may enter a specific amount by unchecking the *Calculate dosage*… check box.

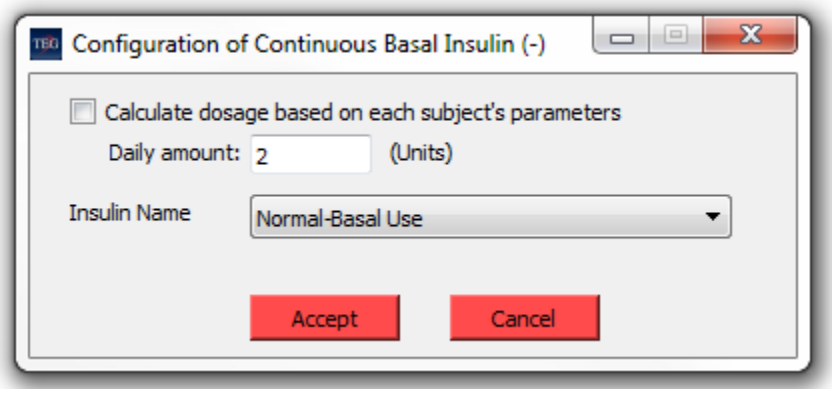


Figure 58: Continuous Basal Insulin Configuration

In the case of Type 2 or Pre-diabetic subjects, the daily amount will be 0. This is because the calculated amount of exogenous insulin is what is required to hold the subject's BG at the basal level. For these subjects, the basal BG level is defined to be that resulting from the subject's naturally secreted insulin.

6.4.6.2.5.4 *Configuring the Correction Bolus Rx Control Element*

The correction bolus configuration dialog allows you to specify several aspects of the correction bolus provider.

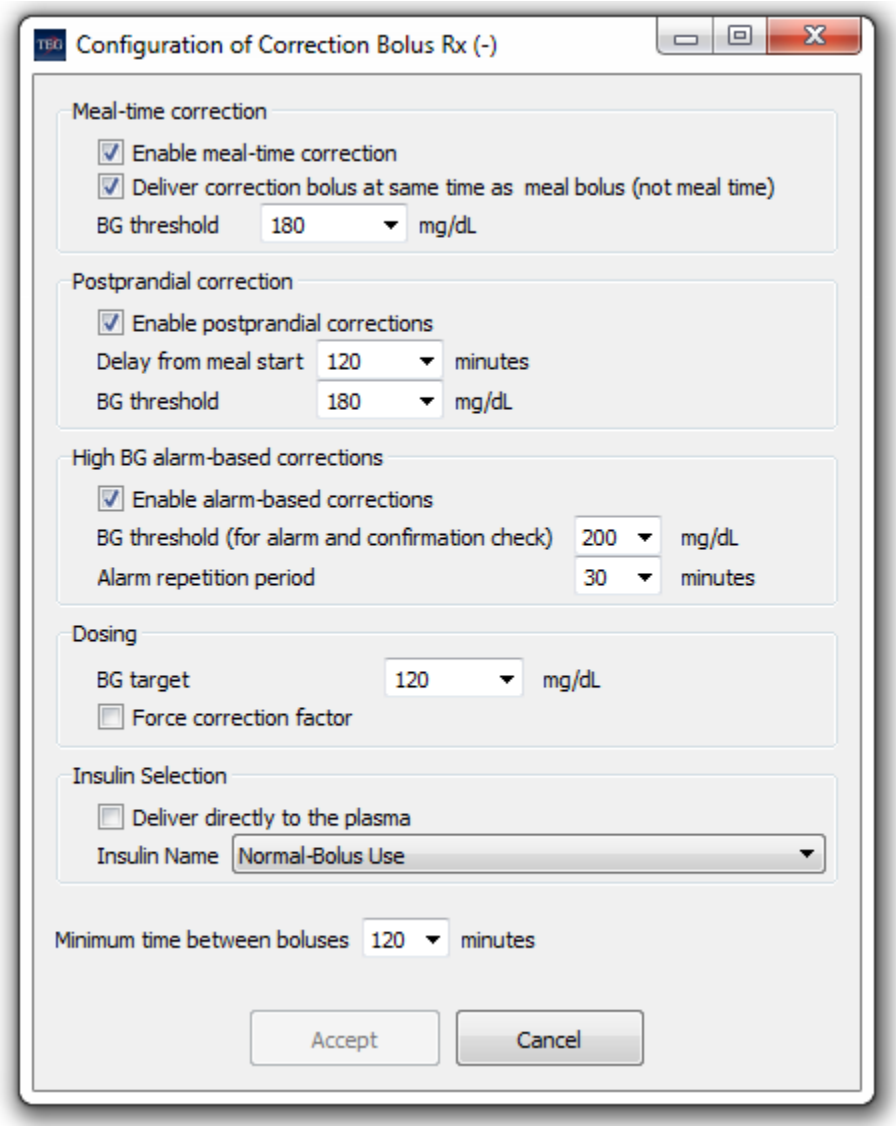


Figure 59: Configuring the Correction Bolus Rx Control Element

The correction bolus provider administers boluses of insulin whenever the subject's blood glucose concentration has exceeded a specified threshold. The dosing of these boluses is based on the amount of glucose that 1 unit of insulin will control.

All three types of correction boluses (mealtime, postprandial, or alarm-based) may be independently enabled or disabled. In all three cases, DMMS.R uses a signal representing a measured blood glucose reading to determine the dosage. In the case of alarm-based boluses, this "dosing signal" is only checked when a "trigger signal" exceeds a configured threshold (i.e., before an alarm-based correction bolus is delivered, both the trigger signal and the dosing signal must be above a configured threshold.) While these signals, and the sensors that produce them, are configurable, the following table shows typical options:

	Sensor Name	Signal Name	Used by Correction Bolus type
Trigger signal	CGM -	CGMBG -	Alarm-based
Dosing signal	SMBG -	SMBG – or BG -	Alarm-based, meal-based, postprandial

Table 6: Correction Bolus Trigger and Dosing Signals

The mealtime and postprandial correction boluses are delivered when, at the applicable time, a dosing signal shows a glucose concentration above a configured threshold. The size of these boluses is calculated based entirely on the dosing signal.

The alarm-based bolus may be delivered in response to a trigger signal showing a blood glucose concentration above a configured threshold – as long as no such event has occurred within a configured "minimum time between boluses" (typically 30 or 60 minutes). When this condition arises, the dosing signal is checked, and, if it exceeds the same threshold, the bolus is delivered, with a size calculated from the dosing signal (not the trigger signal). Note that this sequence may repeat at the configured repetition interval, as long as the trigger signal is still above the threshold when the interval elapses. This allows additional opportunities to deliver the bolus in cases where previous trigger signals coincided with dosing signals that did not yet exceed the threshold.

A minimum time between boluses can be configured (typically 2 hours), which can prevent an overdose of insulin (insulin stacking) caused by:

- An alarm-based bolus following too quickly after any earlier correction bolus (of any type -- mealtime, postprandial, or alarm-based bolus)
- A mealtime bolus following too quickly after any earlier correction bolus
- A postprandial bolus following too quickly after an alarm-based bolus or (in some unusual cases) after another postprandial bolus.

Note that the minimum time between boluses will not affect a postprandial bolus delivered too soon after its corresponding mealtime bolus. This allows the postprandial boluses to be scheduled for a relatively short time after a meal.

6.4.6.2.5.5 *Configuring Drug Dosing*

To add a drug dosing regimen to a treatment plan, select it from the list of *Available Control Types* and press *Add* to move it to the *Included Controls* list. From there press the *Configure Selected Control* button.

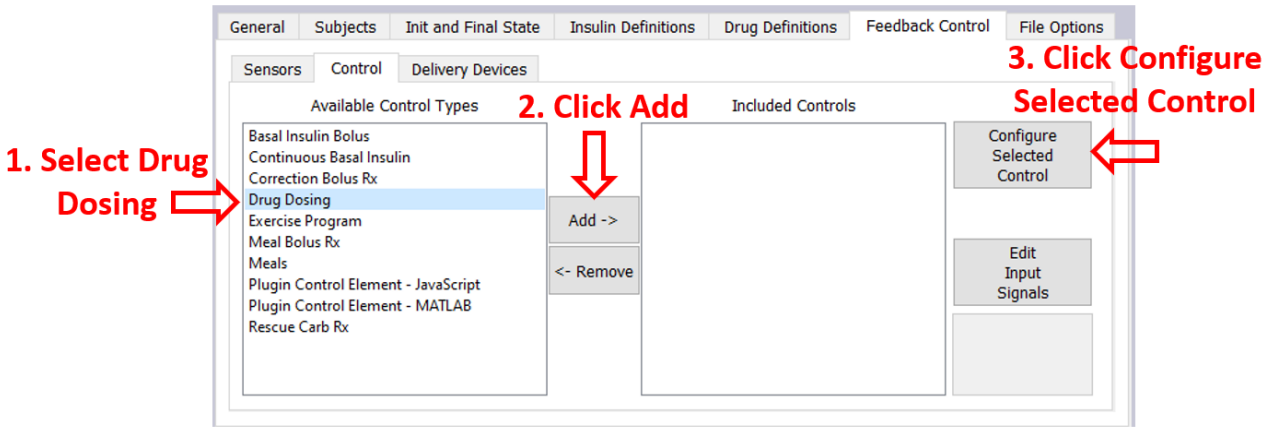


Figure 60: Selecting Drug Dosing control element

The following dialog will appear:

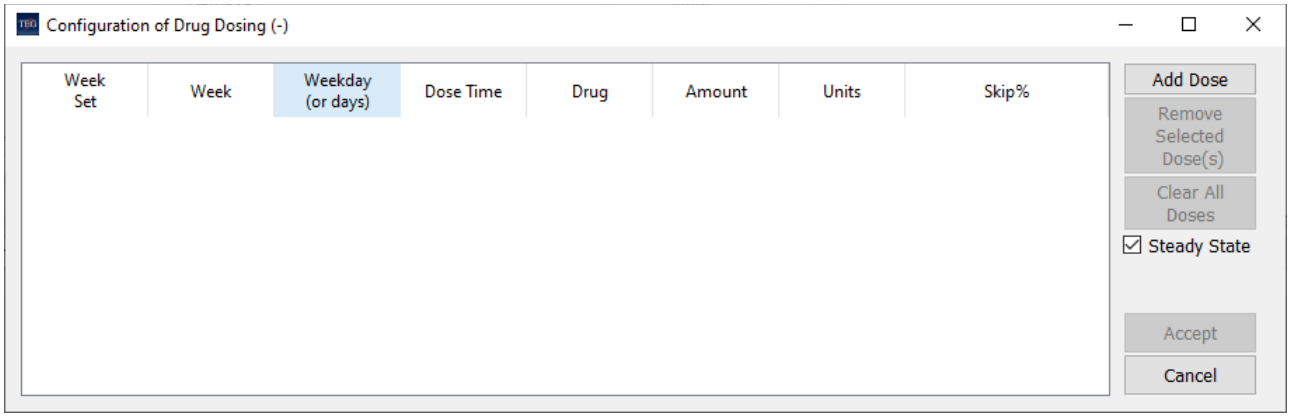


Figure 61: Configuration of Drug Dosing Dialog

Click the *Add Dose* button on the top right and edit the dose.

6.4.6.2.5.5.1 Selecting Weeks

You can configure specific drug dose details to apply to all weeks, just to one week, or to all but one week. The *Week Set* drop-down control enables you to choose from among these options.

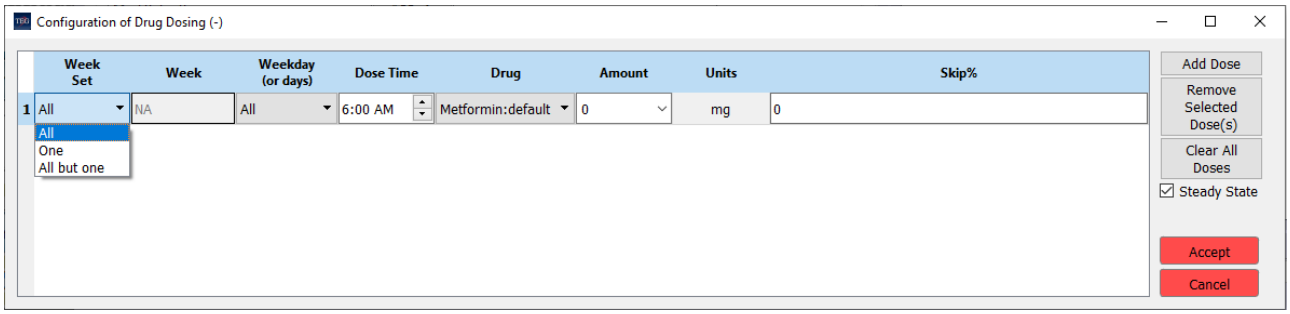


Figure 62: Add Drug Dose -- Selecting a Week

If you choose to apply your drug definition to only one week of the simulation, or to all weeks except one week, specify the week to include/exclude in the *Week* edit box. The first week of the simulation is considered week 1.

6.4.6.2.5.5.2 Selecting a Specific Day or Days

Within your choice for weeks, you may apply your drug definitions to certain days, or to all except certain days. Select the *Weekday (or* days) dropdown and then select the specific day to include or exclude. Use the scroll bar to make additional choices visible.

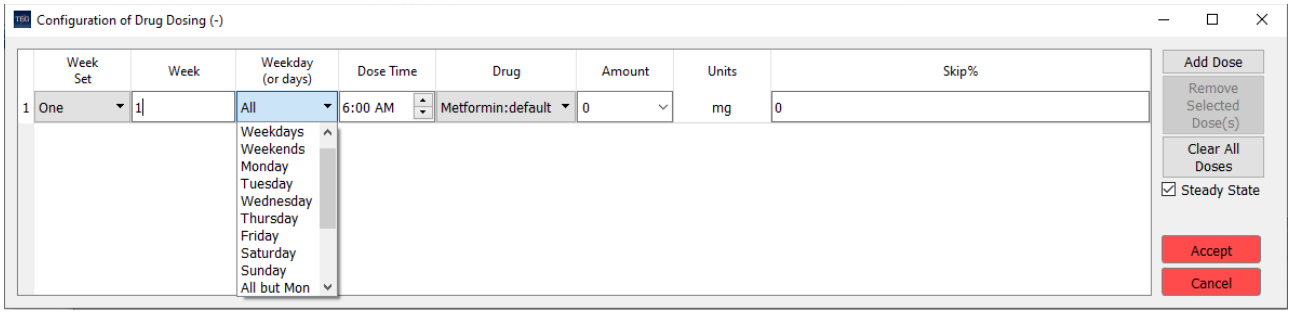


Figure 63: Add Drug Dose -- Selecting a Day

Combinations are also possible. For instance, to configure a similar drug dosing session for all days except Sunday, choose Weekdays. Then add another session, with the same details, for Saturday.

6.4.6.2.5.5.3 Selecting Dosing Amount and Drug Skip Percentage

Once the dosing date time is set to preferred, click the drop-down menu to select the amount of doing, the options are shown in the figure below. To specify the percentage of how often the drug will be skipped, type in a scale between 0 to 100 under *Skip %*. Once you finish editing the dose, click the *Accept* button to accept your change(s). To remove a previously added drug from the list, by clicking the row header to highlight the drug and press the *Remove Selected Dose(s)* button.

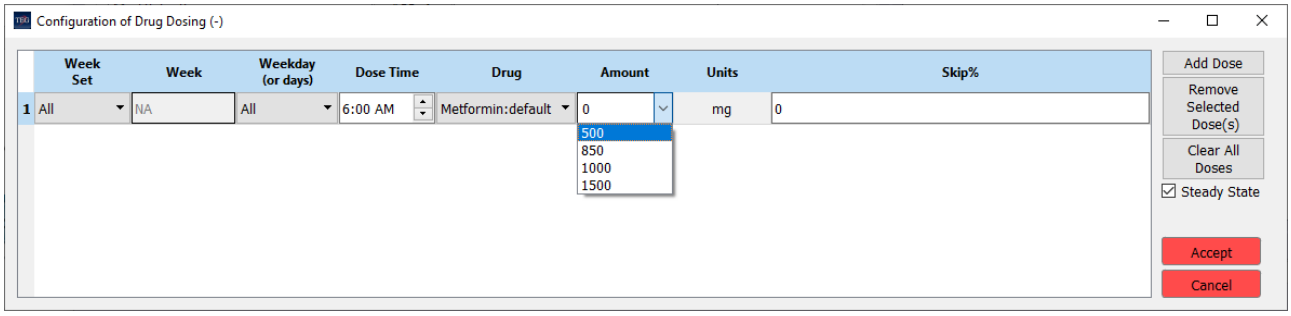


Figure 64: Add Drug Dose -- Selecting dose Amount

6.4.6.2.5.6 *Configuring the Exercise Program Control Element*

To add an exercise regimen select it from the list of *Available Control Types* and press *Add* to move it to the *Included Controls* list. From there press the *Edit Selected Control* button.

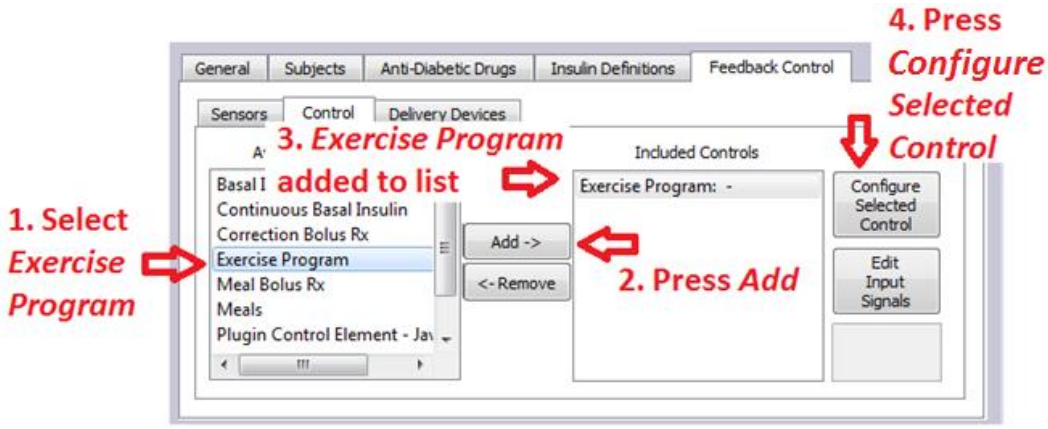


Figure 65: Adding and Configuring the Exercise Program Control Element

The following dialog will appear:

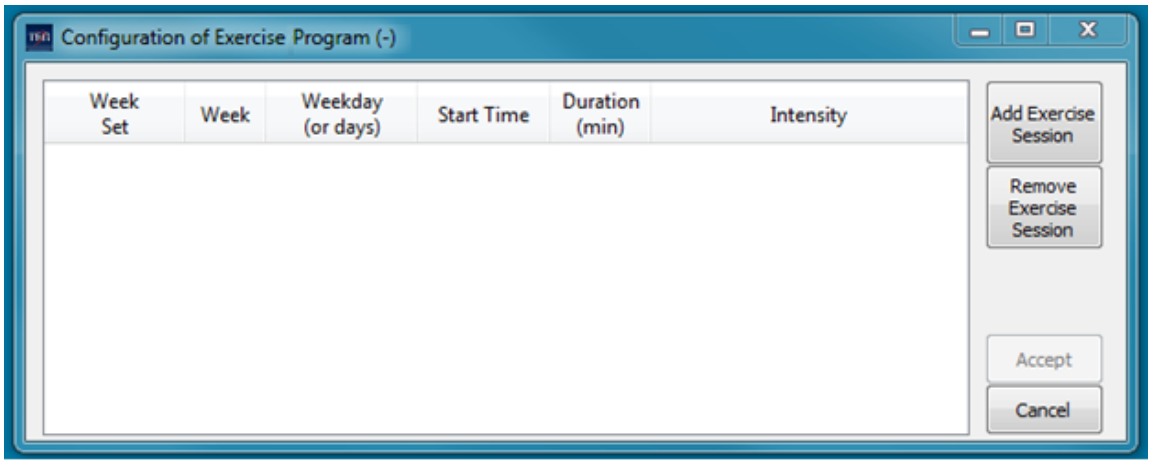


Figure 66: Exercise Configuration Dialog

For each exercise session you want to add, press the *Add Exercise Session* button, and edit the details as needed.

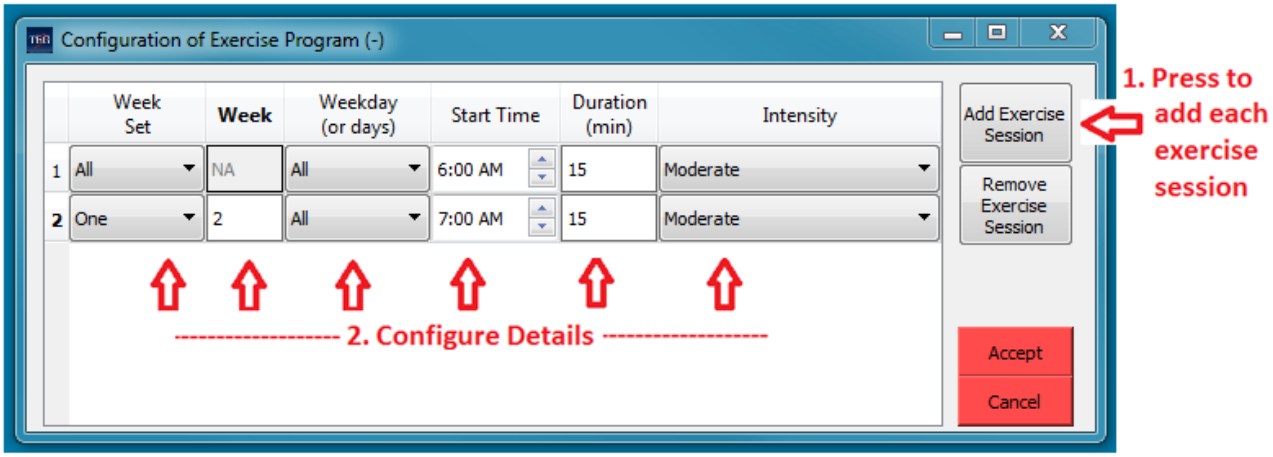


Figure 67: Exercise Configuration Details

6.4.6.2.5.6.1 Selecting Weeks

If you plan to run your simulation for more than a single week, you can configure specific exercise details to apply to all weeks, just to one week, or to all but one week. The *Week Set* drop-down control enables you to choose from among these options.

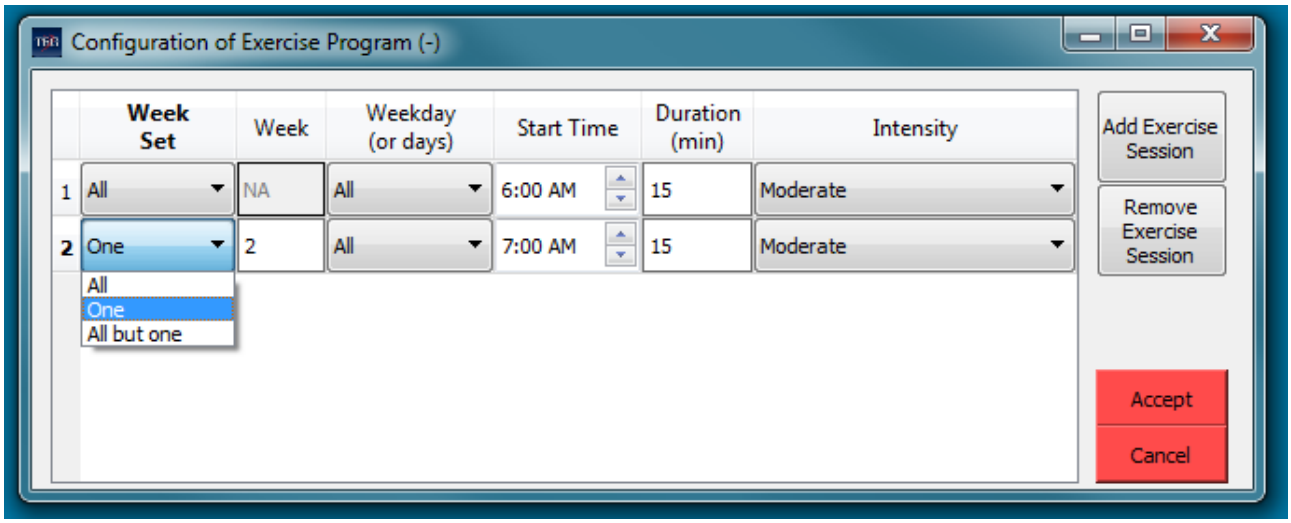


Figure 68: Exercise Program -- Selecting a Week

If you choose to apply your exercise definition to only one week of the simulation, or to all weeks except one week, specify the week to include/exclude in the *Week* edit box. The first week of the simulation is considered week 1.

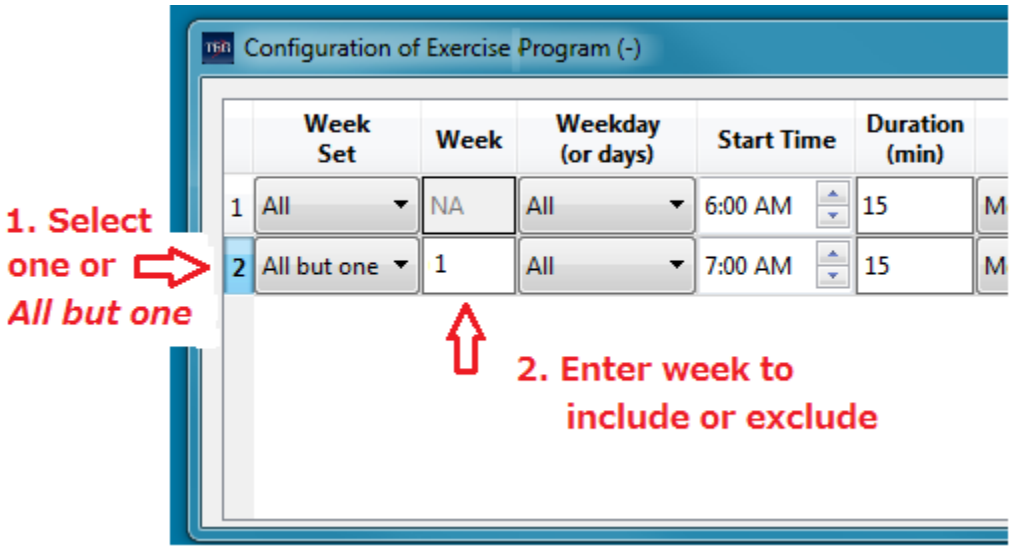


Figure 69: Exercise Program -- Selecting a Week to Include or Exclude

6.4.6.2.5.6.2 Selecting a Specific Day or Days

Within your choice for weeks, you may apply your exercise definitions to certain days, or to all except certain days. Select the *Weekday (or* days) dropdown and then select the specific day to include or exclude. Use the scroll bar to make additional choices visible.

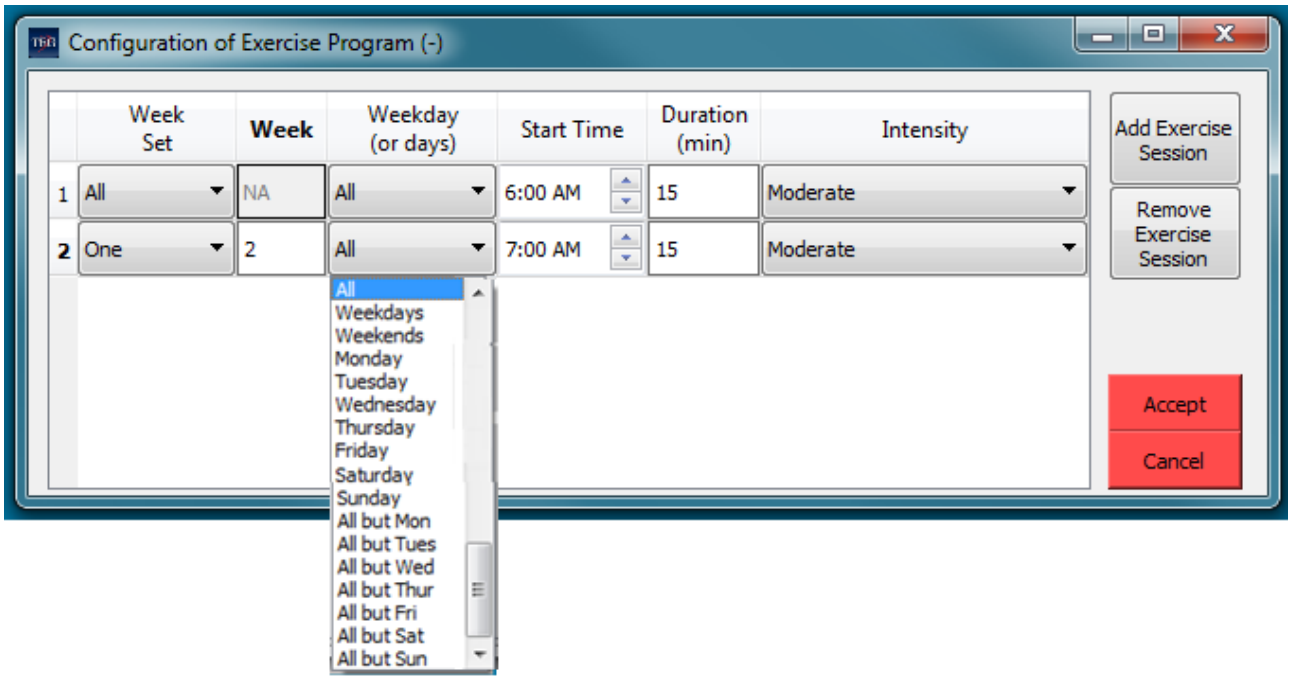


Figure 70: Exercise Program -- Selecting a Day

Combinations are also possible. For instance, to configure a similar exercise session for all days except Sunday, choose Weekdays. Then add another session, with the same details, for Saturday.

6.4.6.2.5.6.3 Selecting Exercise Start Time, Duration, and Intensity

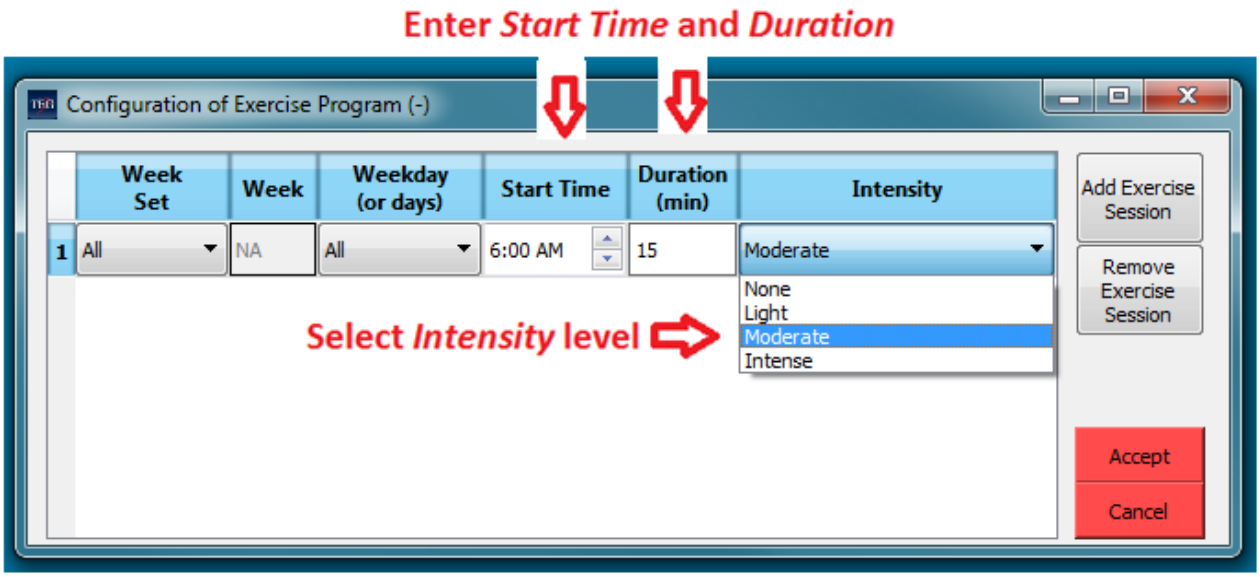


Figure 71: Exercise Program – Selecting Exercise Intensity

Exercise intensity may be defined as light, moderate, or intense. You may also choose an intensity value of "None." "Moderate" intensity is equivalent to that cited in Schiavon's[6] study in which subjects were held at ~50% of VO2max. Other values have been extrapolated from this.

- None (0%)
- Light (25%)
- Moderate (50%)
- Intense (~65%)

Note that these intensity levels, when expressed as percentages, are useful if you should include a Java plugin related to exercise.

When an exercise level of "None" is defined, no exercise, as you might expect, is performed. This choice may be useful to allow (for example) easily switching the configuration between cases of no exercise and a meaningful level of exercise, without losing other settings (e.g., the exercise session's time/day). It may also be used to force a reduced meal bolus size at specific times.

6.4.6.2.5.7 *Configuring the Meal Bolus Rx Control Element*

The Meal Bolus Rx Control configuration dialog is shown below.

[6] Michele Schiavon et al, *In Silico Optimization of Basal Insulin Infusion Rate during Exercise: Implication for Artificial Pancreas* (Journal of Diabetes Science and Technology Volume 7, Issue 6, 2013)

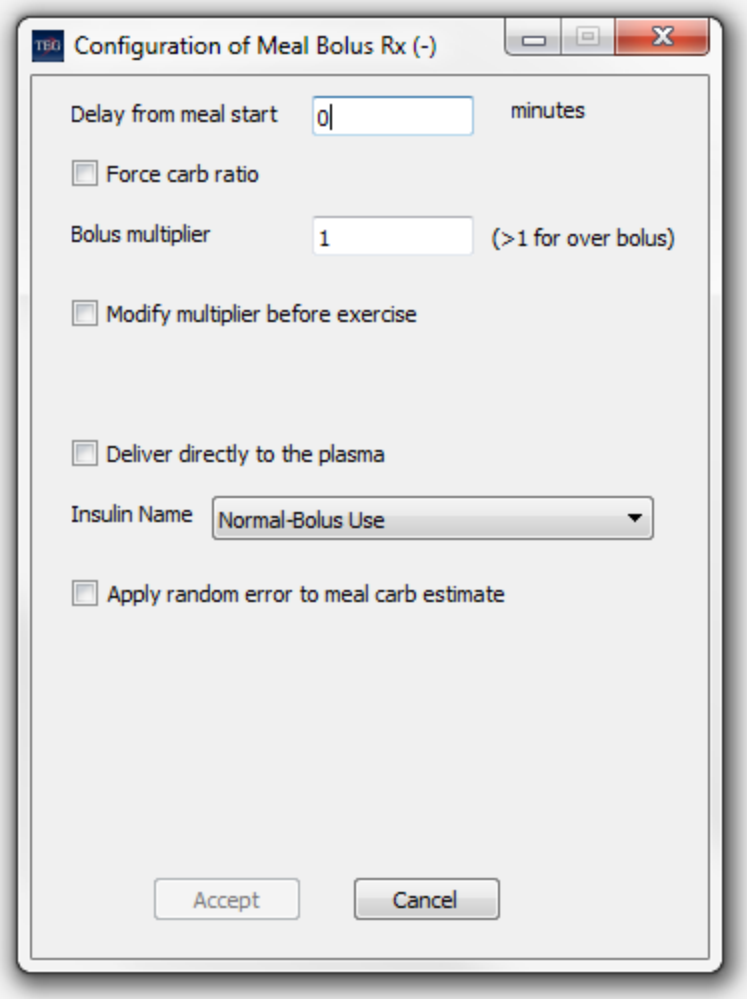


Figure 72: Configuring the Meal Bolus Rx Control

Note that some of the check boxes are un-checked by default. Checking these boxes will allow additional dialog controls to be enabled:

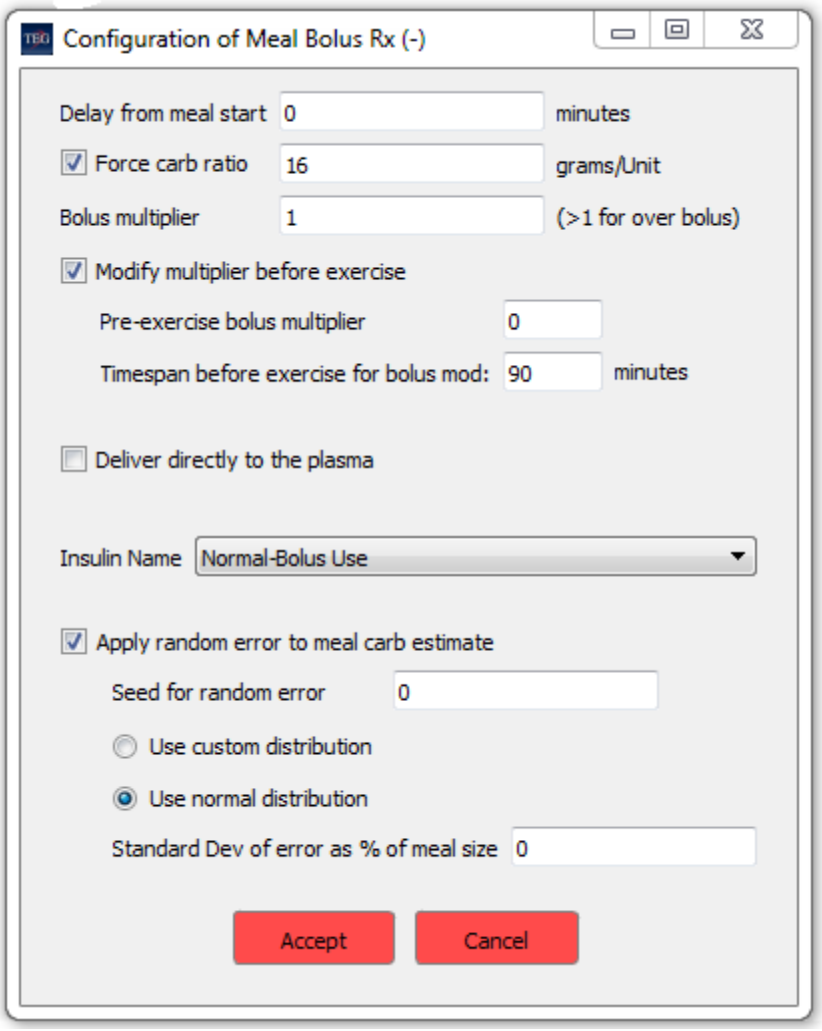


Figure 73: Meal Bolus Rx Control Additional Configuration

This dialog will allow you to specify the following settings:

- The time in minutes, relative to the start of the meal, at which the bolus is to be provided. To deliver the bolus before the meal, enter a negative number.
- Whether the carb ratio (CR) used to determine the bolus size should be explicitly specified or obtained for each subject from the subject database.
- If an explicitly specified carb ratio is chosen, the value of that ratio.
- A bolus multiplier that would be applied to the calculated bolus for all subjects, whether the calculation uses the explicit carb ratio or the subject-specific ones from the subject database. Note that this multiplier is only useful when the subject-specific ratios are used.

- Whether the bolus is delivered subcutaneously or directly to the plasma. (See Section 6.4.6.2.5.1)
- The insulin-release pair to use (For a definition of insulin-release pair, see the related entry in the Glossary, Section 14).
- Whether, in calculating the bolus size, an adjustment should be applied whenever a meal bolus' time comes shortly before the start of an exercise session, and, if such an adjustment is desired, the following:
    - The maximum time prior to an exercise session for which a meal bolus adjustment will be applied.
    - The percentage of the normally calculated bolus that should be used for meal boluses within the specified time of an exercise session.
- Whether a random error is to be applied to the value for the amount of carbohydrates in the meal, as used in calculating the bolus size. If you choose this option, you may specify the seed value to be used in calculating the random error. When the same seed is chosen in two independent simulations, identical sequences of noise will result (facilitating repeatability when desired).

  You may also choose to have the error normally distributed, or you may define a custom distribution. If you choose a normal distribution, you can specify the Standard Deviation of the error to be applied. If you choose a custom distribution, you may define the distribution of the error with the following inputs:
    - Bottom (lower bound) of the distribution range in terms of the percentage of meal size.
    - Histogram resolution, also in terms of % of meal size.
    - A comma separated list of probability values for each sub-range (bucket) in the subrange.

To enter these values, select *Use custom distribution* and press the *Configure Distribution* button on the Meal Bolus Rx configuration dialog.

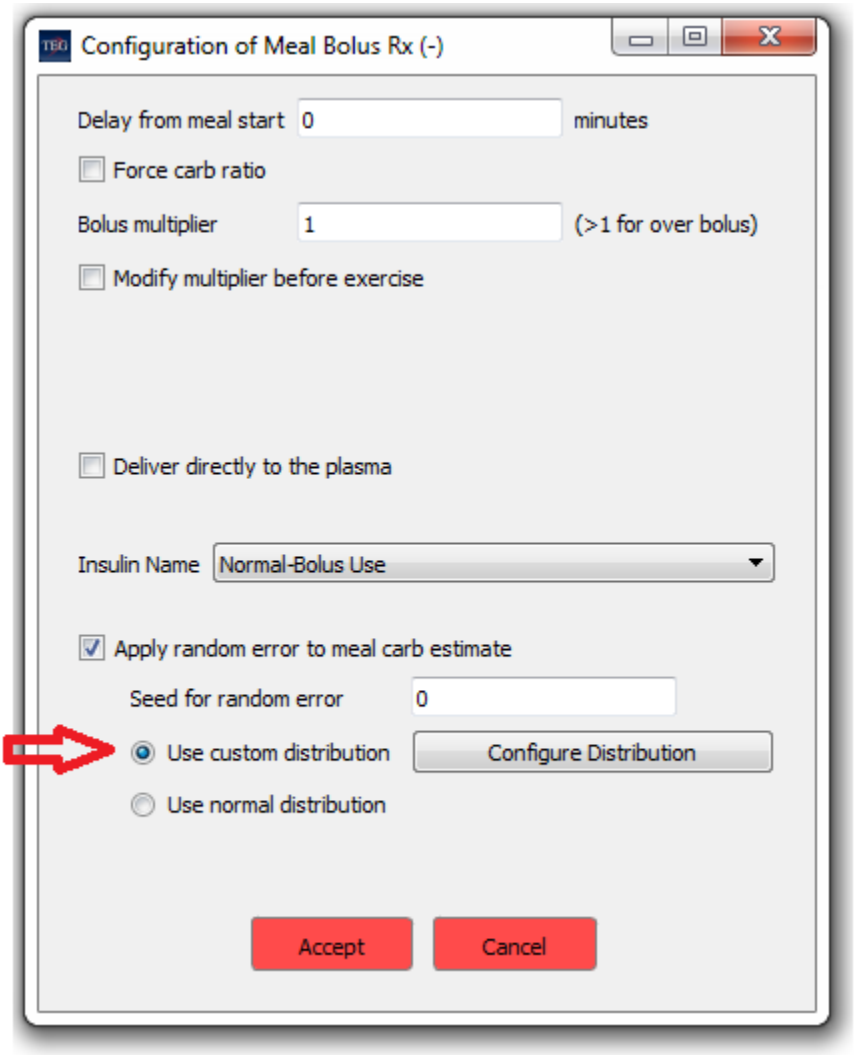


Figure 74: Custom Distribution for Meal Boluses

The following sub-dialog will appear:

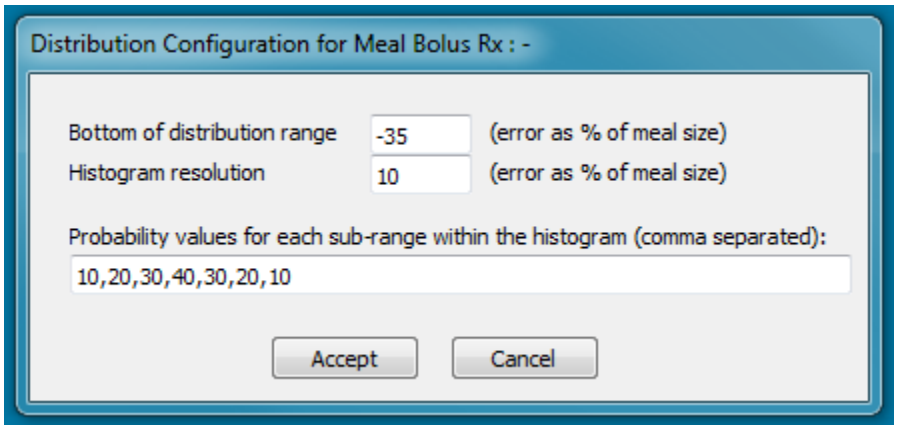


Figure 75: Distribution Configuration for Meal Bolus Rx

6.4.6.2.5.8 *Configuring the Meals Control Element*

Selecting the Meals control and pressing *Edit Selected Control* will produce the following:

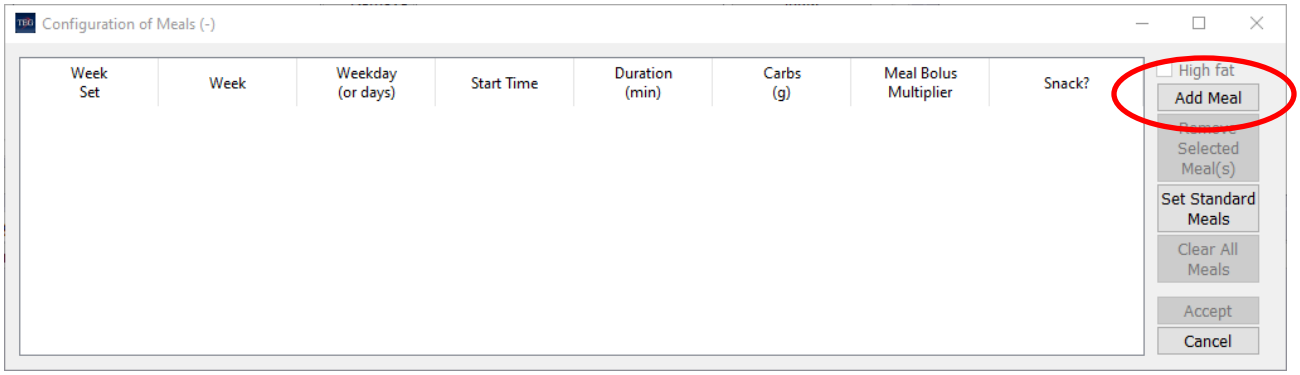


Figure 76: Configuring Meals

Pressing the *Add Meal* button will populate the dialog with additional controls to define the meal.

Most of the inputs for the Meals Control Element are very similar to those for the Exercise Program Control Element. The standard meal is comprised of a mix of Carbohydrate, Protein, and Fat. A “High Fat” meal can be selected to simulate a pizza meal and represents slowed gastric emptying with a higher fat content and higher Carbohydrate content with lower protein content.

To set the meal as a high fat meal, simply check the *High fat* check box above the *Add Meal* button while adding a new meal.

If you are running a multi-week simulation, you can define different meals for different weeks. This works the same way as it does for an exercise regimen. (See Section 6.4.6.2.5.6.1)

Other inputs include the meal *Start Time*, *Duration* (in minutes), *Amount* (in grams), and the Meal Bolus Multiplier. Meal content is modeled after that used in clinical studies by Rita Basu et al[7].

The Meal Bolus Multiplier works automatically with Meal Bolus Rx (See Section 6.4.6.2.5.7). The Meal Bolus Multiplier may also be included in the input to any plugin control element.

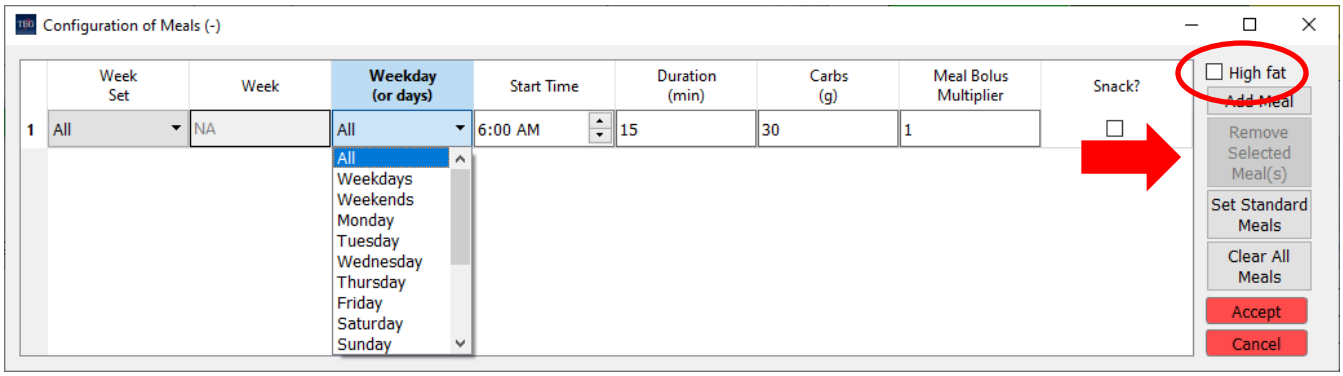


Figure 77: Selecting a Specific Day for Meals

To remove a meal, select the meal to be removed and press *Remove Meal.*

Pressing the *Set Standard Meals* button will populate the meal plan with a predefined set of meals and snacks.

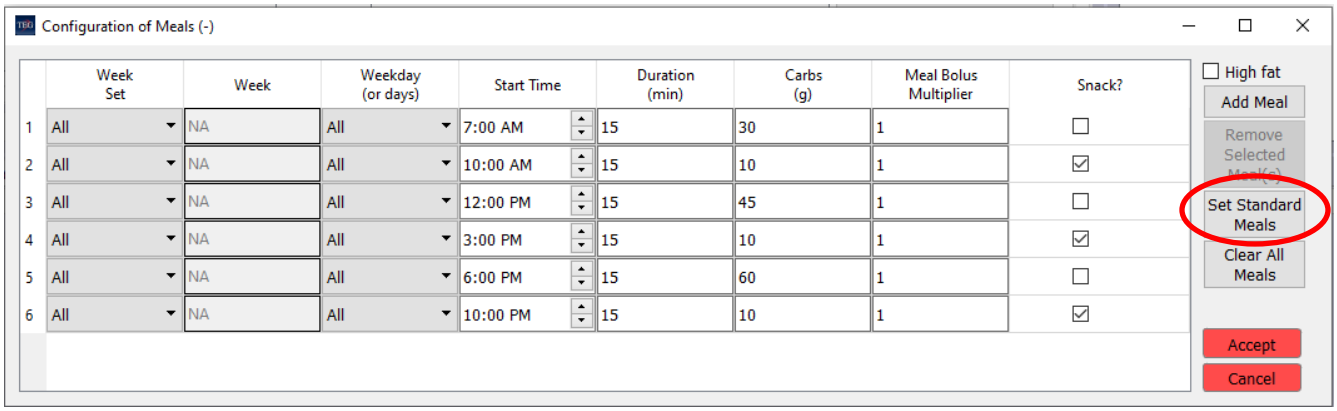


Figure 78: Setting Standard Meals

Of course, you are free to add additional meals, remove meals, or change the details of a meal on an individual basis.

[7] R. Basu, B. D. Camillo, G. Toffolo, A. Basu, P. Shah, A. Vella, R. Rizza, and C. Cobelli, "Use of a novel triple tracer approach to assess postprandial glucose metabolism," *Am. J. Physiol. Endocrinol. Metab.*, vol. 284, pp. E55–69, 2003.

6.4.6.2.5.9 *Configuring the JavaScript Plugin Control Element*

The JavaScript plugin control feature of DMMS.R provides a powerful tool you can use to define a custom control element. The use of JavaScript plugins is supported in the Windows version of DMMS.R.

Use of this control type requires that you have an appropriate JavaScript file to support. The plugin control configuration dialog lets you specify the path to your JavaScript file, either by using the browse feature, or by typing it into an edit box. The location of the plugin file may be specified by either an absolute path or a path that is relative to your default configuration directory (see section 6.3.1.1). This is controlled by a pair of radio buttons. Using relative paths will allow your configuration files to be shared more easily with users who may want to perform simulations in a different directory location on their own computers. Note that the DMMS.R will always display both the relative and absolute path of the plugin file, regardless of which choice is made via the radio buttons.

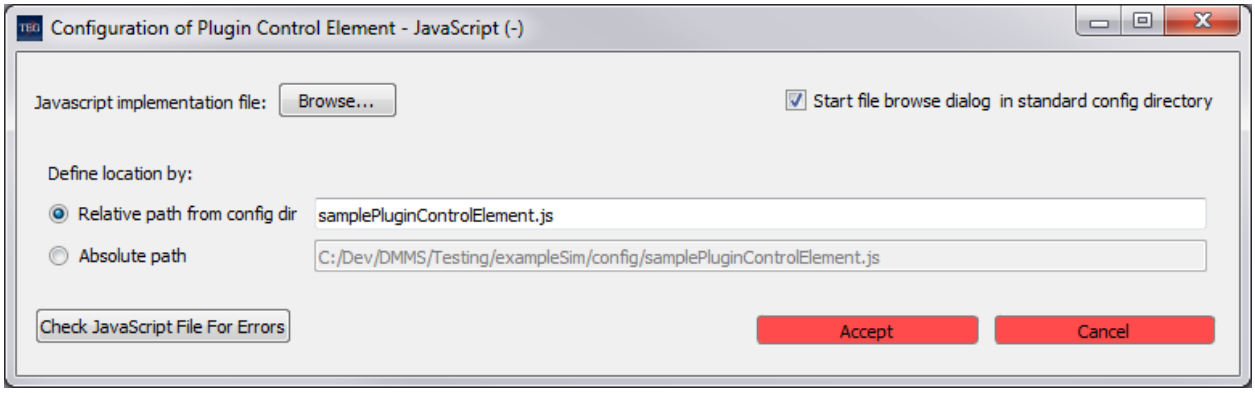


Figure 79: Configuring a Control Element with JavaScript.

As with sensor and delivery element JavaScript plugins, you can press The *Check JavaScript File For Errors* button to ensure the file is error-free.

The JavaScript file you select may (and typically will) require input signals. Unlike other control elements (e.g., Correction Bolus Rx, Rescue Carb Rx) these input signals are not predefined by the application, but rather within the JavaScript file itself (See Section 9.1.2). However, just as with other control elements, you must select the names of the actual signals to be supplied to these signal inputs. (See Section 6.4.6.2.4)

For an example of JavaScript plugin configuration, see Section 12.

For details on writing the code for a JavaScript plugin control element, see Sections 9 and 9.1.2)

Note that the DMMS.R installation includes an example JavaScript file "samplePluginControlElement.js" which shows the JavaScript functions to be implemented as well as the full set of parameters that are made available to the JavaScript by DMMS.R.

6.4.6.2.5.10 *Configuring the MATLAB Plugin Control Element*

The MATLAB plugin control element is available on the Windows version of DMMS.R only.

Configuring the MATLAB control element is similar to configuring the JavaScript control element (see section 6.4.6.2.5.9). There are, however, a few differences:

- The MATLAB control element plugin's configuration dialog displays the settings that are identified within the plugin's code.
- MATLAB requires a license from MathWorks; you must have this license in place.
- When you configure a MATLAB control element plugin, DMMS will connect to the MATLAB engine. This process may take a few seconds, during which, DMMS will display an indication that the loading is in progress. (See Figure 80 below).

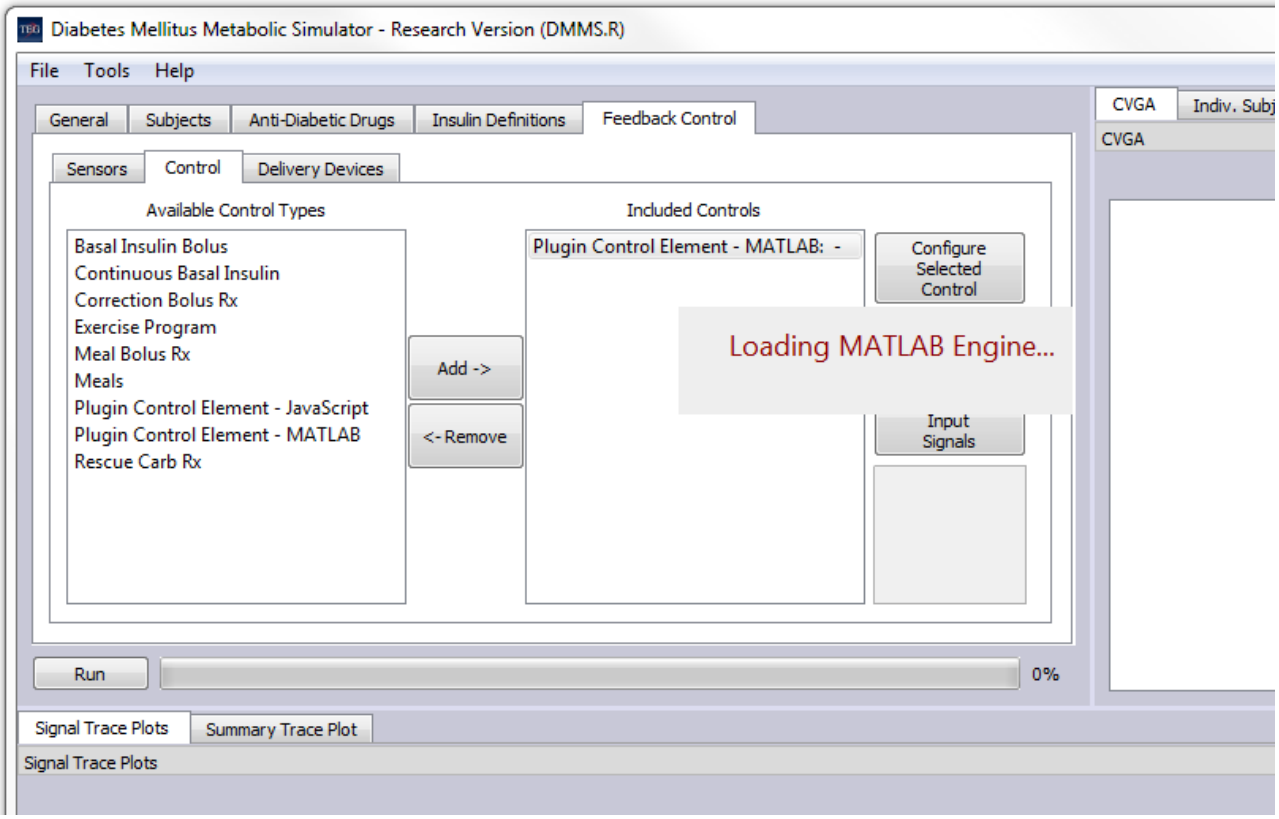


Figure 80: Connecting to the MATLAB Engine

After selecting a MATLAB implementation file, the configuration dialog is populated with information from the associated MATLAB setup file. Note that these values cannot be modified through the dialog; any changes must be made to the setup file itself (see Section 9.2).

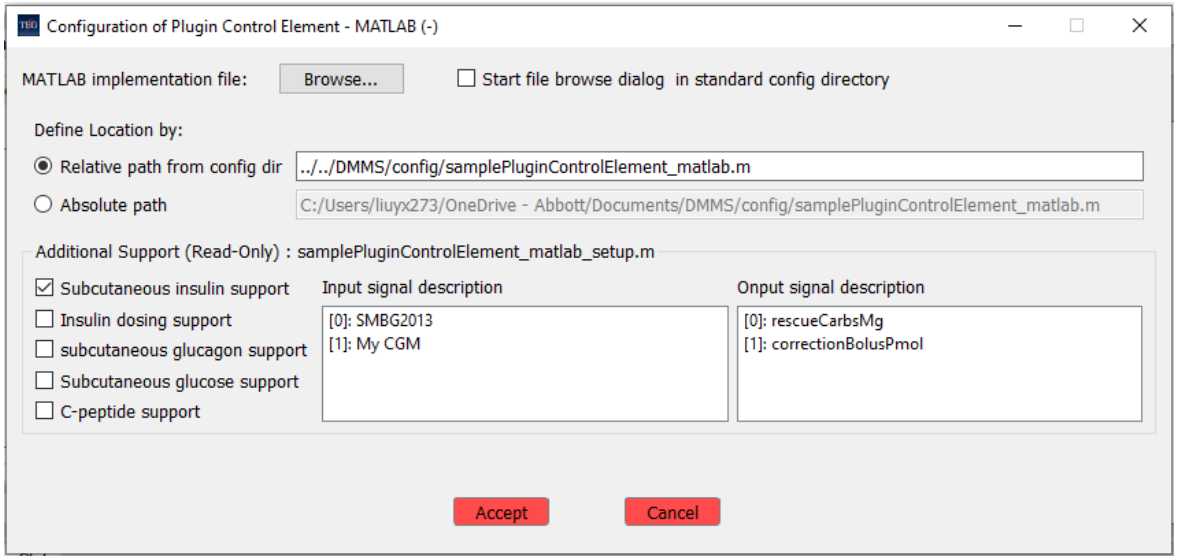


Figure 81: Configuring a Plugin Control Element with MATLAB

The MATLAB file you select may (and typically will) require input signals. Unlike other control elements (e.g., Correction Bolus Rx, Rescue Carb Rx) these input signals are not predefined by the application, but rather within the MATLAB file itself (See Section 9.2.2). However, just as with other control elements, you must select the names of the actual signals to be supplied to these signal inputs. (See Section 6.4.6.2.4).

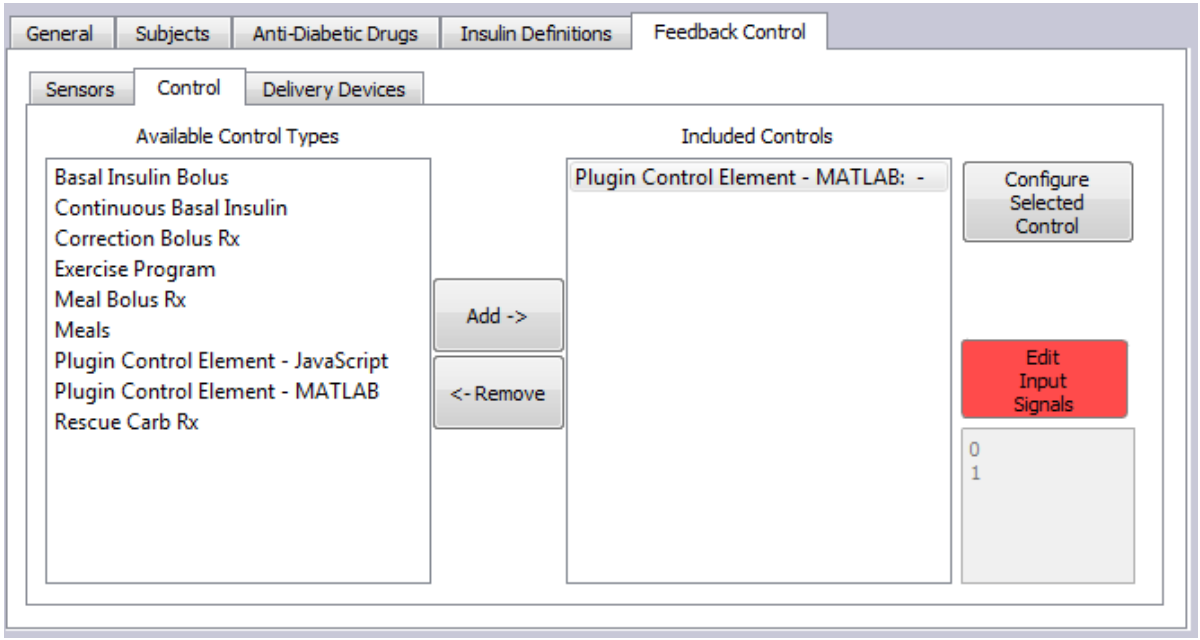


Figure 82: MATLAB Plugin Signal Definition

For details on writing the code for a MATLAB plugin control element, see Sections 9 and 9.2.2. In addition, an example MATLAB plugin control element is provided in a pair of files ("samplePluginControlElement_matlab.m" and "samplePluginControlElement_matlab_setup.m") in the config directory. These files show the functions to be implemented by the user, show the parameters, and return values they use.

6.4.6.2.5.11 *Configuring the Rescue Carb Rx Control Element*

You can configure DMMS.R to deliver a rescue carb (carbohydrate) tablet to the subject whenever the subject's blood glucose level falls below a configured threshold.

The model handles rescue carbs in a manner similar to carb intake from meals or snacks, except that the effect on blood glucose is much more rapid.

Each time rescue carbs are delivered, a minimum time must elapse before another rescue carb delivery can occur. You may configure this time interval, the threshold used to trigger the delivery, and the size of the rescue carb tablet, using the following dialog:

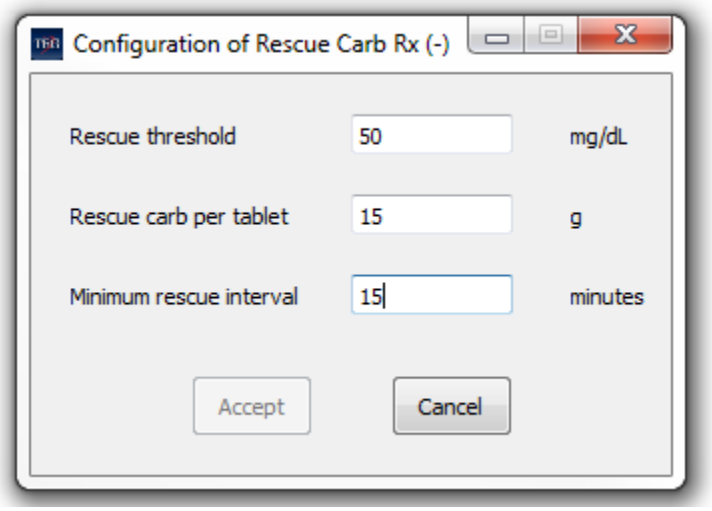


Figure 83: Configuring the Rescue Carb Control Element

6.4.6.3 Delivery Devices

The Delivery Devices subtab is shown below.

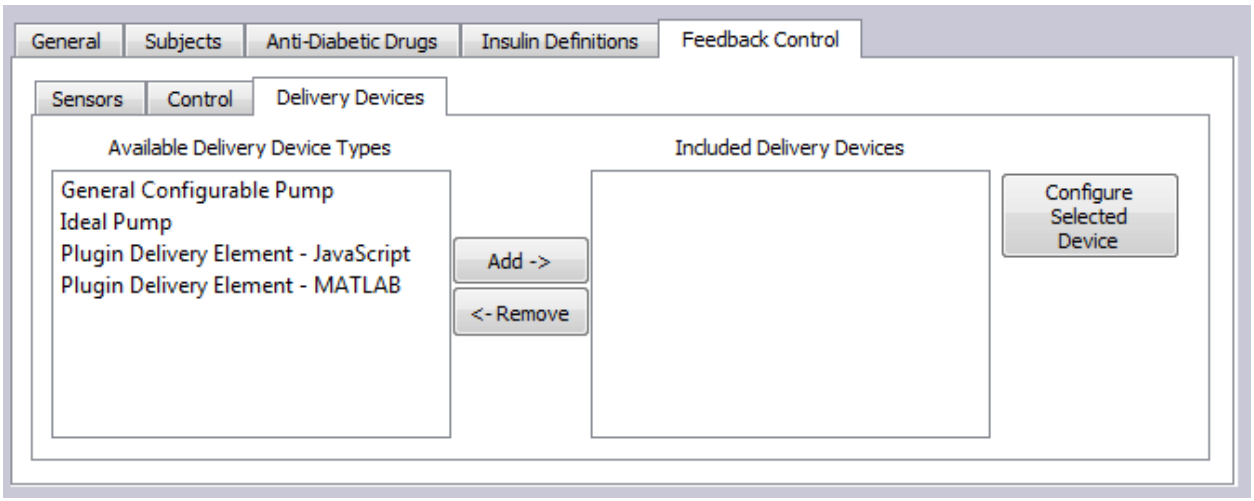


Figure 84: Delivery Devices Subtab

6.4.6.3.1 Adding a New Delivery Element

The process for adding a new delivery element is very similar to that for adding a new Sensor. (See Section 6.4.6.1.1)

6.4.6.3.2 Removing a Delivery Element

The process for removing a delivery element is very similar to that for removing a Sensor. (See Section 6.4.6.1.2)

6.4.6.3.3 Changing the Name or List Position of a Delivery Element

The process for changing the name or list position of a delivery element is very similar to that for changing the name or list position of a Sensor. (See Section 6.4.6.1.3)

6.4.6.3.4 Configuring the General Configurable Pump

The general configurable pump is a delivery element whose input values (provided by various control elements) are desired insulin amounts, and whose output values are the set of insulin amounts that are delivered to the subject. The delivered amounts will be modifications of the desired amounts, based on the configured characteristics of the pump, reflecting the frequency with which the pump can provide injections, the smallest increment the pump can provide, noise magnitude, etc.

The general configurable pump will accept two insulin input values – one for basal-use insulins, and one for a bolus use. Each input may be defined to use any insulin-release pair of the assigned use.

To configure the General Configurable Pump, add it to the list of *Included Delivery Devices*, select it, and press *Edit Selected Device*. The following dialog will appear:

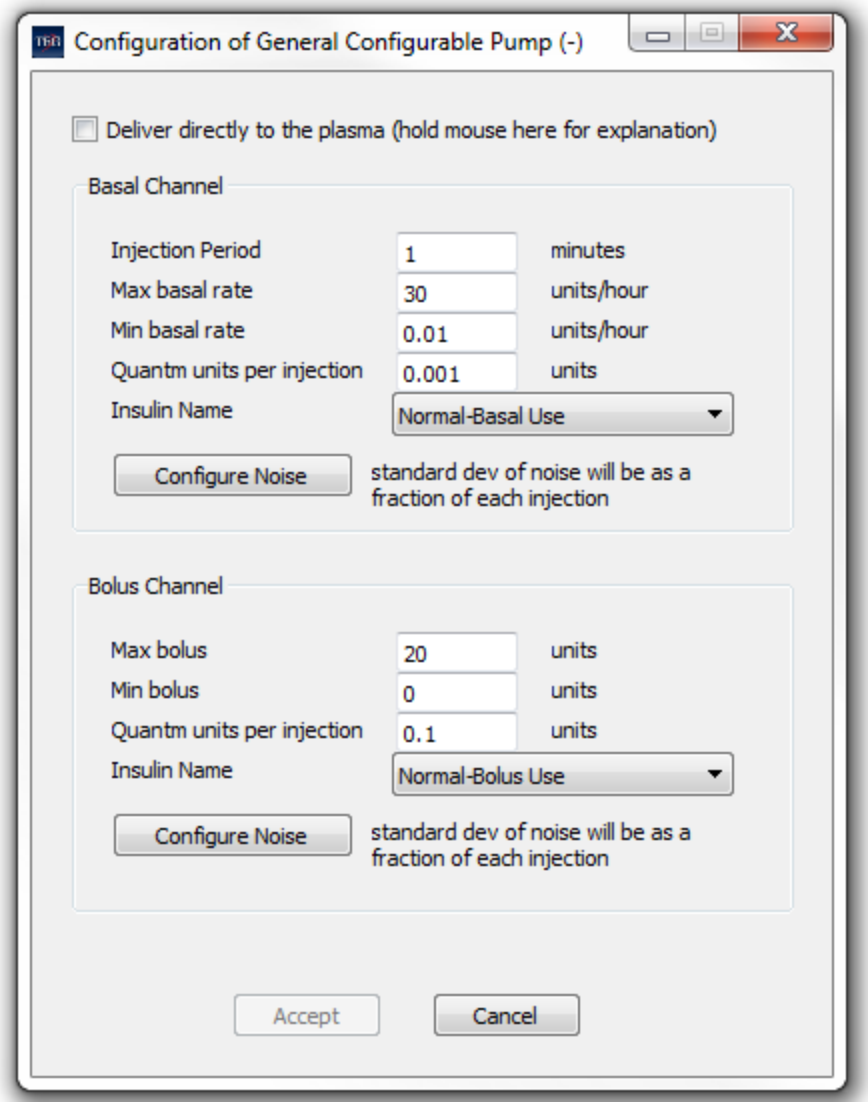


Figure 85: Configuring the General Configurable Pump

This dialog supports the following configuration elements:

- Whether injections are subcutaneous or directly to the plasma. Note that the need for direct-to-plasma injection is rare. One possible instance would be to simulate a clamp study.
- For the basal channel of the pump, the frequency at which basal injections are provided (e.g., once per minute or once every three minutes, etc.).
- For both the basal and the bolus channel of the pump, independently:
    - The **insulin-release pair** to be used.
    - The maximum and minimum amounts that can be delivered in a single injection.
    - The quantum amount of which all injections must be a multiple.
    - Whether Gaussian white noise should be added to the signal.

If you wish to apply Gaussian white noise to the signal from either the Basal or Bolus channel, press the appropriate *Configure Noise* button. A dialog box should appear:

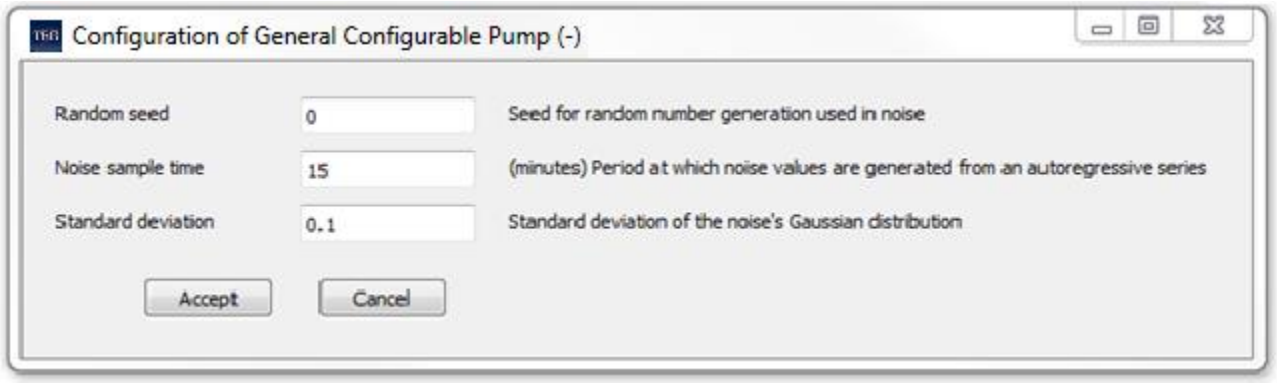


Figure 86: Configuring the Standard Insulin Pump

This dialog allows you to configure:

- The standard deviation of the normally distributed noise, as a percentage of the desired injection.
- A sample time indicating how frequently new raw noise samples are to be generated. These noise values may be filtered to generate samples on each iteration in which the pump updates its injection value. Such filtering will have a time constant that does not exceed the sampling time.
- A random seed, applied to the noise such that, when the same seed is chosen in two pumps or two independent simulations, identical sequences of noise will result (facilitating repeatability when desired).

6.4.6.3.5 Configuring the Ideal Pump

The ideal pump is one that operates at 100% efficiency, delivering the configured bolus immediately. The ideal pump has no configurable parameters.

6.4.6.3.6 Configuring a Plugin Delivery Element

Configuration of the plugin delivery element consists of specifying the path to the JavaScript or MATLAB file that defines the plugin. This is similar to configuring a plugin for a sensor element (see Section 6.4.6.1.5.5).

An example file, “samplePluginDeliveryElement.js” is provided in the config directory. This file shows the functions to be implemented by the user and illustrates how the intended subject input values are passed to the delivery element and modified to produce the actual subject input values (see section 3.2). A similar example file, “samplePluginDeliveryElement_matlab.m,” is provided in the config directory to show how this is accomplished for MATLAB plugin delivery elements.

6.4.7 Configuring Options for Saving Signal Histories

On the Windows version of the DMMS.R, the File Options tab allows you to specify the format(s) used for saving signal histories (see section 5.6.1 and Appendix A: Output Data Definition) automatically at the end of a simulation. The DMMS.R supports 2 formats – comma separated value (.csv) files and MATLAB data (.mat) files.

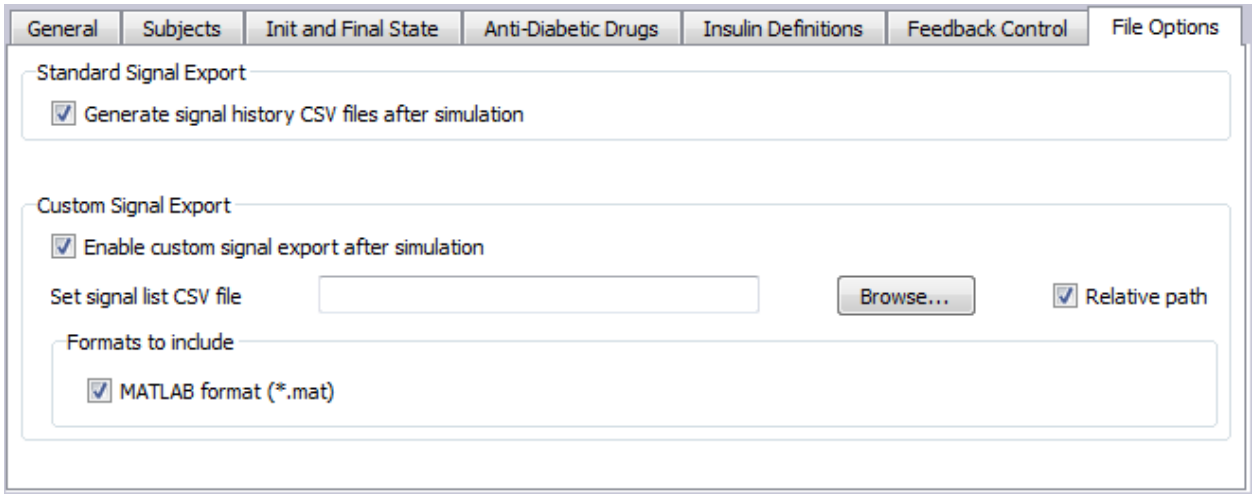


Figure 87: File Options for Saving Signal Histories

The *Generate signal history CSV files after simulation* checkbox is checked by default. You can de-select this option, but doing so will prevent you from re-loading your simulation results into the DMMS.R at a later date (see section 7.7.1). If all your data analysis is to be done using MATLAB, however, you can save much disk space in a long simulation if you only export to a .mat file, since this is a compressed format, and you can limit the saved signals to only those you require.

The *Custom Signal Export* group box can be used to select whether signal histories will be automatically saved as .mat files, and to specify the subset of signals to be included in these files.

Check the *Enable custom signal export after simulation* and the *MATLAB format (*.mat)* checkboxes to make the DMMS.R save signal histories to a .mat file when the simulation completes. You specify the set of signals included in the .mat file by selecting a "signal list csv" file using the other controls in the *Signal Export* group box. The signal list csv file can be created via the Custom Data Output menu item (see section 7.7.2). Note, however, that doing so requires you to first run a simulation so the DMMS.R can "learn" your simulation's full set of available signals from which you can make your choices. You can choose whether to provide the path to this signal list csv file either as a relative or absolute path. When a relative path is provided, the DMMS.R considers it to be relative to your default configuration directory (see section 6.3.1.1).

6.5 Running the Simulation

Once you have entered the data required to setup the simulation, simply press the *Run Simulation* button on the dialog. Once the simulation has started, progress will be indicated by the green band on the bar to the right of the button.

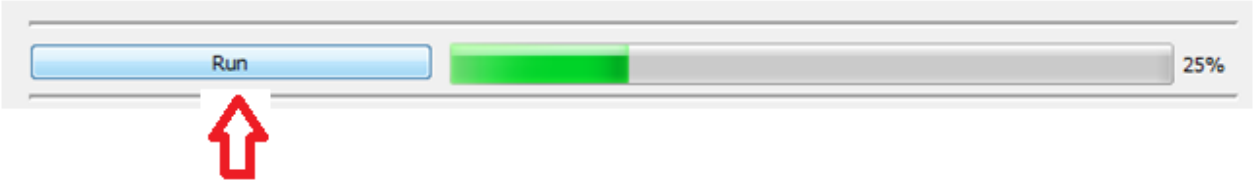


Figure 88: Simulation Progress Bar

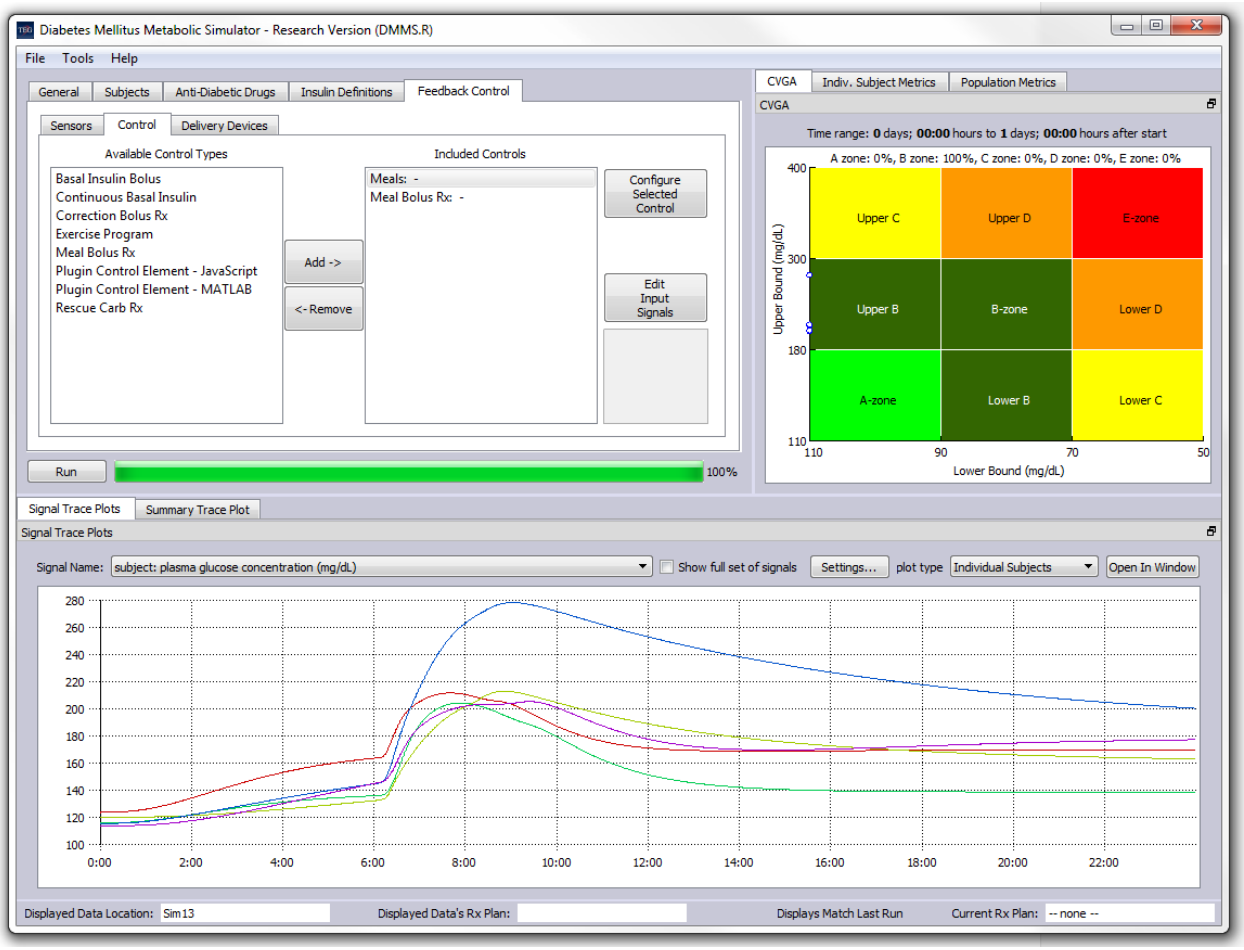


Figure 89: Simulation Results

Output data, along with several additional dialog controls, become visible as soon as the simulation is complete.

Note a simulation may also be run from the command line. (See Appendix B: Running Simulations from the Command Line Interface).

7 Data Output Display

After each simulation is run, DMMS.R will display the following:

- Signal Trace Plots
- Summary Trace Plots
- CVGA display
- Individual Subject Metrics
- Population Metrics

The Signal Trace Plots and Summary Trace Plots appear in separate tabs on the lower half of the main DMMS.R dialog. The CVGA display, Individual Subject Metrics and Population Metrics are presented in the upper right corner of the dialog (See Figure 11.)

7.1 Signal Trace Plots

After you have run your simulation, you may alter the signal trace plot in several ways:

- Choose the output data variable from among all available model output.
- For simulations lasting more than a single day, choose the start and end time (day, hour, minute).
- Choose the labels on the x-axis to be either the day as a number (counting the first day of the simulation as day 1), or the day as a name (e.g., “Saturday”, “Sunday”, etc.).
- Choose to display the data from each subject individually, or as a single trace representing the average for the entire population. Data for the population are displayed as quantiles of 5, 50, and 95%.

Note that making any of these selections does not require that the simulation be re-run; they merely represent different ways of displaying the output data that have already been generated.

7.1.1 Selecting the Value to Display

Select the signal value to display from the contents of the *Signal Name* drop down:

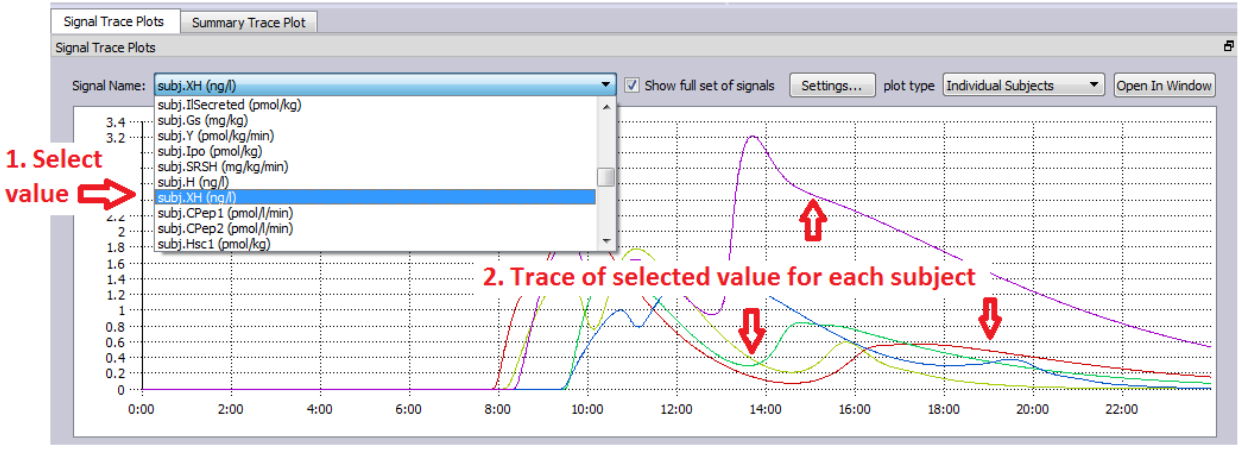


Figure 90: Selecting the Value to Display

Note that you may select from a short list of frequently selected values, or from a longer, complete list. To toggle between these two, check or uncheck the box labeled *Show full set of signals* (See Appendix A: Output Data Definition).

7.1.2 Selecting the Plot Type

You may choose to display a separate trace for each subject in the simulation, or a trace representing the 5, 50, and 95% quantile values for the population:

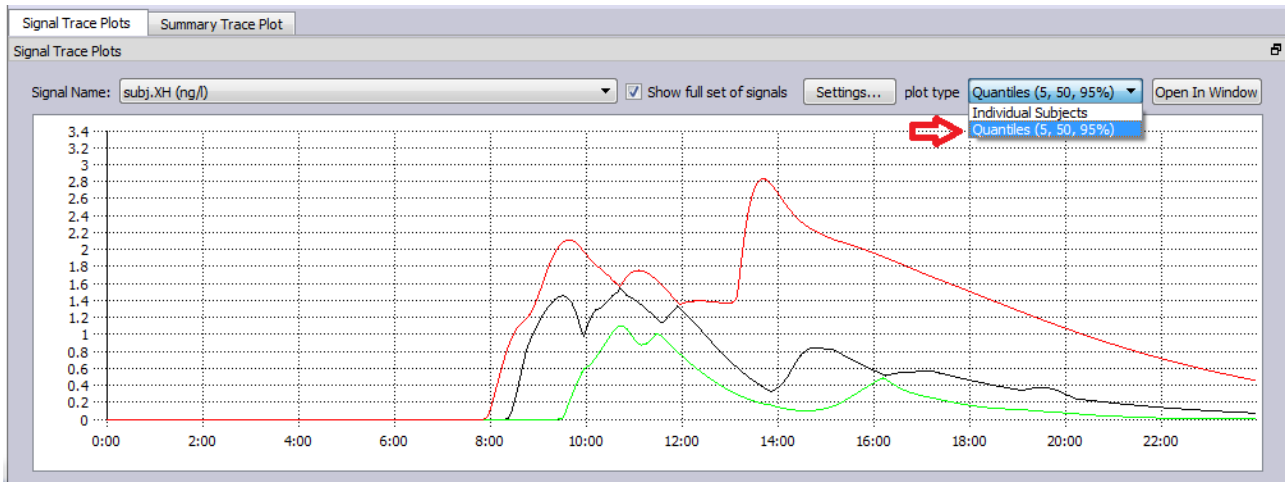


Figure 91: Selecting the Plot Type

7.1.3 Displaying a Subset of the Range for a Trace Plot

If you have run a simulation spanning more than a single day, you may select the beginning time and end time for the output display. To do this, press the *Settings…* button in the plot display area. The following dialog will appear:

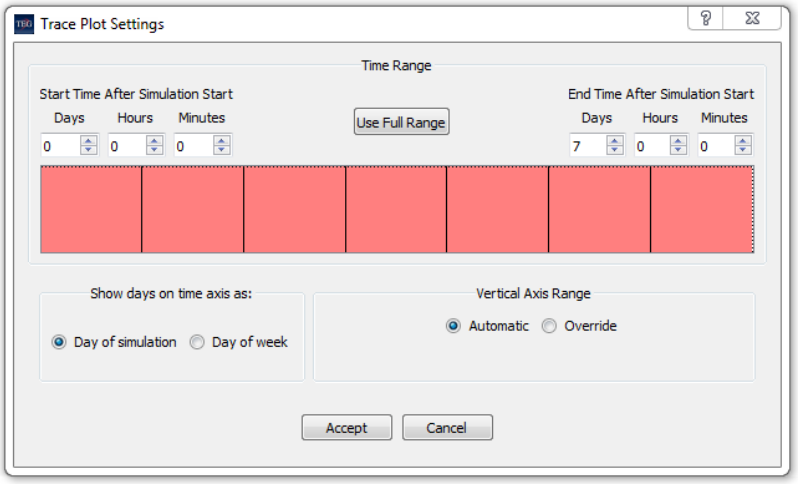


Figure 92: Displaying a Subset of the Range

As shown, the results from an entire 7-day simulation will be displayed:

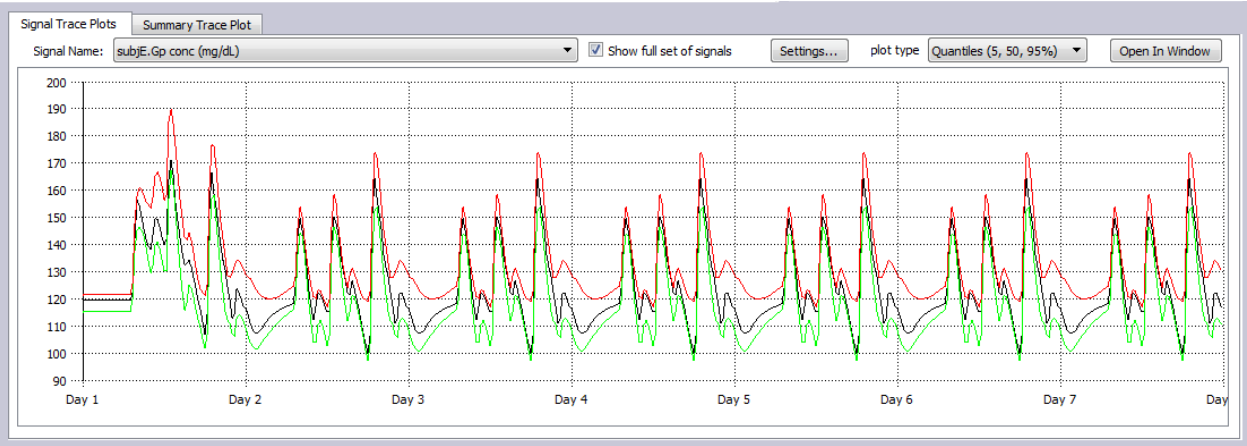


Figure 93: Output from 7-Day Simulation

7.1.4 Displaying Output in Separate Window

You may display output in a separate window, or several windows, each displaying a different signal. Each window can be expanded, in both height and width, for readability:

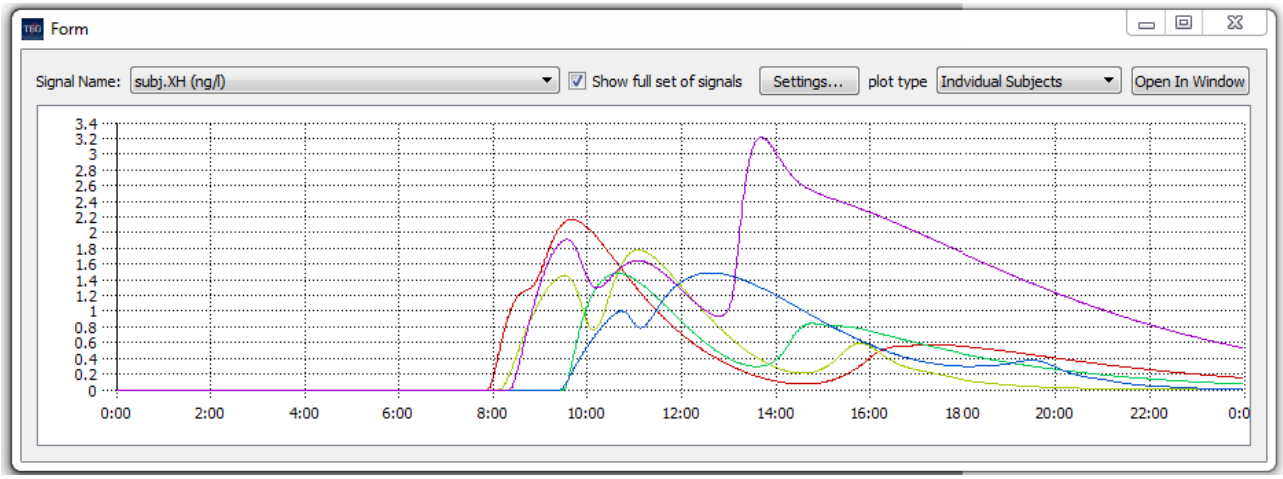


Figure 94: Displaying Output in a Separate Window

To open a new output window, press the Open In Window button:

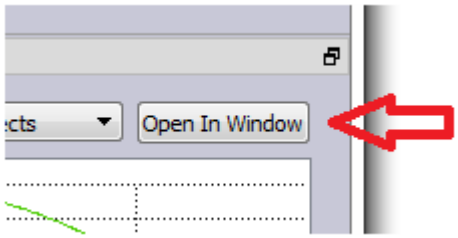


Figure 95: Open in Window Button

7.2 Summary Trace Plots

The Summary Trace Plot shows data for a single subject. You may specify the start day and time span of the data to be displayed. You may also choose to display insulins as the units injected with each bolus, or as IOB (Insulin on Board). Figure 92 shows the signal trace plot, with individual insulin injections, resulting from a one-day simulation with standard meals and default meal boluses. Figure 93 is based on the same simulation but shows insulins as IOB. In both cases the target range is set to 100-180.

Figure 96: Summary Trace Plot (Insulin Injections)

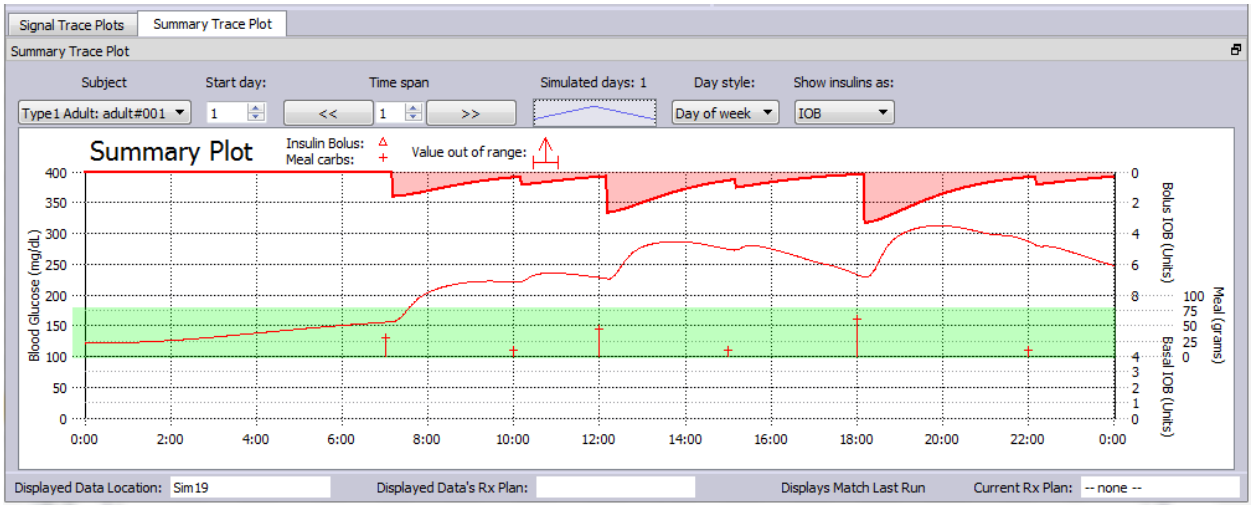


Figure 97: Summary Trace Plot (IOB)

7.3 Control-Variability Grid Analysis (CVGA)

For each simulation DMMS.R displays a CVGA plot, as defined by Lalo Magni[8]. The application establishes the points within this plot by using the maximum and minimum BG values as opposed to using a 95% confidence interval. The reason for this is that Magni specifically identifies the use of the 95% confidence interval as a means of compensating for inaccurate sensor data. Since our CVGA plot is based on perfect BG data, the true maximum and minimum BG values are most appropriate.

Within the CVGA display, the DMMS.R includes an indication of the percentage of subjects in each of the zones (A through E) defined for a CVGA.

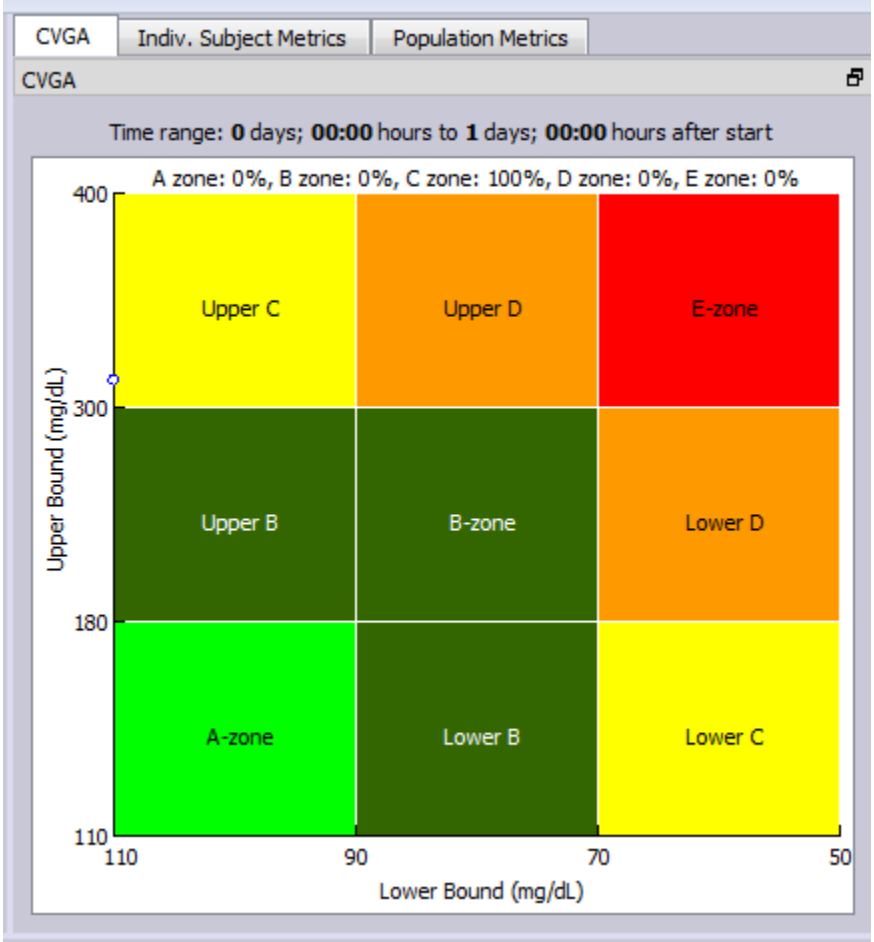


Figure 98: CVGA Plot

7.4 Individual Subject Metrics

For each simulation, DMMS.R presents a table showing, for any selected subject, the following subject state values:

[8] Lalo Magni, P. D., et al (2008 2(4)). Evaluating the Efficacy of Closed-Loop Glucose Regulation via Control-Variability Grid Analysis. Journal of Diabetes Science and Technology, 630-635.

- Mean plasma blood glucose concentration, in mg/dL
- The standard deviation of the collected plasma blood glucose samples
- Maximum plasma blood glucose concentration, in mg/dL
- Minimum plasma blood glucose concentration, in mg/dL
- HbA1c, calculated from the mean plasma blood glucose concentration, in accordance with the equation HbA1c = (mean BG concentration + 46.7) / 28.7
- The number of periods during which the plasma blood glucose concentration remained below the hypoglycemia threshold set by the user
- The percentage of time during which the plasma blood glucose concentration was below the configured hypoglycemia threshold
- The number of periods during which the plasma blood glucose concentration remained above the configured hyperglycemia threshold
- The percentage of time during which the plasma blood glucose concentration was above the configured hyperglycemia threshold
- The percentage of time during which the plasma blood glucose concentration was within the configured target range
- LBGI: The low blood glucose index, as defined by William Clarke[9], but with the following modification: One value of *rl(BG)* is computed for each full hour of the simulation, starting at time 0. Each such *rl(BG)* value is computed based on an $f_{1hr}(BG)$ instead of *f(BG)*, where $f_{1hr}(BG)$ is defined as the mean of all f(BG) values over the applicable 1-hour period. The LBGI value are then calculated from the mean of these *rl(BG)* values, as described in the referenced paper.

[9] William Clarke, M. a. (2009 11(S1)). Statistical Tools to Analyze Continuous Glucose Monitor Data. Diabetes Technology and Therapeutics, S45-S54.

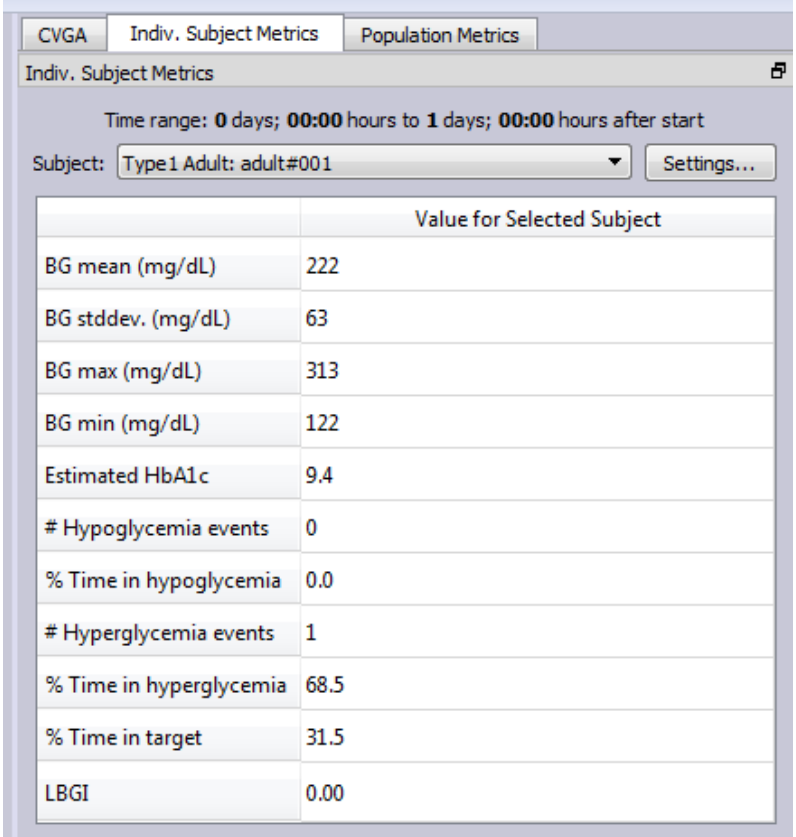


Figure 99: Individual Subject Metrics

You may configure the display of individual subject metrics with these parameters:

- The hypoglycemia threshold, in mg/dL
- The hyperglycemia threshold, in mg/dL
- The lower value for the target range, in mg/dL
- The upper value for the target range, in mg/dL

Press the *Settings* button. An additional dialog will be presented to allow you to define the glycemic threshold and BG target parameters which will define several important metric output values.

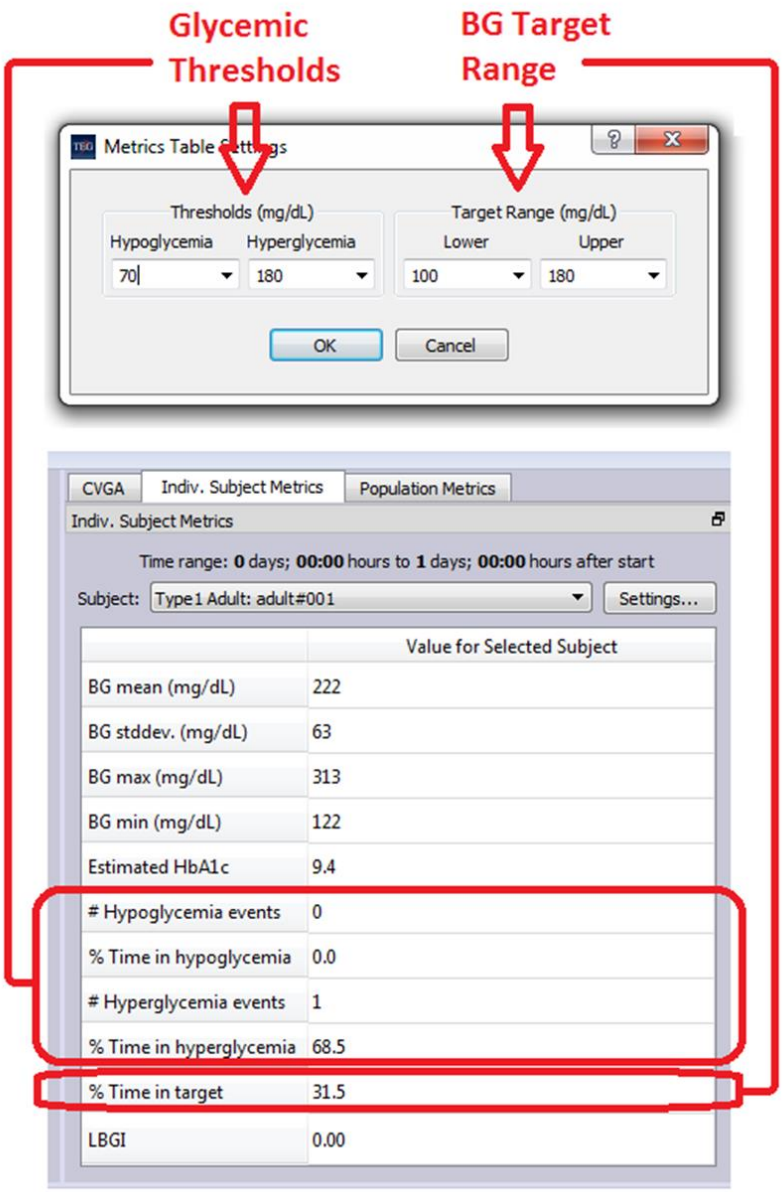


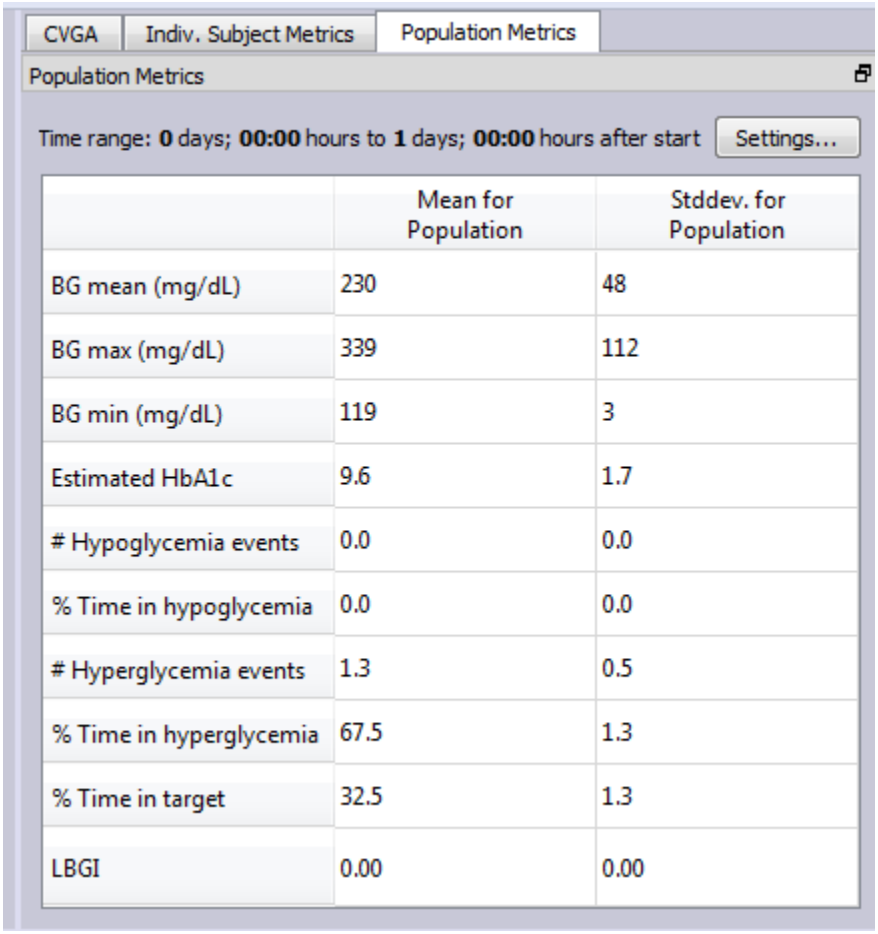
Figure 100: Metric Table Settings

The target range selected here will also be applied to the summary trace plot (the region highlighted in green in that plot). The hypo/hyperglycemic thresholds entered will govern the number of hypo/hyperglycemic events and the "% Time…" values reported.

The glycemic threshold and target range parameters you enter will be saved in your computer's registry, so you will not have to reenter them the next time you start DMMS.R.

7.5 Population Metrics

DMMS.R calculates and presents metrics for the population as a whole, for each simulation run.



	Mean for Population	Stddev. for Population
BG mean (mg/dL)	230	48
BG max (mg/dL)	339	112
BG min (mg/dL)	119	3
Estimated HbA1c	9.6	1.7
# Hypoglycemia events	0.0	0.0
% Time in hypoglycemia	0.0	0.0
# Hyperglycemia events	1.3	0.5
% Time in hyperglycemia	67.5	1.3
% Time in target	32.5	1.3
LBGI	0.00	0.00

Figure 101: Population Metrics

The Population Metrics display shows data generated over the time range for the simulation (by default) or over a sub-range that you can define (see Section 7.1.3).

7.6 Selecting Time Ranges for Metrics and CVGA Displays

To set the time ranges for metrics and CVGA displays, go to the main menu, and select "Tools/ Metrics and CVGA Time Span…." This opens a dialog for choosing the time span applied to the calculation of both metrics displays and the CVGA display. Note that this is independent of the time range settings for the signal trace plots and the summary trace plot.

The currently selected time range will always be displayed at the top of each of the output metrics displays and will be included in the pdf document that is saved via the "File/Save Metrics" menu.

Entry of the start and end of the time range is always in terms of days, hours, and minutes since the start of the simulation (as is the case for the time range selection of signal trace plots). For reference, the selected time range is shown graphically, as a horizontal bar with vertical lines marking either day boundaries or

hour boundaries. So, for example, if a simulation is defined to have a four-day duration and to start at 6AM, selecting a time range that goes from zero days to two days after simulation start would be shown as follows:

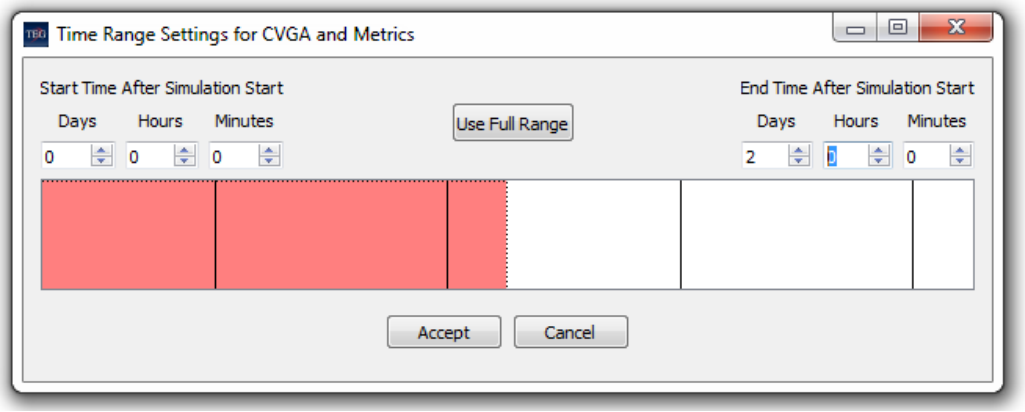


Figure 102: Time Range Settings for CVGA and Metrics

7.7 Saving and Retrieving Simulation Results

DMMS allows you to save the data generated by a simulation in files (see Section 5.6). These files can be used for future reference and analysis or may be loaded back into the application for display. The default location for these files is /Users/<your user name>/Documents/DMMS.R/results. You may change this default location (see Section 6.3.1.1). Of course, the file location, as well as the file name, can be specified when the file is saved.

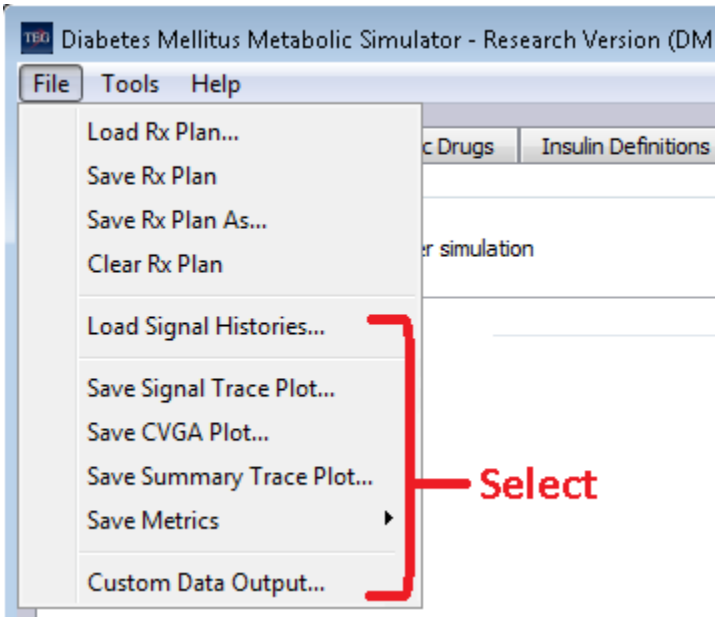


Figure 103: Saving and Loading Simulation Results

7.7.1 Loading Signal Histories

You may load the results of a previous simulation into DMMS.R for display. Choose the "Load Signal Histories…" option from the file menu to access a file dialog box and choose the signal history file to load. Loading a signal history file will cause all five types of data output displays (Signal Trace Plots, Summary Trace Plots, CVGA display, Individual Subject Metrics, and Population metrics) to display the loaded data.

For further detail on signal history files see Section 5.6.1.

Note that you cannot load data saved with versions of the DMMS.R prior to 1.1.0.

A status bar at the bottom of the DMMS.R main window shows information identifying the data currently being displayed, and the Rx plan that was used to generate the data that is currently displayed. Note that these status bar indications provide tooltips that show the directory holding the data and the directory in which the Rx Plan (xml file) resides. There is also an indication on the status bar showing whether the currently displayed data matches the most recent simulation run performed in the DMMS.

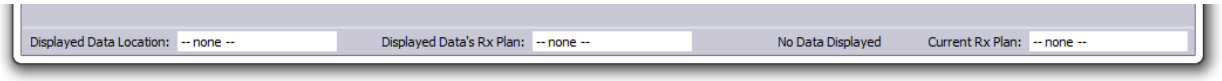


Figure 104: Displayed Data Status Bar

The displayed data status bar shows:

- Displayed Data Location – The folder containing the simulation data shown on the display
- Displayed Data's Rx Plan – Filename of xml file containing the configuration associated with the data currently shown in the output displays
- Current Rx Plan – The most recently loaded Rx plan

Each item displayed on the status bar has a tool tip that shows the full path to the file or folder shown.

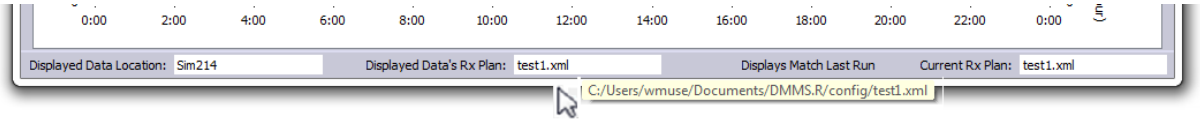


Figure 105: Displayed Data Status Tooltip

The indication "Displays Match Last Run" tells us that no new Rx plan or results data have been loaded since the last simulation was run. The indication "Displays May Not Match Last Run" means that an Rx plan has been loaded after the last simulation was run.

7.7.2 Exporting Signal Histories

On the Windows version of DMMS.R, you may save any simulation's signal history data to a MATLAB data file (.mat extension) using the "Custom Data Output…" option of the File menu.

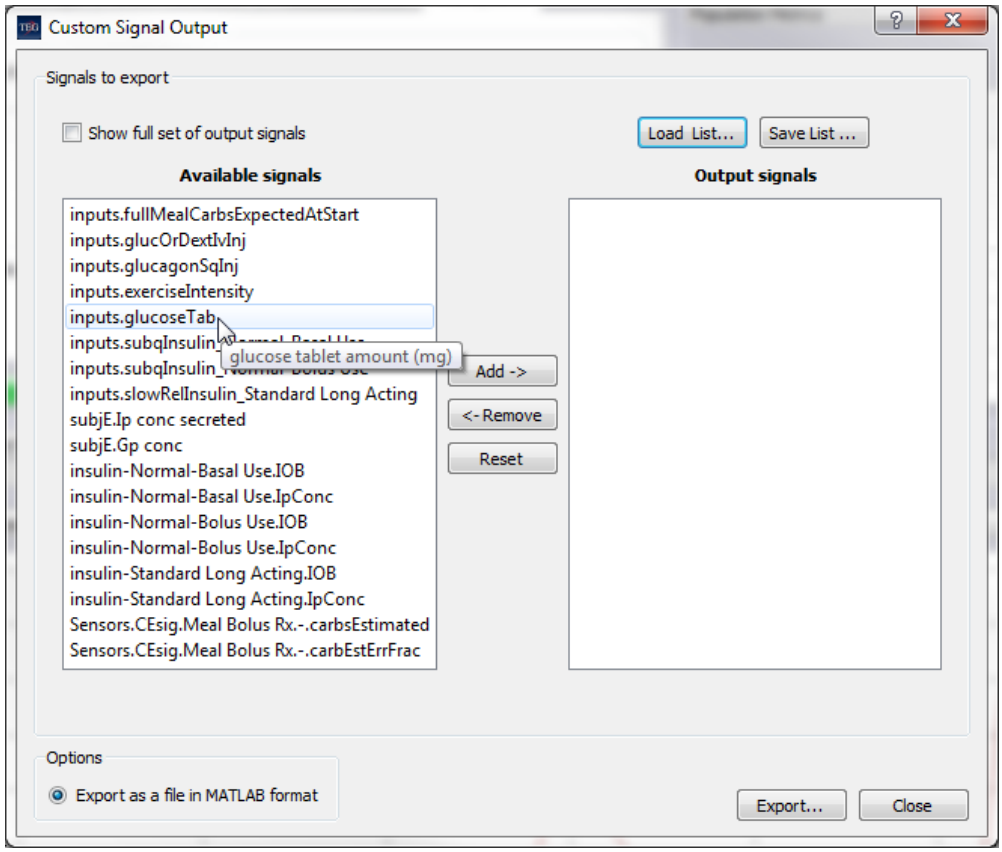


Figure 106: Exporting Signal Histories in MATLAB Format

The *Available signals* list shows the names of the available signals as described in Appendix A: Output Data Definition. If you hover over a signal name, a tooltip will display the "friendly" version of the signal's name along with its associated units. When the *Show full set of output signals* checkbox is not checked, the *Available signals* list will only show the most commonly used signals. If you check this box, the complete set will be displayed.

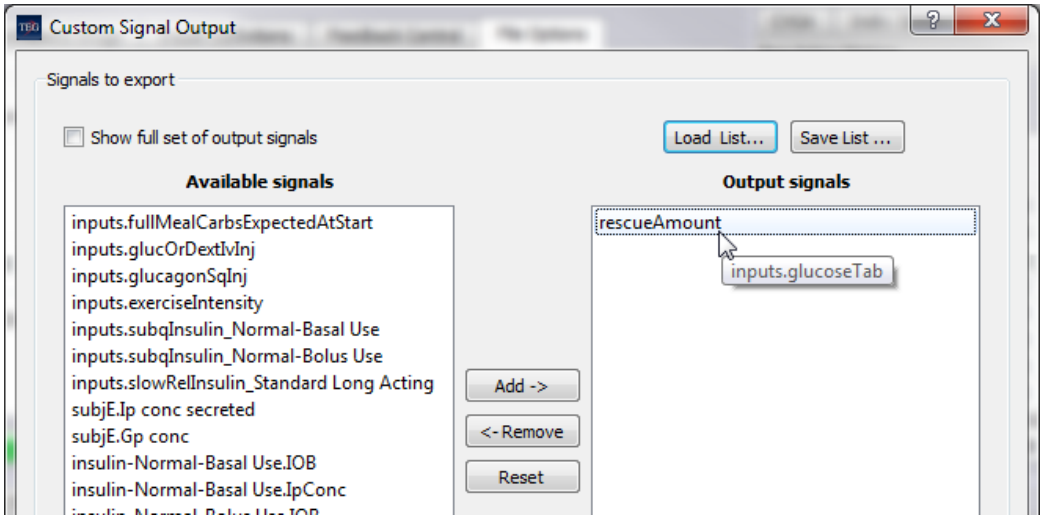


Figure 107: Addition of a signal into the Output signals list

You can select any signals for inclusion in an exported data file by moving them between the *Available signals* and the *Output signals* lists using the *Add* and *Remove* buttons. Once you have moved a signal to the *Output signals* list, you can rename it by double clicking on it. The name, as it appears in this list will correspond to the MATLAB name used in the exported .mat file. Note that the names appearing in the *Available signals* list are generally not legal MATLAB names, so when a signal name, as it appears in the *Available signals* list is first moved to the *Output signals* list it is automatically converted to a legal MATLAB name (e.g., by replacing dots with underscores, etc.). If you hover the cursor over the output signal, the DMMS.R will show the original name of the signal (as it appears in Appendix A: Output Data Definition).

Once you have completed your output signal list, you can press the *Export…* button to open a standard file save dialog box and specify the .mat file in which to save the signal histories. The exported .mat file will contain:

- For each subject:
  - Signal histories (minute-by-minute data) for each of the included signals
  - The subject's principal defining parameters (name, treatment parameters, etc.)
  - Survival information
- Principal simulation settings (e.g., duration and random seed settings)

The detailed format of the .mat file is provided in Appendix C: MATLAB Signal History Data Format.

If you have created an output signal list which you may like to use in the future, you may save this list to a comma separated value (csv) file via the *Save List…* button. This list will then be available for loading with the *Custom Signal Output* dialog and will also be available when you configure automatic saving of signal histories via the *File Options* tab of the Input Data tab control.

7.7.3 Plot Files

You may save Signal Trace Plots, CVGA Plots, or Summary Trace Plots to scalable vector graphics (.svg) files or a portable document format (pdf) files. Each plot will be saved in the same form in which it is displayed. See Sections 6.3.1 and 5.6.

- Summary Trace Plots
- CVGA display
- Individual Subject Metrics
- Population Metrics

7.7.4 Metrics Files

You may save individual subject or population metrics derived from simulations to pdf files. See Sections 6.3.1, 7.4 and 7.5.

To save metrics, go to the main menu bar and choose *File/Save Metrics*.

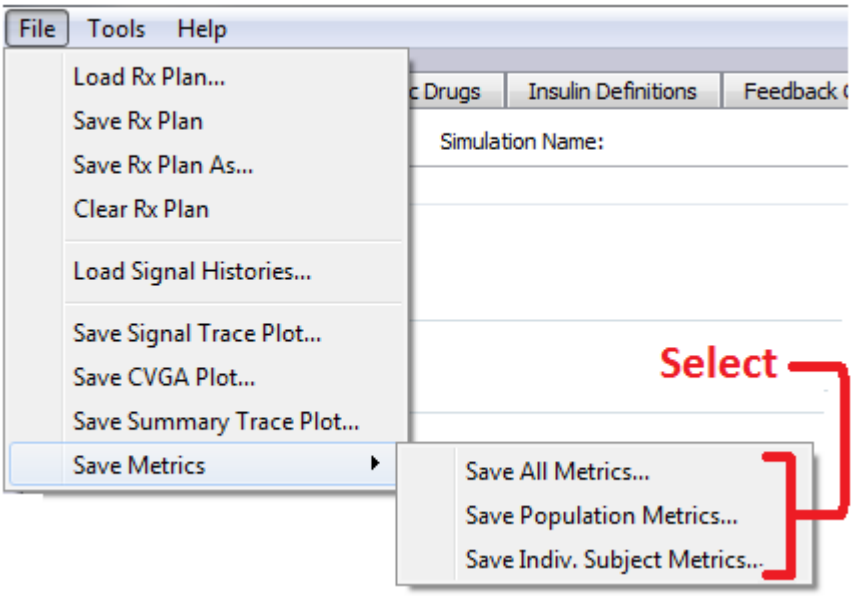


Figure 108: Saving Metrics

8 Configuration and Configuration Files

During installation, a few example configuration files will be written to your User File location. (See Section 5.2). In general, however, configuration files are created at run time, with a name, and possibly location, that you specify (See Section 6.3.1.1).

While configuration files may be modified by an experienced user using a text editor, it is strongly recommended that you use the method described in Section 5.4 should you need to change the content of these files.

So far, we have seen how we can manually enter treatment plan data into DMMS.R. Once the data for a treatment plan has been entered you can save them in a special file called a configuration file (or "config file"). You can then retrieve that data for use in another simulation.

Aside from being a significant time saver, this feature ensures that any number of treatment plans, initiated with the same config file, will be identical.

After loading a config file into a treatment plan, you are then free to change individual data items manually using the dialog controls on the displays. You may also edit the contents of an existing config file by loading it into the application, using the dialog controls to change the data, and then saving the file back to the same filename.

8.1 Loading, Saving, and Clearing Configurations

User access to configurations is through the first four menu options under *File* on the main menu bar:

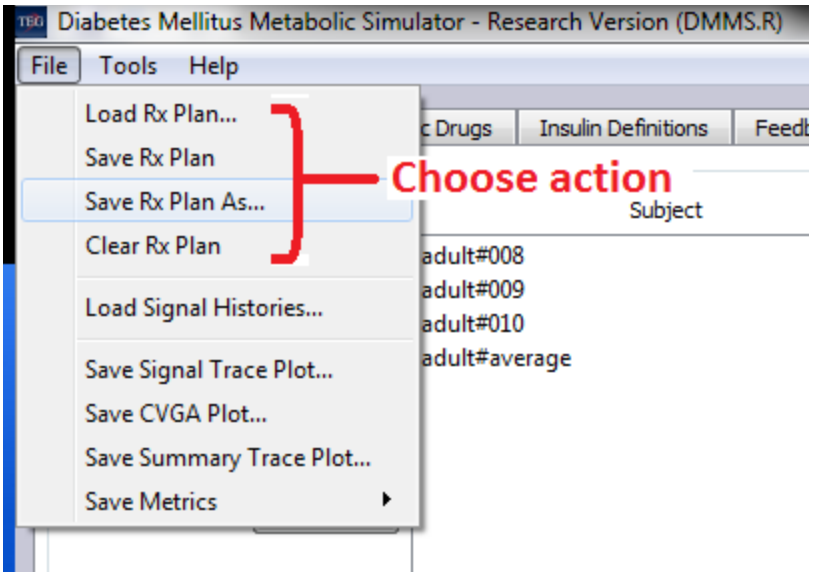


Figure 109: Rx Plan Actions

On the DMMS.R display, the abbreviation Rx is used, simply because it is shorter than "configuration", or even "config". In most uses here, "Rx", "configuration", and "config" are synonyms.

Options beginning with "Load" or "Save" refer to configuration files saved on disk. Choosing "Load Rx Plan…" or "Save Rx Plan As…" will result in the display of a standard Windows file dialog to allow you to specify the desired filename. By default, these files are written to the config folder of your user file location, but you can change that if needed (See Section 6.3.1.1).

"Save Rx Plan" will simply write the contents of the configuration back to the file from which it was loaded. If you have created the configuration *ab initio*, without loading any configuration files, the action of "Save Rx Plan" is the same as "Save Rx Plan As…".

"Clear Rx Plan" is a little different in that it does not deal with files in any way; it simply clears any configuration data you have created or loaded into DMMS.

Use caution when saving configuration data to an existing file. The contents of the original file will be lost making it difficult to reproduce any previous simulations based on it.

While the DMMS.R dialog-based interface is the usual choice for re-running a previously saved configuration, you may also use the command line for this purpose. See Appendix B: Running Simulations from the Command Line Interface.

9 Writing Plugin Elements

Plugin elements can be created for the DMMS to create custom sensors, control elements, and delivery elements for use in any simulation, these elements can be produced by writing either JavaScript or MATLAB functions.

Other languages may also be used by taking advantage of the JavaScript plugins' support for web service calling. These elements may even interact with each other and with the built-in elements of the DMMS.

The primary task in creating a plugin element is to write a function that defines the element's behavior on any given 1-minute iteration of a simulation. For sensors, this means calculating the sensor's single output value based on whatever "sensed" values of the subject's state effect the sensor (e.g., subcutaneous glucose concentration, plasma insulin concentration, etc.). For control elements, this means setting desired subject input values to be delivered (e.g., insulin, carbohydrates, etc.) based on such things as sensor signals, time information, etc. For delivery elements, defining the behavior means calculating how the delivery element modify the *desired* subject input values coming from the control elements to produce the *actual* subject input values to be delivered.

Each of the 3 types of plugin elements has its own well-defined interface with the DMMS, through which data is passed between the DMMS and the plugin on each iteration of a simulation. This interface is essentially the same in content, regardless of whether JavaScript or MATLAB is used. The differences are mainly in the mechanics of how the data is passed.

9.1 JavaScript Plugin Elements

When JavaScript is used to create a custom element (sensor, control element, or delivery element), the primary task (mentioned in the preceding section) is addressed by writing a *runIteration* function. This function will be called by the DMMS.R on every iteration of the simulation (where each iteration represents one minute of time in the simulated environment). The signature of this function (input and output parameters and return value) varies based on the type of element.

In addition to the runIteration function, each plugin element requires some simple supporting functions. These vary based on the element type. Their purpose is primarily to advertise attributes of the plugin that the DMMS.R needs to be aware of, and to provide a mechanism for initializing the plugin before the first

iteration of the simulation. The full set of functions representing any single plugin element must all reside in a single JavaScript file.

The ability to interpret the JavaScript code is entirely built into the DMMS.R, so no external JavaScript engine is required on the user's computer. The entire effort associated with creating the plugin involves creation of the one JavaScript file and configuration of the plugin element via the DMMS.R's GUI.

9.1.1 JavaScript Plugin Sensors

JavaScript Plugin support is available on the Windows OS version of DMMS.R.

The plugin sensor's *runIteration* function takes a set of numeric parameters, where each parameter represents the current value of an individual subject state signal. The number of such parameters is determined by the user based on the needs of the planned sensor. Most sensor implementations would only be sensitive to a single state signal (e.g., plasma glucose concentration for an SMBG sensor), but it is possible to make an individual sensor whose output is calculated from multiple state signals. In any event, the signal or signals to be used by the sensor must be configured as described in section 6.4.6.1.5.5. The order of the parameters in the *runIteration* function will match the order in which the state signals are entered in the configuration. The parameters can be given any names the user feels are appropriate, as the order of their appearance will entirely determine how they are populated with state signals by the DMMS.

The return value of the *runIteration* function will represent the output of the sensor. As for any other sensor configured into the DMMS.R, this value can be given a signal name via the general sensor configuration GUIs (see Section 6.4.6.1.4) and can be used by any control element.

Note that the *runIteration* function can make use of values that it may have assessed on previous iterations. In other words, it can maintain its own state. It is important to note that this state is guaranteed to be independent for each subject in the simulation, so a value established in a plugin sensor for subject #1 will never interfere with or overwrite a value established in a plugin sensor for subject #2.

In addition to the *runIteration* function, there are several supporting functions that should be implemented by the user, as described below:

The *initialize* function can be used to perform any initializations of variables that need to be performed prior to the first invocation of *runiteration*. This function is provided with the following parameters when invoked by the DMMS:

- popName: name of the population associated with the currently running subject (as it appears in the DMMS.R's "Subjects" tab – see section 6.4.2).
- subjName: name of the currently running subject (as it appears in the DMMS.R's "Subjects" tab – see section 6.4.2). Often, this (and the population name) may be useful in building the name of a file to which we want to record data (see section 9.1.6).
- simDuration: number of minutes in the full simulation

- configDir: default configuration file directory, as set in the DMMS.R's Tools/Settings menu (see section 6.3.1.1).
- baseResultsDir: default results directory, as set in the DMMS.R's Tools/Settings menu (see section 6.3.1.1).
- simName: the simulation name, as configured in the DMMS.R's "General" tab (see section 6.4.1).
- The *initializeRandomization* function is called by the DMMS.R at the start of the simulation (after the *initialize* function is called) and can be implemented to setup random number generators that may be used throughout the life of the plugin (typically by the *runIteration* function). Creation of the random number generators can be performed using the support object described in section 9.1.7. The DMMS.R passes the following parameters to the *initializeRandomization* function:useUniversalSeed: A Boolean parameter which will be true when the *Use random seed common to all elements* checkbox is checked. (See section 3.6)
- universalSeed: The numeric value which, when useUniversalSeed is true, should be used as the seed (or as a basis for the seed) in any random number generators created by this plugin. The value of this seed will be determined by the DMMS.R based on the *Base seed* value and *Make unique seed for each subject* checkbox setting (see section 3.6).

The *cleanup* function is called by the DMMS.R after the last iteration of the *runIteration* function. This function can be used to close files (see 9.1.6) that may need to remain open through all invocations of the *runIteration* function. It may also be used to add final statistical information to files written by the plugin.

. The *debugLoggingEnabled* function simply provides a mechanism for enabling or disabling logging by the plugin (see Section 9.1.4)

9.1.2 JavaScript Plugin Control Elements

The primary purpose of the plugin control element's *runIteration* function is to add contributions to the intended subject inputs. Most of the parameters in the long list passed to this function are included to supply all of the information that may potentially be needed by a control element to allow it to decide how it should add to or otherwise modify the set of intended subject inputs. These "modifiable" subject inputs are provided in the final parameter to the *runIteration* function.

The full set of parameters passed to the control element's *runIteration* function are as follows (further detail may be found in the samplePluginControlElement.js file installed with the DMMS.R):

- subjObject: An object containing properties representing some of the fundamental parameters defining the subject for whom this *runIteration* function is being called. This includes some treatment parameters (like the optimal carbohydrate ratio), which the plugin may need in dosing logic, and the subject's name, which can be used to support subject-specific logic or settings.
- sensorSigArray: An array of values representing the input signals configured for this control element (see Section 6.4.6.2.4). The order of the values in this array corresponds to the order as entered in the "Input Signal Config" dialog.

- nextMealObject: An object providing information about the next meal scheduled to be delivered to the subject, based on configuration of the "Meals" control element (see Section 6.4.6.2.5.8).
- nextExerciseObject: An object providing information about the next exercise session scheduled via the "Exercise" control element (see Section 6.4.6.2.5.6).
- timeObject: An object with information on the simulated data and time, and a measure of total elapsed time in the simulation.
- modelInputsToModObject: This object serves as both an input and an output of the *runIteration* function. Through it, the plugin is given the full set of intended subject inputs, as it has been built so far, via contributions from any of the control elements that have already run on the current iteration. Note that these may be other plugin control elements, or any of the built-in control elements. The current control element's *runIteration* function can supply its contributions to this object by adding to any of the individual properties' values, or by modifying these values in any way. You can specify the order of the control elements in a simulation (see Section 6.4.6.1.3), which allows control elements to be designed to modify subject inputs set by other elements that are configured to run earlier in the sequence. This could, for example, allow a custom "meal bolus modifier" plugin to reduce or remove a meal bolus created by the built-in Meal Bolus Rx element under special circumstances.
- prevModelInputs: This object contains a snapshot of the input properties of modelInputsToModObject from the previous iteration.
- prevControlOutputs: This object contains a snapshot of the output properties of the modelInputsToModObject from the previous iteration.
- inNamedSignals: This object is used by the DMMS.R to supply the plugin control element with all the output signals that other plugin control elements have contributed to the signal histories, as declared by those elements' *outSignalName* functions (see the description of this function later in this section). The inNamedSignals parameter's property names will match the names provided via the outSignalName functions and will only include properties associated with plugin control elements which ran earlier in the configured sequence (see section 6.4.6.1.3).
- outSigArray: This is an output-only parameter used to provide the values of each signal that this plugin control element intends to contribute to the signal histories (and to make available to other plugin control elements via their inNamedSignals parameter described above). This array should have several elements matching the value returned by this plugin's *numOutSignals* function (described later in this section), and the value at a given index position in this array should correspond to the signal name associated with that index via this plugin's *outSignalName* function (also described later in this section).
- outRunStopStatus: This object is an output parameter that can be used to force the simulation to stop and display a message giving the reason, along with an indication of whether the reason

represents an error. When this parameter is left unchanged by the *runIteration* function, it will always indicate that the simulation should not stop. This object has the following properties:

- stopRun: a Boolean to be set to true if the simulation is to be stopped
- error: an indication of whether the simulation is being stopped due to an error
- message: a string value providing the message to be displayed by the DMMS.R

Specific properties within each parameter of the runIteration function are documented in the samplePluginControlElement.js file that can be found in the DMMS.R's config directory after installation.

As for sensors, the *runIteration* function of a plugin control element can maintain its own state across iterations of the simulation, and this state is maintained independently for each subject in the simulation.

Plugin control elements support the same *initialize, initializeRandomization, cleanup,* and *debugLoggingEnabled* functions that were described for plugin sensors (see section 9.1.1).

Plugin control elements also require that you implement several additional supporting functions. Several of these are used exclusively for validation of the plugin's use in a configuration, to ensure that it does not require any features of the simulator that are not supported for any of the selected subjects. For example, none of the type 2 subjects have a defined carb ratio, so if a control element depends on knowing each subject's predefined carb ratio to calculate an insulin dosage, it cannot be included in a valid configuration that also includes type 2 subjects. In such a situation, the DMMS.R will identify the problem when you attempt to run the simulation. The following JavaScript functions must be implemented to support this type of validation:

- requiresSqInsulinSupport
- requiresSqGlucoseSuport
- requiresInsulinDosingSupport
- requiresCPeptideSupport
- requiresSqGlucagonSupport

Each of these functions simply returns a Boolean value.

Two additional functions support configuration of the sensor signals to be used by the control element. These are:

- numSignals: This function simply returns an integer indicating the number of sensor signals the *runIteration* function should expect to be provided in its *sensorSigArray* parameter.
- signalDescription: This function takes one parameter, named *signalIndex*, which will be the 0-based index of the signal for which a description is being requested. The function should return a

text description of the role that is expected to be filled by the signal in the sensorSigArray having the specified index.

These functions allow the developer of the plugin to tell the DMMS.R how many sensor input signals should be selected to be passed to the control element. They also let the developer provide text that will appear in the Input Signal Config dialog (see Section 6.4.6.2.4) to ensure the person configuring the DMMS.R can make appropriate choices for the signal that's passed to each element of the *runIteration* function's *sensorSigArray* parameter.

As an example, the following code from the samplePluginControlElement.js file results in the text seen in the left column of the Input Signal Config dialog shown in Figure 110.

```
function numSignals() { return 2; }
function signalDescription(signalIndex)
{
   switch(signalIndex)
   {
      case 0:
         return "SMBG measurement";
      case 1:
         return "CGM reading";
   }
}
```

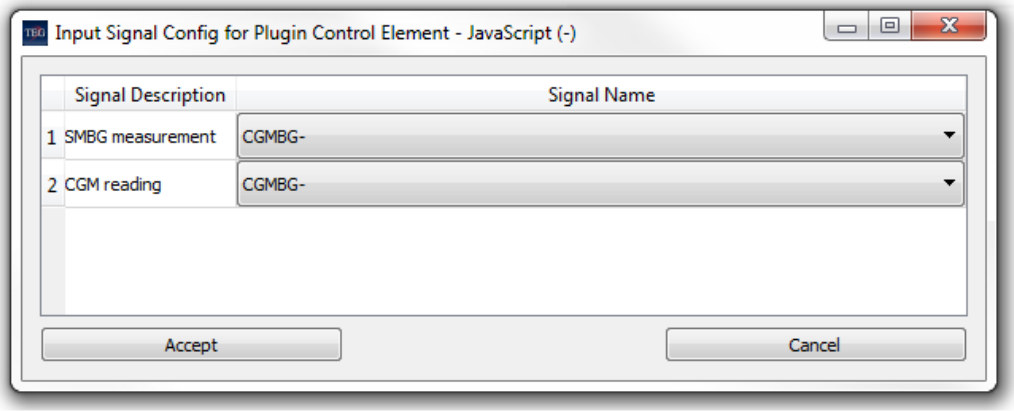


Figure 110: Input Signal Config Affected by a JavaScript Plugin

Two more optional functions can be used to let a plugin control element contribute signals to the recorded signal histories (see section 5.6.1 and Appendix A: Output Data Definition). Such signals can also be used to pass information from one JavaScript plugin control element to another, via the *inNamedSignals* and *outSigArray* parameters of the *runIteration* function (see documentation on that function at the beginning of this section).

The functions used for this are:

- numOutSignals: This function returns an integer indicating the number of signals produced by the plugin control element.
- outSignalName: This function is called by the DMMS.R to get the name of each of each signal produced by the plugin. The DMMS.R provides an integer *signalIndex* parameter when it makes this call, and the user's implementation of this function should provide the name it wants to associate with the signal that will be provided at the *signalIndex* location of the outSigArray produced by the plugin's *runIteration* function.

9.1.3 JavaScript Plugin Delivery Elements

The primary purpose of the plugin delivery element's *runIteration* function is to produce subject input values by modifying the intended subject inputs that ultimately resulted from all contributions made by all control elements. To support this, the runIteration function of a delivery element just takes one parameter, this being the *modelInputsToModObject*. Its content is identical to that for the plugin control elements, and the manner in which it is used by the delivery elements is also the same.

As for the other JavaScript plugin elements, the *runIteration* function of a plugin delivery element can maintain its own state across iterations of the simulation, and this state is maintained independently for each subject in the simulation.

Plugin delivery elements support the same *initialize, initializeRandomization, cleanup,* and *debugLoggingEnabled* functions that are supported by plugin sensors and control elements (see section 9.1.1).

9.1.4 JavaScript Debug Logging

On each invocation of the *runIteration* function, all three types of plugin elements can specify a block of text to be recorded to a debug log file. To enable this, a *debugLoggingEnabled* function, returning a Boolean value of true, must be included in the JavaScript file for the plugin. This function is evaluated by DMMS.R just once, at the start of the simulation, to determine whether to perform logging throughout the simulation run.

When logging is enabled, DMMS.R will log to an independent file for each plugin script and each subject. The log files will always be written to the same directory as that of the selected JavaScript file, and will have names following the pattern:

<js filename>_<subject name>.log

Here, <js filename> is the name of the JavaScript file defining the given plugin, and <subject name> is the full name of the subject, as it would appear in the "Subject" list in the "Available Subjects" portion of the "Subjects" configuration tab (see Section 6.4.2).

On each invocation of the plugin's *runIteration* function, any value assigned to the *debugLog* variable will be automatically appended to the log file, prefixed with an indication of the simulation day and time. If the

runIteration function completes with the *debugLog* variable empty, nothing will be written to the log file (not even the day and time).

If a log file of the name described above already exists at the time the simulation is begun, then the current log file will be renamed to have a "_previous" suffix (prior to the .log extension), overwriting any existing log file with this suffix.

9.1.5 JavaScript Web Service Invocation

Any JavaScript plugin can perform web service calls by making use of an object called *httpWebServiceInvoker* that the DMMS.R automatically makes available to its plugins. This object supports calls performed via either an HTTP GET or HTTP POST method. The invocation is always made synchronously (the JavaScript blocks until the web service call has completed or has timed out), and you may specify any timeout value for this.

The following methods are provided by the *httpWebServiceInvoker*:

Method	Parameters & Return Value
performGetRequest(url, timeout)	**url**: string holding the full URL of the web service endpoint **timeout**: integer, indicating the maximum time, in minutes, to wait for a response **return value**: Boolean, indicating success
performPostRequest(url, body, contentType, timeout)	**url**: string holding the full URL of the web service endpoint **body**: string holding the body of the HTTP request **contentType**: string holding the text to include in a "Content-Type" header within the HTTP request (e.g., "application/json" if the response body is to contain JSON data). If this string is empty, then no "Content-Type" header will be included in the request. **timeout**: integer, indicating the maximum time, in minutes, to wait for a response **return value**: Boolean, indicating success
responseBody()	**return value**: Text containing the full body of the HTTP response to the last performGetRequest or performPostRequest
timeout()	**return value**: Boolean indicating whether the last performGetRequest or performPostRequest resulted in a timeout
responseStatusCode()	**return value**: Number indicating the status code in the HTTP response for the last performGetRequest or performPostRequest
responseStatusDescrip()	**return value**: Text description of the status code (e.g., "NOT FOUND" for code 404) for the last performGetRequest or performPostRequest

resultDescription()	**return value**: Text description of the results of the last call to performGetRequest or performPostRequest. May be “Error,” “Success,” “Timeout,” or “Exception”

Table 7: httpWebServiceInvoker Methods

The following is a simple example of the runIteration function for a control element that calls a web service and simply logs all information regarding the response:

```
function runIteration(subjObject, sensorSigArray, nextMealObject, nextExerciseObject, timeObject,
modelInputsToModObject)
{
    success = httpWebServiceInvoker.performGetRequest("http://localhost:8080/testService" , 100);
    debugLog = "Success: " + success + "\n";
    debugLog += "Timeout: " + httpWebServiceInvoker.timeout() + "\n";
    debugLog += "Result Descrip: " + httpWebServiceInvoker.resultDescription() + "\n";
    debugLog += "Status Code: " + httpWebServiceInvoker.responseStatusCode() + "\n";
    debugLog += "Status Descrip: " + httpWebServiceInvoker.responseStatusDescrip() + "\n";
    debugLog += "Body: " + httpWebServiceInvoker.responseBody() + "\n";
}
```

9.1.6 JavaScript Plugin File Access

Any JavaScript plugin can read data from or write data to text files by making use of an object called *fileAccessor* that the DMMS.R automatically makes available to its plugins.

The following methods are provided by the *fileAccessor*:

Method	Parameters & Return Value
openForRead(filePath)	**filePath**: string holding the fully qualified path of the file to be opened. Note that the configDir or baseResultsDir parameters passed to the plugin element’s *initialize* function can be used to build the fully qualified path if desired. **return value**: Integer representing a file handle that can be used in all subsequent access of the file
openForWrite (filePath, truncate)	**filePath**: string holding the fully qualified path of the file to be opened. Note that the configDir or baseResultsDir parameters passed to the plugin element’s *initialize* function can be used to build the fully qualified path if desired. **truncate**: Boolean, indicating whether the file should be truncated (emptied) before opening. When set to false, everything written will be appended to the end of the file. **return value**: Integer representing a file handle that can be used in all subsequent access of the file
closeForRead(fileHandle)	**fileHandle**: Integer file handle, as provided by the openForRead function. **return value**: Boolean indicating whether the close operation was successful

closeForWrite(fileHandle)	**fileHandle**: Integer file handle, as provided by the openForWrite function. **return value**: Boolean indicating whether the close operation was successful
readLine(fileHandle)	**fileHandle**: Integer file handle, as provided by the openForRead function. **return value**: String value containing the line of text that was read from the file.
readFileAtEnd(fileHandle)	**fileHandle**: Integer file handle, as provided by the openForRead function. **return value**: Boolean indicating true when there are no more lines to be read in the file.
write(fileHandle, text)	**fileHandle**: Integer file handle, as provided by the openForWrite function. **text:** string value containing the txt to be written to the end of the file. **return value**: Boolean indicating whether the write operation was successful

Table 8: randomGenFactory Methods

The following is a simple example of an initialize function which performs all file access-related operations. It reads lines from one file and writes corresponding lines to an output file (after first writing the subject name to the file). Note that the fileAccessor is available in all functions of any type of JavaScript plugin:

```
function initialize(popName, subjName, simDuration, configDir, baseResultsDir, simName)
{
    var inFileHandle = fileAccessor.openForRead(configDir + "/pluginInput/testInput.txt");
    var outFileHandle = fileAccessor.openForWrite(configDir + "/pluginOutput/test.txt", false);
    fileAccessor.write(outFileHandle, subjName + "\n");
    while(!fileAccessor.readFileAtEnd(inFileHandle))
    {
        var lineFromFile = fileAccessor.readLine(inFileHandle);
        fileAccessor.write(outFileHandle, lineFromFile + " has been read\n");
    }
    fileAccessor.closeForRead(inFileHandle);
    fileAccessor.closeForWrite(outFileHandle);
}
```

9.1.7 JavaScript Plugin Random Number Generation Support

Any JavaScript plugin can create random number generators by using an object called *randomGenFactory* that the DMMS.R automatically makes available to its plugins. This factory object can create random number generators for uniform, normal, and beta distributions. The random number generator objects created in this way can be seeded with any value, including the “common seed” provided by the DMMS.R, based on the settings made in the “Randomization” group box of the “General” tab of the GUI, as described in section 3.7.

The following methods are provided by the *randomGenFactory*:

Method	Description, Parameters & Return Value
createNormDistribRndGen(stdDev, seed)	Provides a random number generator which can be used to obtain random values with a normal distribution. **stdDev**: numeric value indicating the standard deviation of the values provided by the random number generator. **seed**: numeric value used as the seed for the random number generator. **return value**: An object that can be used to generate random numbers with a normal distribution.
createUniformDistribRndGen (min, max, seed)	Provides a random number generator which can be used to obtain random variables with a uniform distribution. **min**: numeric value indicating the smallest random number that should be generated. **max**: numeric value indicating the largest random number that should be generated. **seed**: numeric value used as the seed for the random number generator. **return value**: An object that can be used to generate random numbers with a uniform distribution spanning the specified range.
createBetaDistribRndGen (alpha, beta, min, max, seed)	Provides a random number generator which can be used to obtain random variables with a beta distribution, transformed to span any desired range (maximum and minimum values). **alpha:** the alpha parameter of the beta distribution **beta:** the beta parameter of the beta distribution **min**: numeric value indicating the smallest random number that should be generated. **max**: numeric value indicating the largest random number that should be generated. **seed**: numeric value used as the seed for the random number generator. **return value**: An object that can be used to generate random numbers with the specified beta distribution, transformed to span the specified range.

Table 9: fileAccessor Methods

Once a random generator object is obtained from the *randomGenFactory*, the *nextValue* function of that object can be called (with no parameters) to obtain new random numbers.

The following is a simple example of an *initializeRandomization* function which creates a random number generator and acquires random numbers from it. The values obtained could be used by the runIteration function to determine sizes of meals to deliver to the subject, when to provide a bolus, and how large the bolus should be. Of course, the runIteration function itself could make use of the random number generators to obtain new random meal sizes, etc. each time a meal is to be given. This example is confined to the initializeRandomization function for the sake of compactness:

```
var breakfastSize;
var lunchSize;
var dinnerSize;
var cr;
var bolusDelay;
function initializeRandomization(useUniversalSeed, universalSeed)
{
    //The default value of our random seed will be 3.
    //This value will not be used if the DMMS.R is configured to provide a
    //universal seed to all elements.
    var seed = 3;
    if (useUniversalSeed)
    {
        seed = universalSeed;
    }
    var rndGenMealSize = randomGenFactory.createBetaDistribRndGen(3.0, 3.0, 30.0, 80.0, seed);
    breakfastSize = rndGenMealSize.nextValue();
    lunchSize = rndGenMealSize.nextValue();
    dinnerSize = rndGenMealSize.nextValue();
    var rndGenCR = randomGenFactory.createUniformDistribRndGen(10, 30, seed);
    cr = rndGenCR.nextValue();
    var rndGenBolusDelay = randomGenFactory.createNormDistribRndGen(0, 10, seed);
    bolusDelay = Math.max(0, rndGenBolusDelay.nextValue() + 20);
}
```

9.2 MATLAB Plugin Elements

The Windows version of DMMS.R provides support for MATLAB plugin elements.

The DMMS uses the MATLAB Engine API to run a MATLAB process in which your plugin code can be executed. To use this, you must have a supported version of MATLAB installed on their computer. The DMMS.R has been tested with, and fully supports, MATLAB versions 2014b through 2016b for this. If multiple versions of MATLAB are installed, the DMMS.R will automatically discover the latest version and use this for the MATLAB process.

When MATLAB is used to create a custom element (sensor, control element, or delivery element), the primary task is addressed by writing a single function. The name of this function must match the name of its file with the ".m" extension omitted (as is the convention in MATLAB functions). This function will be called by the DMMS.R on every iteration of the simulation (where each iteration represents one minute of time in the simulated environment). The signature of this function (input and output parameters and return value) varies based on the type of element.

In the case of control elements, one additional function (in a separate file) is required to inform the DMMS.R of certain attributes of the plugin, but for other element types, only the plugin's main function is required. The entire effort associated with development of a MATLAB plugin involves creation of one or two MATLAB functions and configuration of the plugin elements via the DMMS.R's GUI.

9.2.1 MATLAB Plugin Sensors

The plugin sensor's MATLAB function takes a set of numeric parameters, where each parameter represents the current value of an individual subject state signal. The number of such parameters is determined by the user based on the needs of the planned sensor. Most sensor implementations would only be sensitive to a single state signal (e.g., plasma glucose concentration for an SMBG sensor), but it is possible to make an individual sensor whose output is calculated from multiple state signals. In any event, the signal or signals to be used by the sensor must be configured as described in section 6.4.6.1.5.5. The order of the parameters in the plugin's function will match the order in which the state signals are entered in the configuration. The parameters can be given any names you feel are appropriate, as the order of their appearance will entirely determine how they are populated with state signals by the DMMS.

The return value of the plugin's function will represent the output of the sensor. As for any other sensor configured into the DMMS.R, this value can be given a signal name via the general sensor configuration GUIs (see Section 6.4.6.1.4) and can be used by any control element.

Note that, by using MATLAB's persistent variables, the plugin's function can make use of values that it may have assessed on previous iterations. In other words, it can maintain its own state. This state is guaranteed to be independent for each subject in the simulation, so a value established in a plugin sensor for subject #1 will never interfere with or overwrite a value established in that plugin sensor for subject #2.

Persistent variables can also be used to identify the first iteration of the plugin function, by relying on the fact that MATLAB's *isempty* function will return *true* for an uninitialized persistent variable. This mechanism can be used to perform initialization processing for the plugin. Note that this approach takes the place of the *initialize* function used for JavaScript plugins.

The MATLAB sensor plugins also support logging any text to a debug log file (see Section 9.2.4).

All MATLAB plugin delivery elements also can access several configured simulation settings (see section 9.2.5) to support:

- Random number generation
- Identification of the subject and simulation
- Expected duration of the simulation

9.2.2 MATLAB Plugin Control Elements

The MATLAB plugin control element is implemented via two MATLAB functions, each in its own file. One addresses the plugin's primary task of updating the intended subject inputs on each iteration. The other, a "setup" function, provides the DMMS.R with various attributes of the plugin, used in validating configurations and in supporting the assignment of sensor signals to be provided by DMMS.R to the plugin's primary function on each iteration.

The primary function of the control element can be given any name your desire (e.g., pluginControlElement), but must be placed in a file of the same name, with a ".m" extension. This is the file that you should have selected through the DMMS.R GUI when configuring the plugin. The setup function

must have a name identical to that of the primary function, but with a "_setup" suffix (e.g., pluginControlElement_setup). It should also be in a file whose name matches the function name, but with a ".m" extension.

The specific job of the primary function is to add contributions to the intended subject inputs. Most of the parameters in the long list passed to this function are intended to supply all of the information that may potentially be needed by a control element to allow it to decide how it should add to or otherwise modify the set of intended subject inputs. The set of these inputs, based on contributions from any control elements that ran prior to this plugin on the current simulation iteration, are provided in the modelInputsToModObject parameter. The 1st return parameter of this function must also be named modelInputsToModObject, and, upon returning from this function, its fields' values should equal the values provided via this input parameter plus any contributions this plugin should make.

Another parameter, runStopStatusToModObject is handled similarly to the emodelInputsToModObject. It is provided as an input parameter, and a return parameter of the same name is used to indicate whether the simulation should be stopped and why. When left unmodified, the value will always indicate that the simulation should not stop.

The full set of parameters passed to the control element's primary function are as follows:

- subjObject: A struct containing fields representing some of the fundamental parameters defining the subject for whom this *runIteration* function is being called. This includes some treatment parameters (like the optimal carbohydrate ratio), which the plugin may need in dosing logic, and the subject's name, which can be used to support subject-specific logic or settings.

- sensorSigArray: An array of values representing the input signals configured for this control element (see Section 6.4.6.2.4). The order of the values in this array corresponds to the order as entered in the "Input Signal Config" dialog.

- nextMealObject: A struct providing information about the next meal scheduled to be delivered to the subject, based on configuration of the "Meals" control element (see Section 6.4.6.2.5.8).

- nextExerciseObject: An struct providing information about the next exercise session scheduled via the "Exercise" control element (see Section 6.4.6.2.5.6).

- timeObject: An struct with information on the simulated data and time, and a measure of total elapsed time in the simulation.

- modelInputsToModObject: A struct providing the plugin with the full set of intended subject inputs, as it has been built so far, via contributions from any of the control elements that have already run on the current iteration. Note that these may be other plugin control elements, or any of the built-in control elements. The current control element's primary function can supply its contributions to these intended subject inputs by modifying the values in modelInputsToModObject before providing it as the function's return value. You may specify the order of the control elements in a simulation (see Section

6.4.6.2.3). This allows control elements to be designed to modify subject inputs set by other elements that are configured to run earlier in the sequence. This could, for example, allow a custom "meal bolus modifier" plugin to reduce or remove a meal bolus created by the built-in Meal Bolus Rx element under special circumstances.

- runStopStatusToModObject: A struct that the plugin my use to indicate whether the simulation should stop, and, if so, why. It has the following fields:
    - stopRun: a Boolean to be set to true if the simulation is to be stopped
    - error: an indication of whether the simulation is being stopped due to an error
    - message: a string value providing the message to be displayed by the DMMS.R

Specific fields within each parameter of the plugin control element's primary function are documented in the samplePluginControlElement_matlab.m file that can be found in the DMMS.R's config directory after installation.

As for sensors, the primary function of a plugin control element can maintain its own state across iterations of the simulation, and this state is maintained independently for each subject in the simulation.

The primary function of a plugin control element can handle initialization and debug logging in the same manners as were described for plugin sensors. This function can also access configured simulation settings associated with randomization, names (of the simulation and subjects), and the simulation duration (see section 9.2.5).

Plugin control elements' setup function takes no parameters and returns a single structure with fields that inform the DMMS.R of various characteristics of the plugin. Several of these are used exclusively for validation of the plugin's use in a configuration, to ensure that it does not require any features of the simulator that are not supported for any of the selected subjects. For example, none of the Type 2 subjects have a defined carb ratio, so if a control element depends on knowing each subject's predefined carb ratio to calculate an insulin dosage, it cannot be included in a valid configuration that also includes Type 2 subjects. In such a situation, the DMMS.R will identify the problem when you attempt to run the simulation. The following Boolean fields must be included in the struct returned by the setup function:

- requiresSqInsulinSupport
- requiresSqGlucoseSuport
- requiresInsulinDosingSupport
- requiresCPeptideSupport
- requiresSqGlucagonSupport

Two additional fields must also be included in the returned struct, in order to support the configuration of the sensor signals to be used by the control element. These are:

- numSignals: This field holds an integer indicating the number of sensor signals the primary function should expect to be provided in its *sensorSigArray* parameter.
- signalDescription: This field is an array where each element holds a text description for sensor signal found in the corresponding element of the *sensorSigArray*. These descriptions should indicate the role that is expected to be filled in the control element by the given signal.

These fields allow the developer of the plugin to tell the DMMS.R how many sensor input signals should be selected to be passed to the control element. They also let the developer provide text that will appear in the Input Signal Config dialog (see Section 6.4.6.2.4) to ensure the person configuring the DMMS.R can make appropriate choices for the signal that's passed to each element of the primary function's *sensorSigArray* parameter.

As an example, the following code from the samplePluginControlElement_matlab_setup.m file results in the text seen in the left column of the Input Signal Config dialog shown in Figure 111**Error! Reference source not found.**.

```
function output = samplePluginControlElement_matlab_setup
    output.requiresSqInsulinSupport = true;
    output.requiresSqGlucoseSupport = false;
    output.requiresInsulinDosingSupport = false;
    output.requiresCPeptideSupport = false;
    output.requiresSqGlucagonSupport = false;
    output.numSignals = 2;
    output.signalDescription = {'SMBG2013','My CGM'};
```

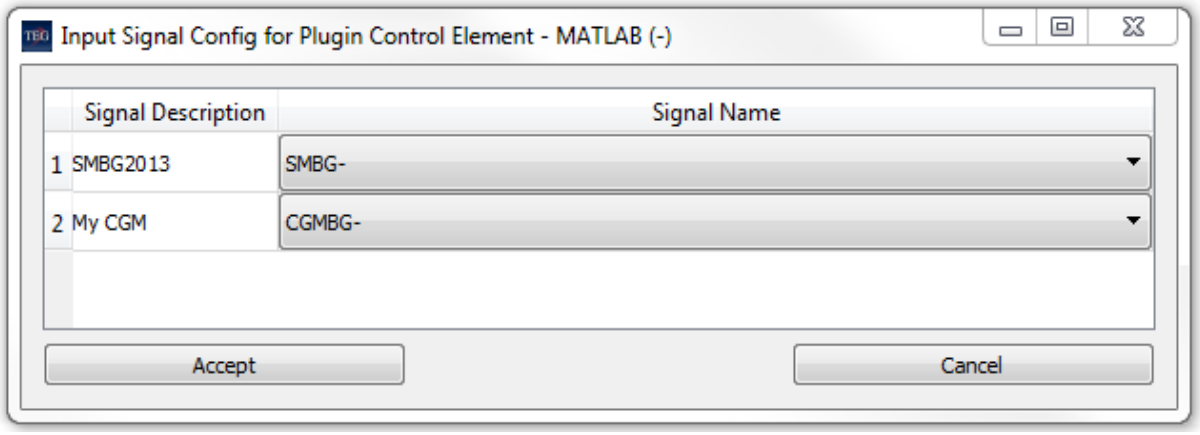


Figure 111: Input Signal Config Affected by a MATLAB Plugin

9.2.3 MATLAB Plugin Delivery Elements

The main purpose of the plugin delivery element's single MATLAB function is to produce subject input values by modifying the intended subject inputs that ultimately resulted from all contributions made by all control elements. To support this, the MATLAB function of a delivery element just takes one parameter, this being the *modelInputsToModObject*. Its content is identical to that for the plugin control elements, and the manner in which it is used by the delivery elements is also the same. The delivery element's MATLAB function's return value is the modified version of the modelInputsToModObject, just as was the case for the MATLAB plugin control element.

As for the other MATLAB plugin elements, the MATLAB function of a plugin delivery element can maintain its own state across iterations of the simulation, and this state is maintained independently for each subject in the simulation.

The MATLAB function of a plugin delivery element can handle initialization and debug logging in the same manners as were described for plugin sensors (see section 9.2.4). This function can also access configured simulation settings associated with randomization, names (of the simulation and subjects), and the simulation duration (see section 9.2.5).

9.2.4 MATLAB Debug Logging

On each invocation of the primary function for any control element, or of the single MATLAB function of a sensor or delivery element, a block of text can be designated to be recorded to a debug log file. To enable this, a global variable named debugLog must be declared.

When logging is enabled, the DMMS.R will log to an independent file for each plugin element and each subject. The log files will always be written to the same directory as that of the applicable MATLAB file, and will have names following the pattern:

<MATLAB filename>.matlab_<subject name>.log

Here, <MATLAB filename> is the extensionless name of the MATLAB file defining the given plugin, and <subject name> is the full name of the subject, as it would appear in the "Subject" list in the "Available Subjects" portion of the "Subjects" configuration tab (see Section 6.4.2).

On each invocation of the plugin's function, any value assigned to the *debugLog* global variable will be automatically appended to the log file, prefixed with an indication of the simulation day and time. If this function completes with the *debugLog* variable empty, nothing will be written to the log file (not even the day and time).

If a log file of the name described above already exists at the time the simulation is begun, then the current log file will be renamed to have a "_previous" suffix (prior to the .log extension), overwriting an existing log file with this suffix, if one already existed.

9.2.5 MATLAB Plugins' Access to Simulation Settings

The DMMS.R provides all MATLAB plugins with access to a global getsimsettings() function which provides information about the current simulation and about the subject currently being simulated. This function may be called with no parameters to obtain a 1x1 structure with the following fields:

- simDurationMinutes: the planned duration of the simulation, in minutes
- subjectName: Subject name, as displayed in the DMMS.R's subject selection tab (e.g., adult#001)
- populationName: name of the population to which this subject belongs, as displayed in the DMMS.R's subject selection tab (e.g., Type1 Adult).
- useUniversalRandomSeed: boolean, with a value of true if the "Use random seed common to all elements" setting is checked on the DMMS.R's "general" tab, indicating that the same initial seed should be used in random number generation across all sensors, control elements, and delive ry elements.
- universalRandomSeed: When "useUniversalRandomSeed" is true, the value of the seed to be used with all elements that use random number generation. When useUniversalRandomSeed is true, this value should be used by the MATLAB plugin to initialize random number generation (typically done on the first iteration of the primary function).
- simName: The name of the simulation, as configured in the DMMS.R's "General" tab.

Alternatively, the getsimsettings() function can be called with a single parameter, representing the name of a desired field, to obtain the value of this field.

10 Walkthrough 1: Meals and Meal Boluses

This section offers a brief tutorial on using DMMS.R to demonstrate the effects of carbohydrate intake and insulin boluses. First, we will setup a very simple simulation: a single Type 1 subject, 3 meals, no insulin therapy. Then we will repeat the simulation adding meal boluses.

To begin, bring up the DMMS.R application and open the *General* tab. The dialog will show a tab control with a text box for the simulation name. Enter "MealsDemo" in this box. This will provide a unique name for the output files, to make them easier to access later.

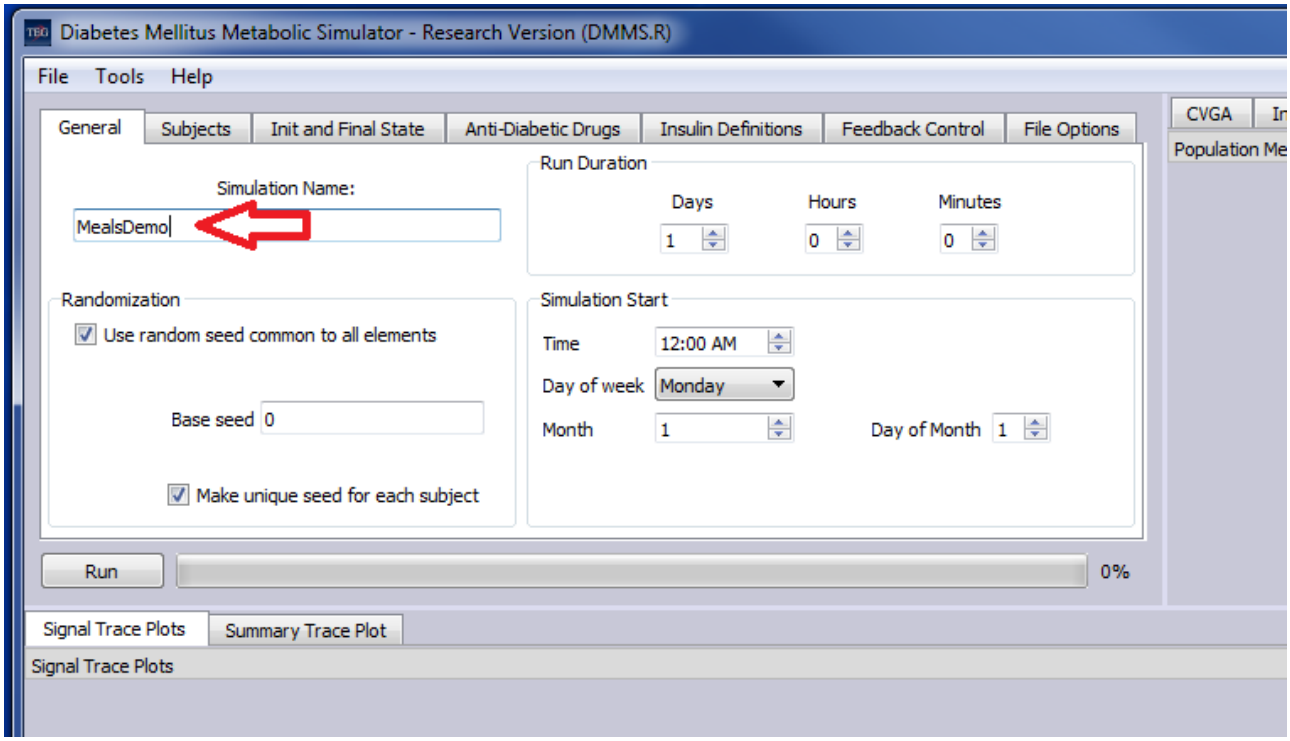


Figure 112: Walkthrough 1 -- Naming the Simulation

Accept the default values for everything else. Now open the *Subjects* tab.

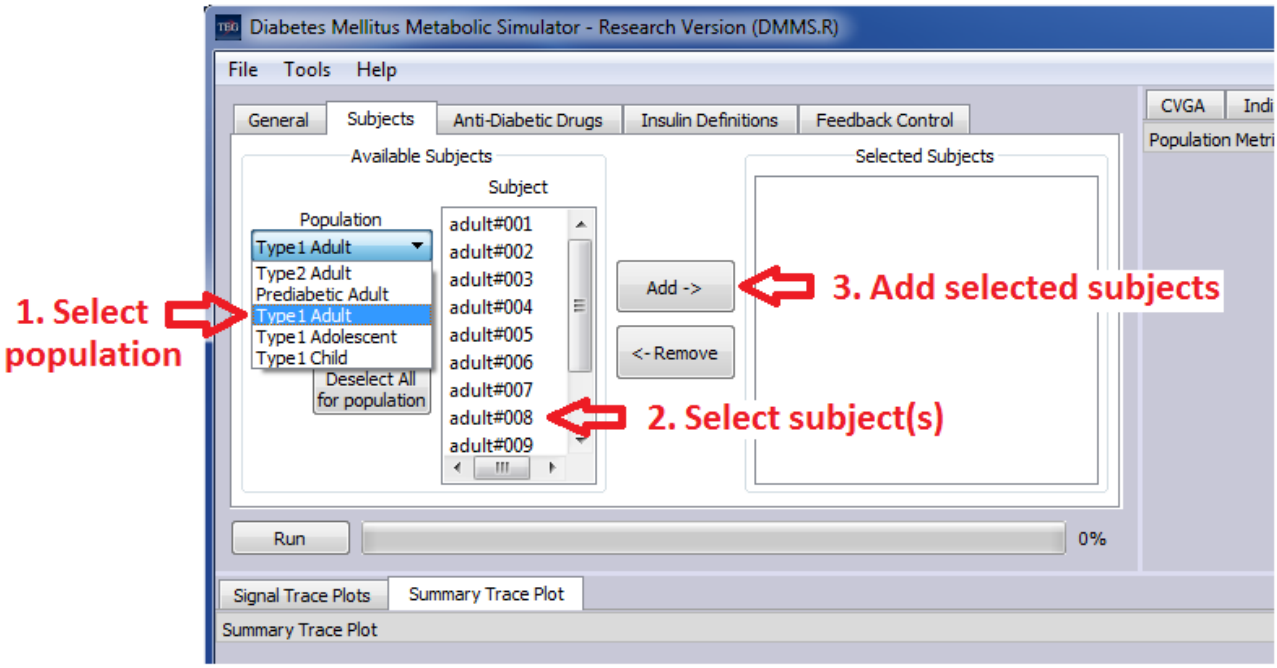


Figure 113: Walkthrough 1 -- Selecting the Population and Subject

Open the *Population* drop down and select Type 1 Adult. In the *Subject* list box, adult#001 will be selected by default. Press the *Add* button to move this subject from the *Subject* list box to the *Selected Subjects* list box.

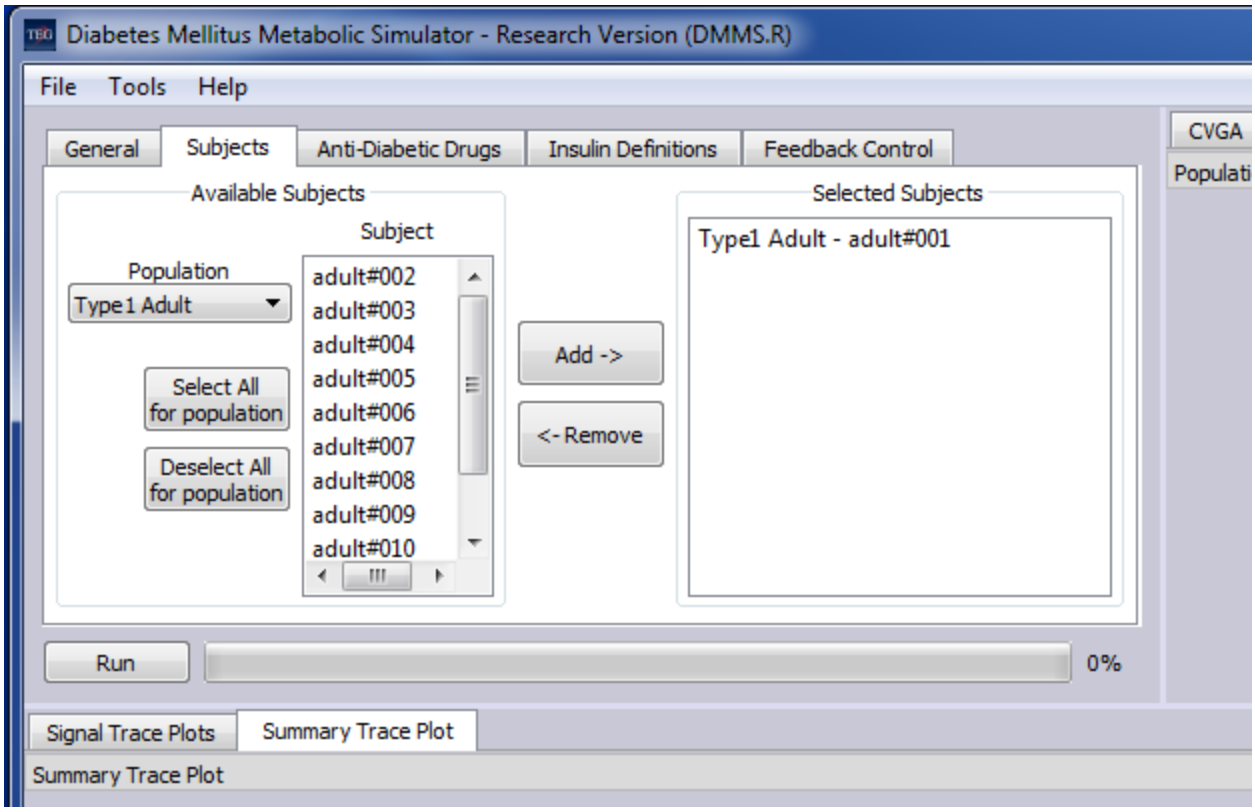


Figure 114: Walkthrough 1 -- Subject Selected

We will not be administering any pharmaceuticals to this Type 1 subject so we can skip the *Anti-Diabetic Drugs.* We will include insulin therapy later, but for now we will just define meals.

Open the *Feedback Control* tab, and then the *Control* subtab. Select *Meals* from the *Available Control Types* and press *Add*.

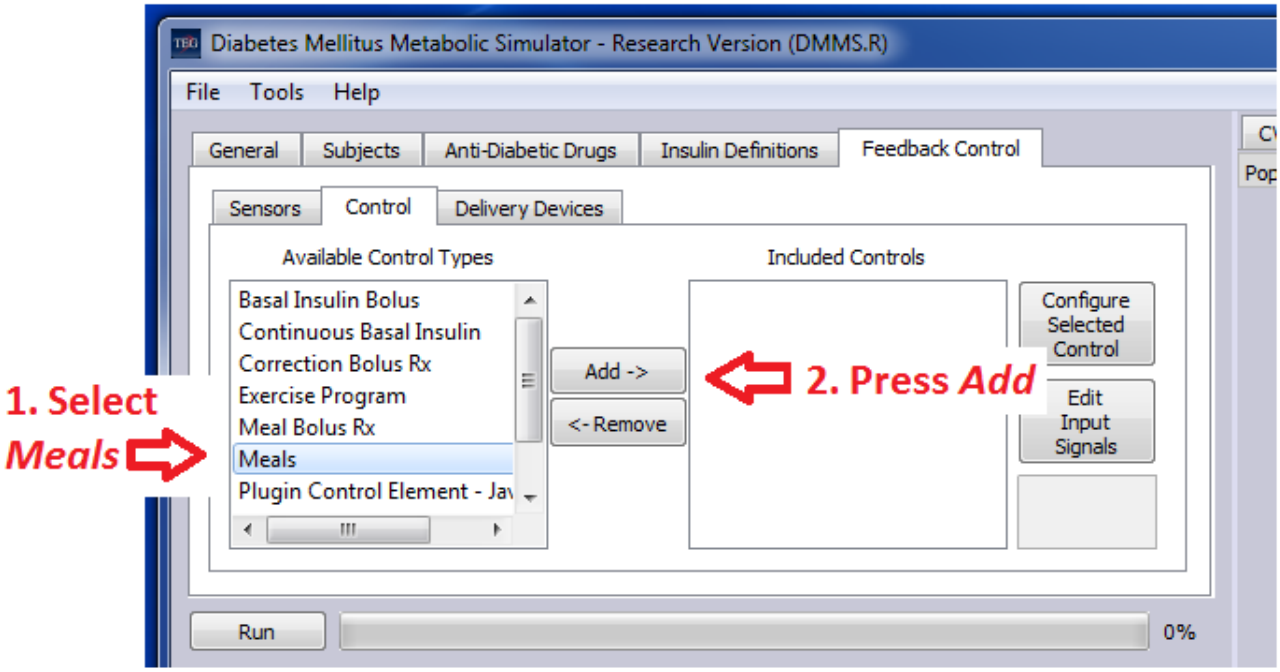


Figure 115: Walkthrough 1 -- Adding the Meals Control

Press the *Configure Selected Control* button. (Since the *Meals* control is the only one on the list, it is already selected.)

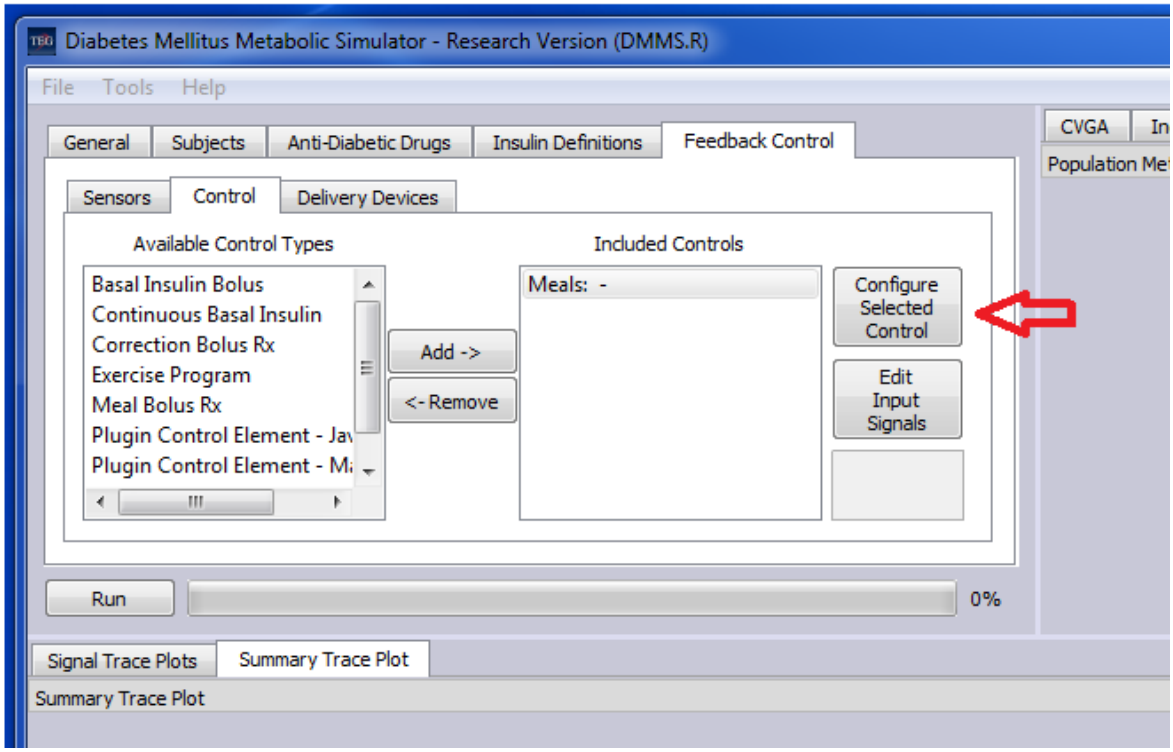


Figure 116: Walkthrough 1 -- Configure Selected Control

The following dialog will appear:

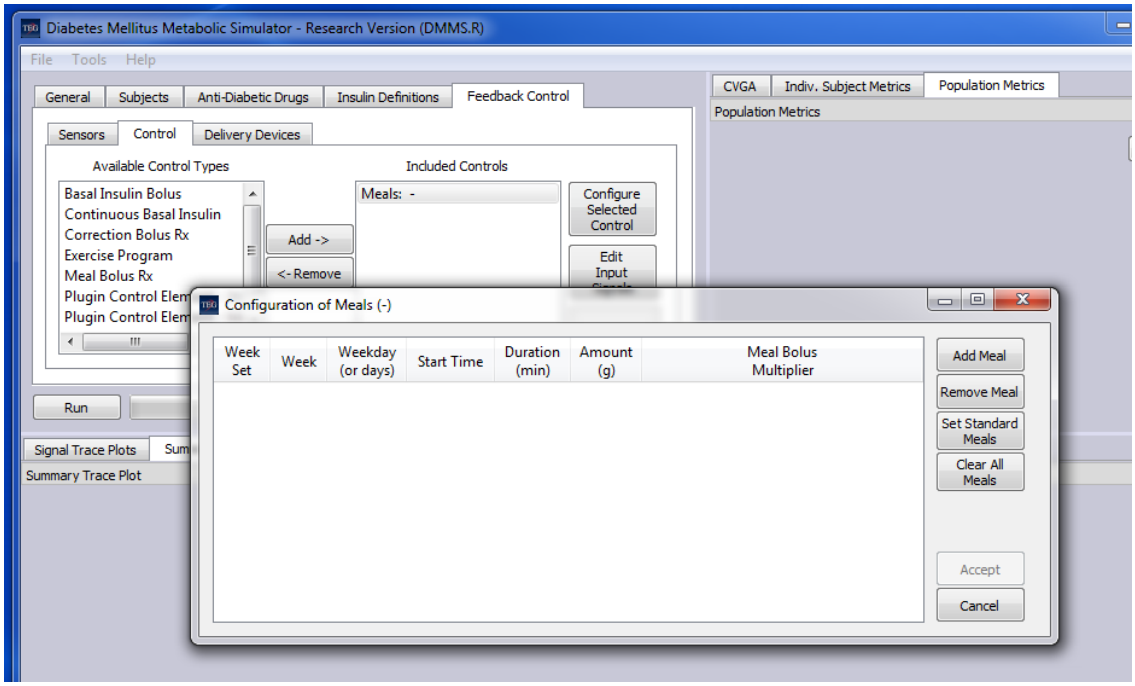


Figure 117: Walkthrough 1 -- Meal Configuration Dialog

Press the *Add Meals* button three times to add three meals. By default, the first meal is at 6:00 am and additional meals are spaced an hour apart.

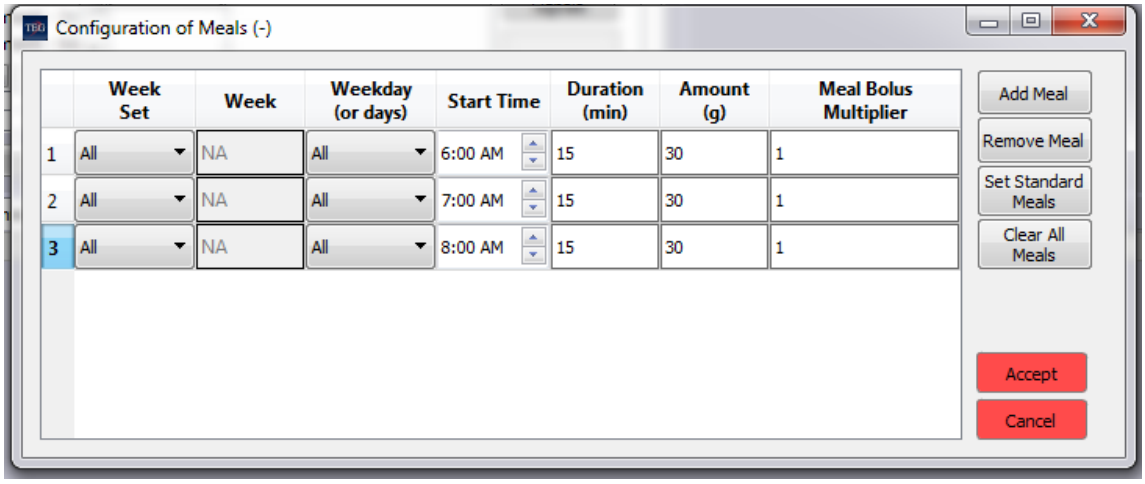


Figure 118: Walkthrough 1 -- Add Meals

Change the times of the second two meals to correspond to typical times for lunch and dinner (12:00 and 6:00 pm). We will assume our subject is an early riser and eats breakfast at 6:00 am.

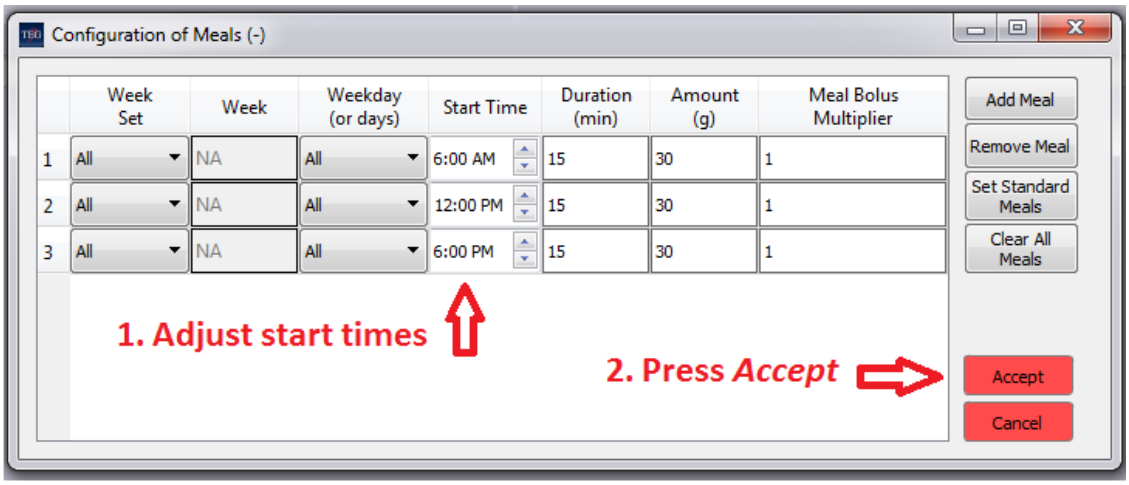


Figure 119: Walkthrough 1 -- Configure Meal Details

That is all the input we need at this point. Go ahead and press the *Run* button.

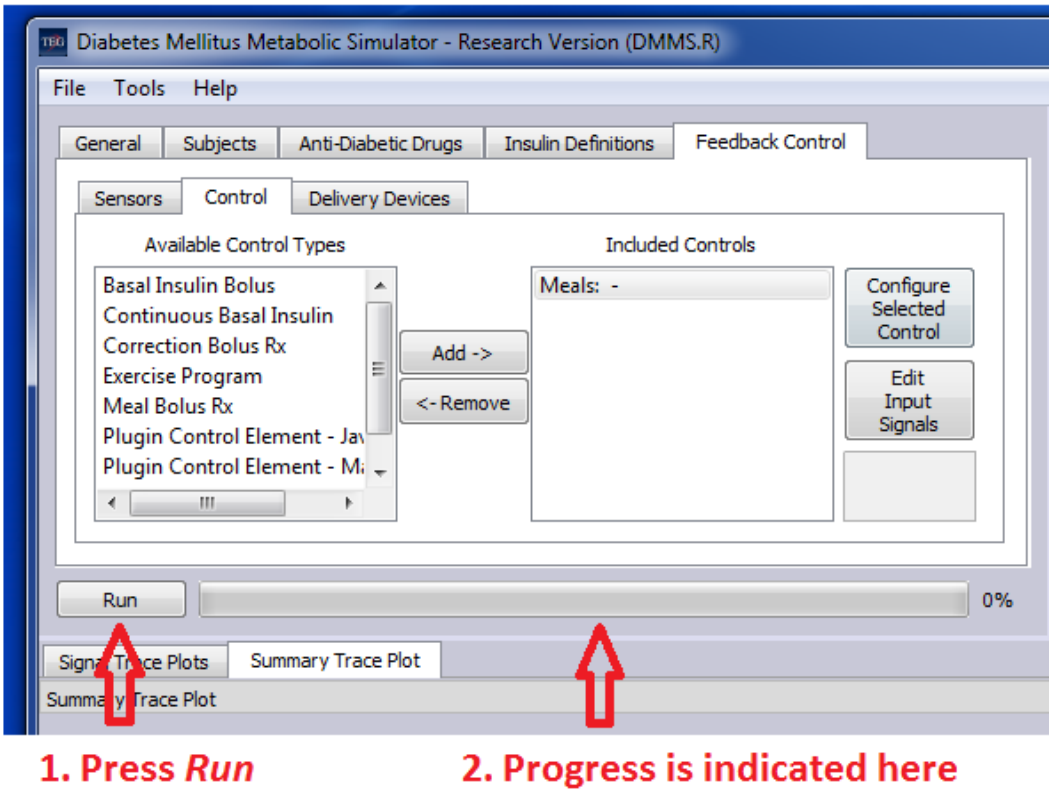


Figure 120: Walkthrough 1 -- Run the Simulation

When the simulation is complete (it should only take a few seconds) the signal trace plot will be filled out:

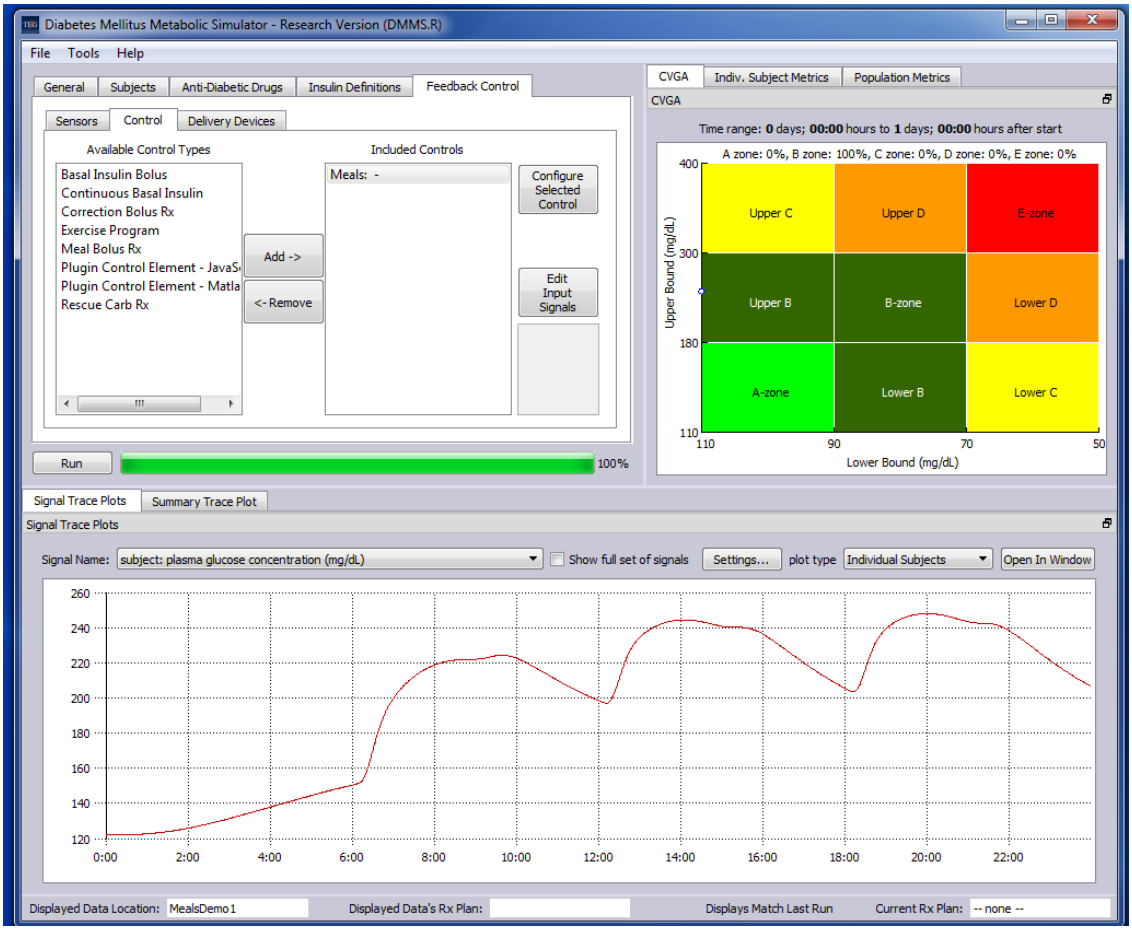
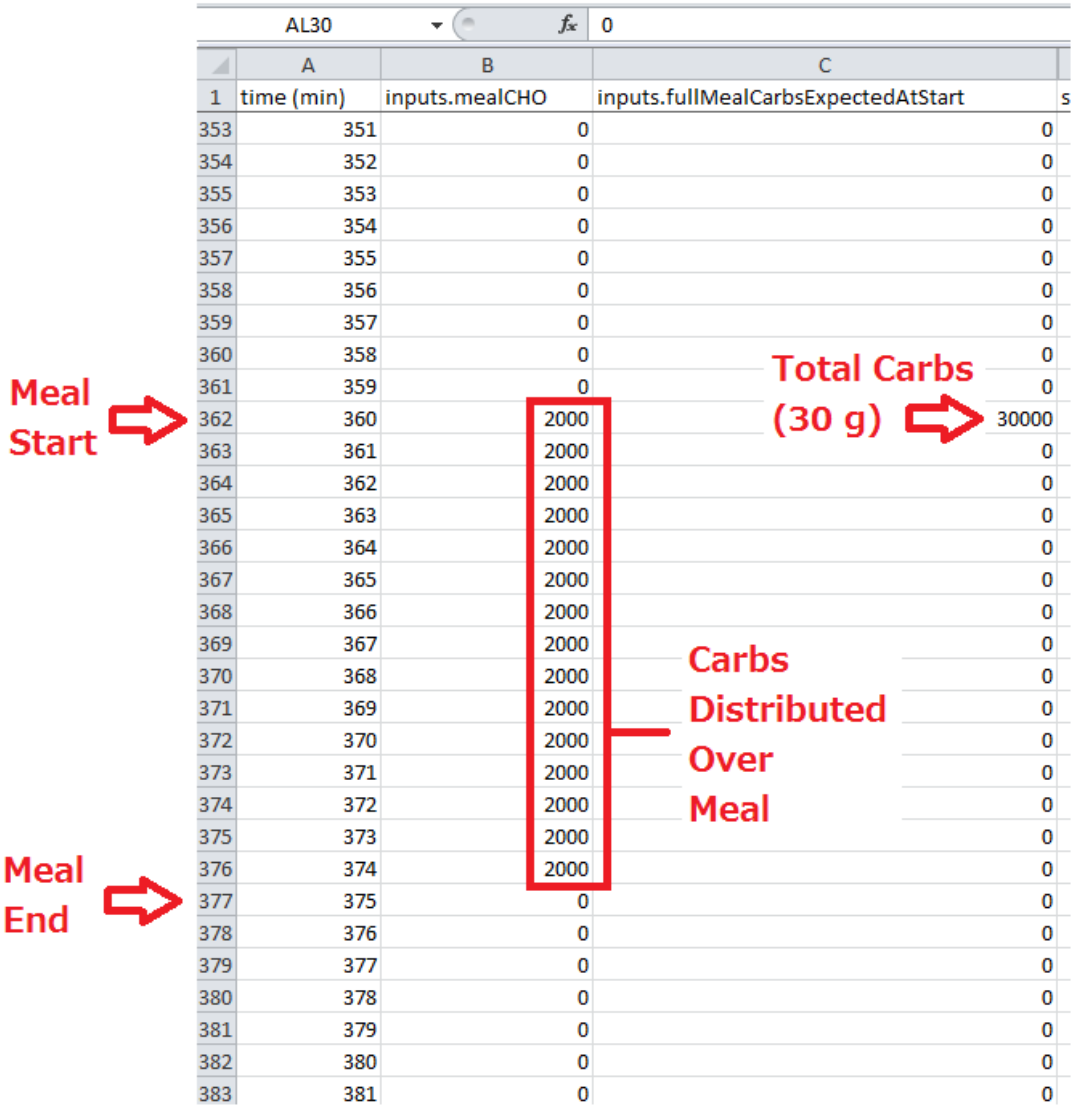


Figure 121: Walkthrough 1 -- Simulation Results Display

Note that the subject's blood glucose concentration rises above 240 mg/dL in the postprandial periods after lunch and dinner.

We can see these data in more detail in the results file generated. Go to your user files location and navigate down to …/DMMS.R/results/MealsDemo.

There should be two csv files in the folder. Open signalHistory.Type1 Adult.adult#001.csv. The left-most column indicates the number of minutes elapsed in the simulation. Scroll down to 360. Recall that our simulation started at 12:00 am and that the first meal was scheduled for 6:00 am (360 minutes). The first 3 columns should look like this:



	A	B	C	
1	time (min)	inputs.mealCHO	inputs.fullMealCarbsExpectedAtStart	s
353	351	0	0	
354	352	0	0	
355	353	0	0	
356	354	0	0	
357	355	0	0	
358	356	0	0	
359	357	0	0	
360	358	0	0	
361	359	0	0	
362	360	2000	30000	
363	361	2000	0	
364	362	2000	0	
365	363	2000	0	
366	364	2000	0	
367	365	2000	0	
368	366	2000	0	
369	367	2000	0	
370	368	2000	0	
371	369	2000	0	
372	370	2000	0	
373	371	2000	0	
374	372	2000	0	
375	373	2000	0	
376	374	2000	0	
377	375	0	0	
378	376	0	0	
379	377	0	0	
380	378	0	0	
381	379	0	0	
382	380	0	0	
383	381	0	0	

Figure 122: Walkthrough 1 -- Simulation Results File, Meal Data

Note that we have not defined any treatment plan; the data merely confirm that a 30-gram meal was consumed over 15 minutes starting at 6:00 am.

To check the subject's blood glucose levels, sort (descending), column AF (subjE.Gp.conc). The largest value should be 249.72 mg/dL.

AE	AF	AG
bjE.I app rate	subjE.Gp conc	subjE.Gl
0	249.72	
0	249.719	
0	249.717	
0	249.714	
0	249.711	
0	249.705	
0	249.701	
0	249.693	
0	249.687	
0	249.677	
0	249.669	
0	249.658	
0	249.647	

Figure 123: Walkthrough 1 -- Simulation Results File, Glucose Concentrations (sorted)

Now we will run another simulation, adding meal bolus therapy. To preserve the results of our previous work, we will give this simulation a different name: MealBolusesDemo.

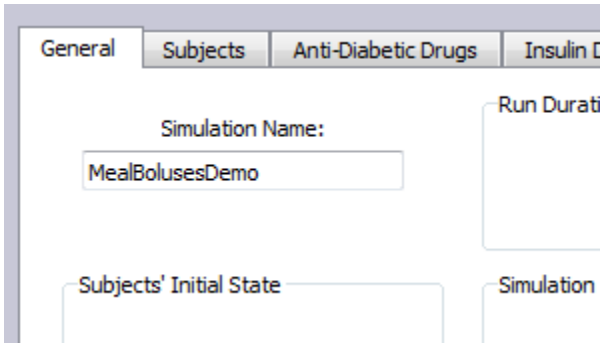


Figure 124: Walkthrough 1 -- Naming MealBolusesDemo

Open the *Controls* tab again and add the MealBolusRx control:

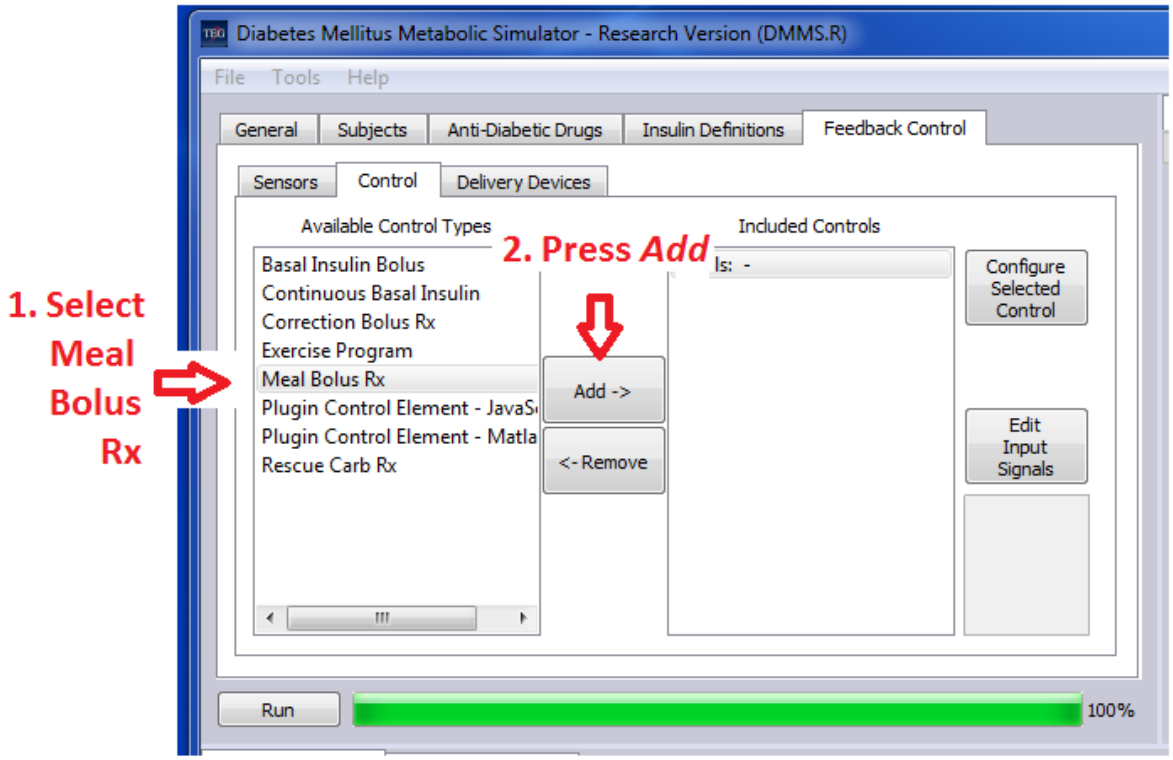


Figure 125: Walkthrough 1 -- Including the Meal Bolus Rx Control

Select the Meal Bolus Rx and press *Configure Selected Control*.

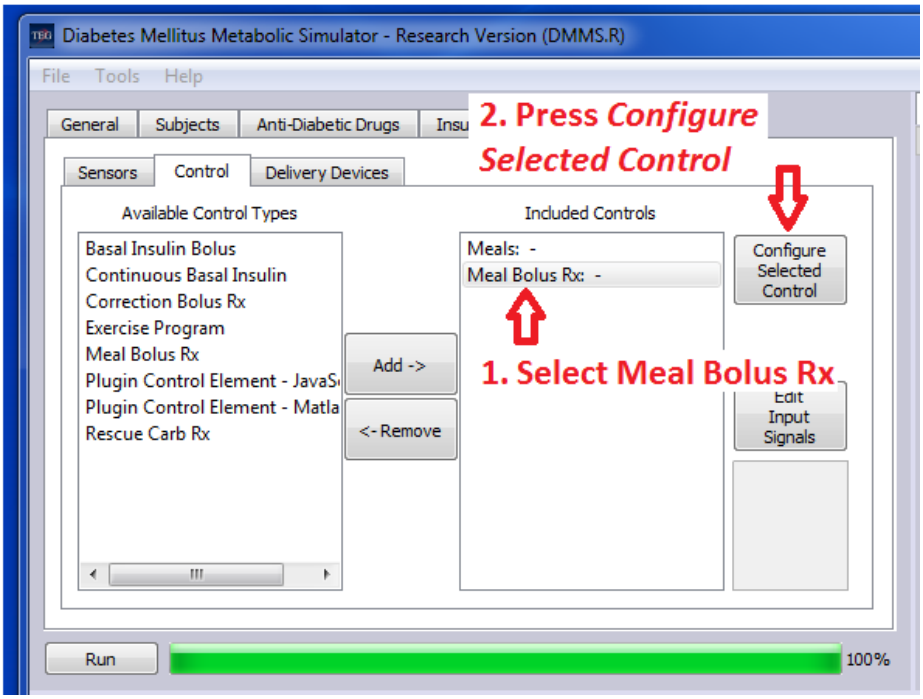


Figure 126: Walkthrough 1 -- Configuring the Meal Bolus Rx Control

The following dialog should appear:

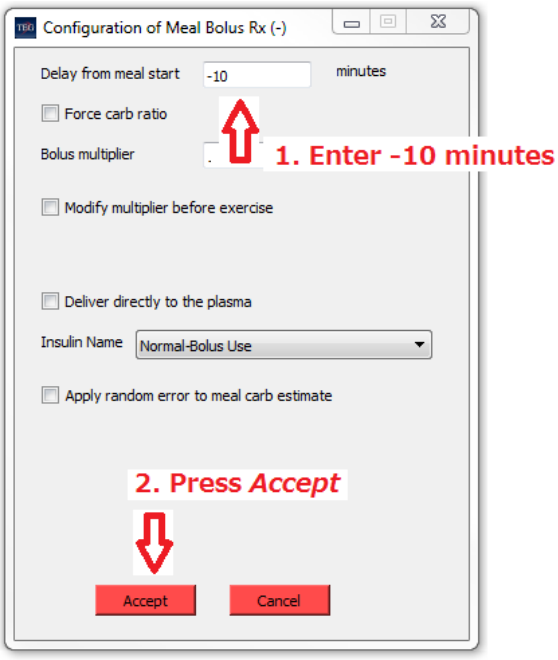


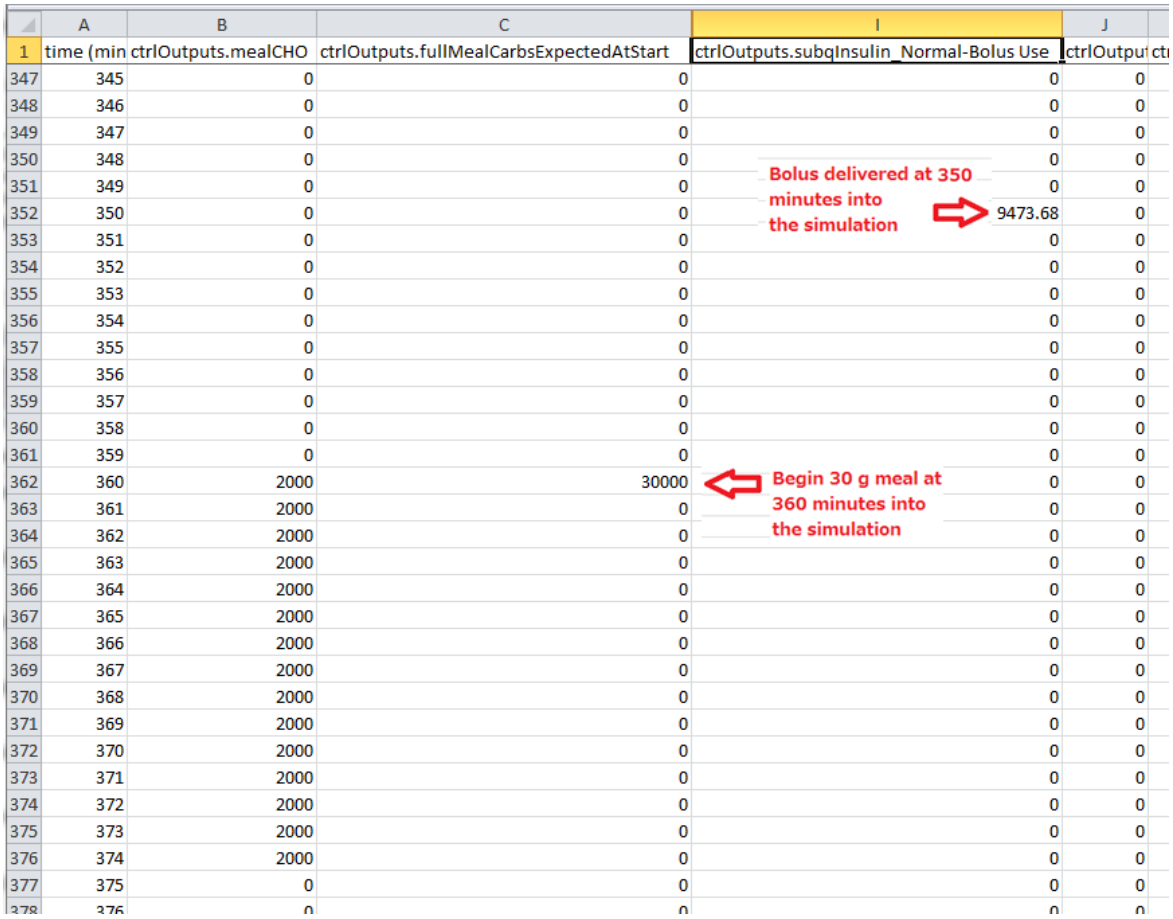
Figure 127: Walkthrough 1 -- Meal Bolus Configuration

Specify delivery of the bolus at 10 minutes before the meal starts. Do this by entering -10 in the *Delay from meal start* edit box. (Negative values indicate a time span before the meal start; positive values indicate a time span after the meal start).

Now press the *Run* button again. Open "…\Documents\DMMS.R\results\MealBolusesDemosignalHistory.Type1 Adult.adult#001.csv."

Column H contains values for the subcutaneous insulin for the subject. We can see here that a bolus of 9473.68 mg was delivered 10 minutes before the start of the meal.

We can confirm this by examining the .csv file produced during the simulation. (Columns D-H hidden.)



	A	B	C	I	J
1	time (min	ctrlOutputs.mealCHO	ctrlOutputs.fullMealCarbsExpectedAtStart	ctrlOutputs.subqInsulin_Normal-Bolus Use	ctrlOutput
347	345	0	0	0	0
348	346	0	0	0	0
349	347	0	0	0	0
350	348	0	0	0	0
351	349	0	0	0	0
352	350	0	0	9473.68	0
353	351	0	0	0	0
354	352	0	0	0	0
355	353	0	0	0	0
356	354	0	0	0	0
357	355	0	0	0	0
358	356	0	0	0	0
359	357	0	0	0	0
360	358	0	0	0	0
361	359	0	0	0	0
362	360	2000	30000	0	0
363	361	2000	0	0	0
364	362	2000	0	0	0
365	363	2000	0	0	0
366	364	2000	0	0	0
367	365	2000	0	0	0
368	366	2000	0	0	0
369	367	2000	0	0	0
370	368	2000	0	0	0
371	369	2000	0	0	0
372	370	2000	0	0	0
373	371	2000	0	0	0
374	372	2000	0	0	0
375	373	2000	0	0	0
376	374	2000	0	0	0
377	375	0	0	0	0

Figure 128: Walkthrough 1 -- Results File Showing Meal Boluses

The subject's plasma glucose concentration data are shown in column AF (subjE.Gp conc). Sort this column (Data tab, Sort & Filter, descending) and note that the largest value is 238.534 mg/dL.

	AE	AF	AG
G ut	subjE.I ap	subjE.Gp conc	subjE.Gluc su
286	0	238.534	0
286	0	238.533	0
286	0	238.53	0
286	0	238.529	0
286	0	238.522	0
287	0	238.52	0
287	0	238.511	0
287	0	238.508	0
287	0	238.495	0
287	0	238.492	0

Figure 129: Walkthrough 1 -- Glucose Concentrations with Meal Boluses

This completes our very simple walkthrough. We have shown how a meal bolus therapy can have an observable effect on the blood glucose level of a Type 1 patient.

11 Walkthrough 2: Alarm-based Correction Boluses

In this example, we are going to monitor the subject's BG levels via CGM. When the application detects that the BG value has exceeded a configured threshold it will deliver a correction bolus.

Begin by naming the simulation "AlarmBasedDemo." Select the same subject we used in the Meals and Meal Boluses example: Type1 Adult – adult#001, but without Meals, Meal Boluses, or any other therapy. Without making any other changes, run a simulation to show this subject's baseline plasma glucose concentration. The output should look like this:

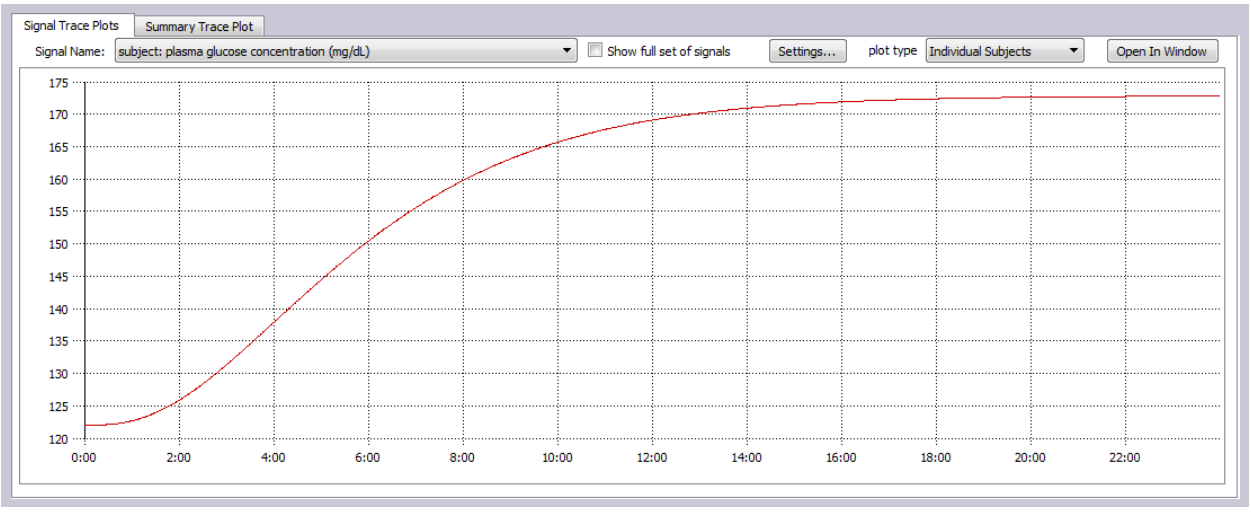


Figure 130: Walkthrough 2 -- Baseline BG (no treatment)

Note that under these conditions, the subject's BG level rises to around 165 mg/dL by noon and stays above this level for the rest of the day.

The alarm-based correction bolus feature requires two sensor signals:

- A signal from the CGM that will serve to trigger the delivery of the bolus
- An SMBG-generated signal that will be used to confirm that the CGM signal is accurate

To provide these two signals, open the *Sensors* tab and select the Ideal CGM sensor and the SMGG sensor. The dialog should look like this:

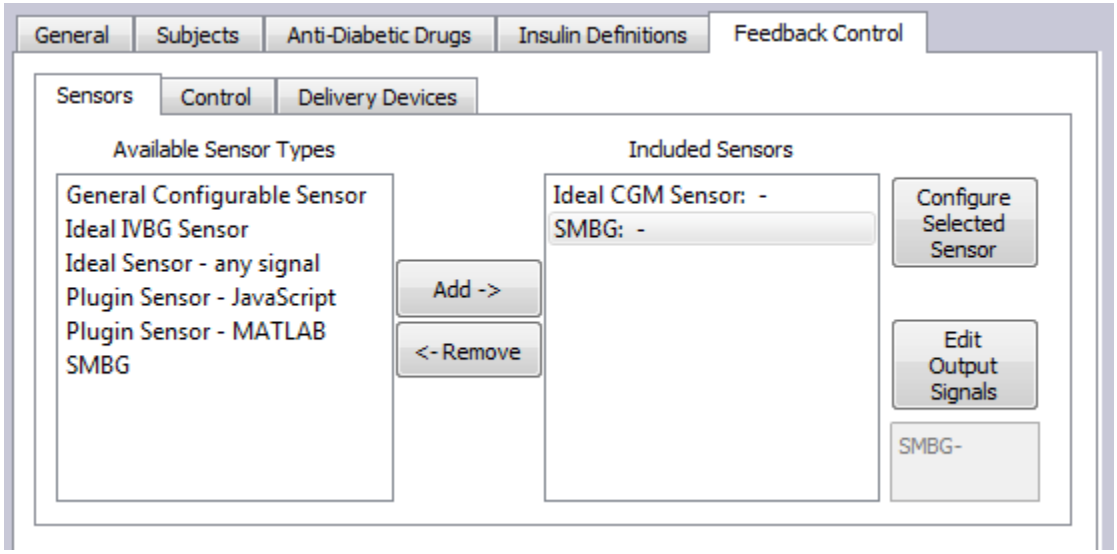


Figure 131: Walkthrough 2 -- Sensors and Sensor Output Signals

Note that each of these sensors is configured to provide the needed signal.

Now go to the *Control* tab and add the correction bolus provider (Correction Bolus Rx):

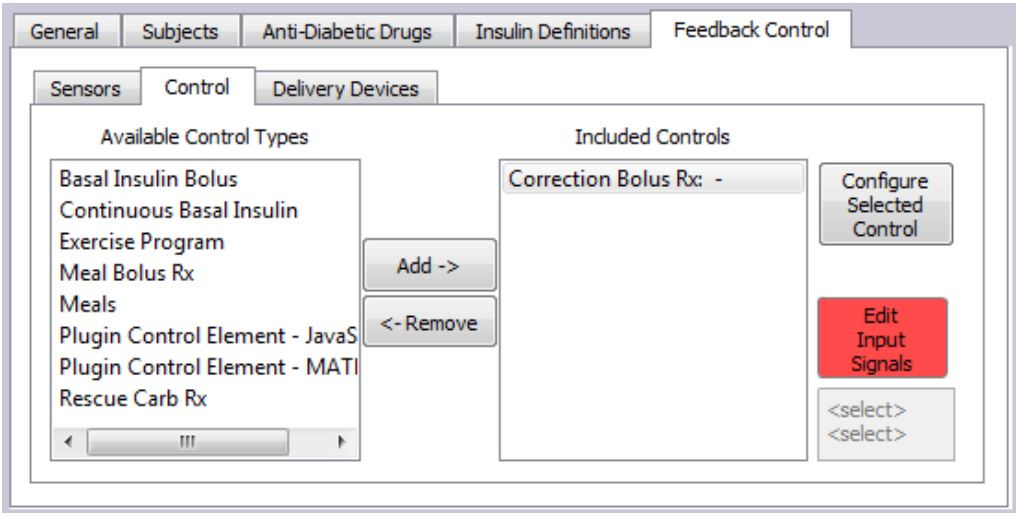


Figure 132: Walkthrough 2 -- Control Elements and Control Element Input Signals

Note that the Edit Input Signals button is red. This is to call your attention to the fact that the Correction Bolus Rx is not pre-configured to accept any specific input signals. It is our job to match the available sensor output signals to the desired control input signals. To do this, press the *Edit Input Signals* button. The following dialog should appear:

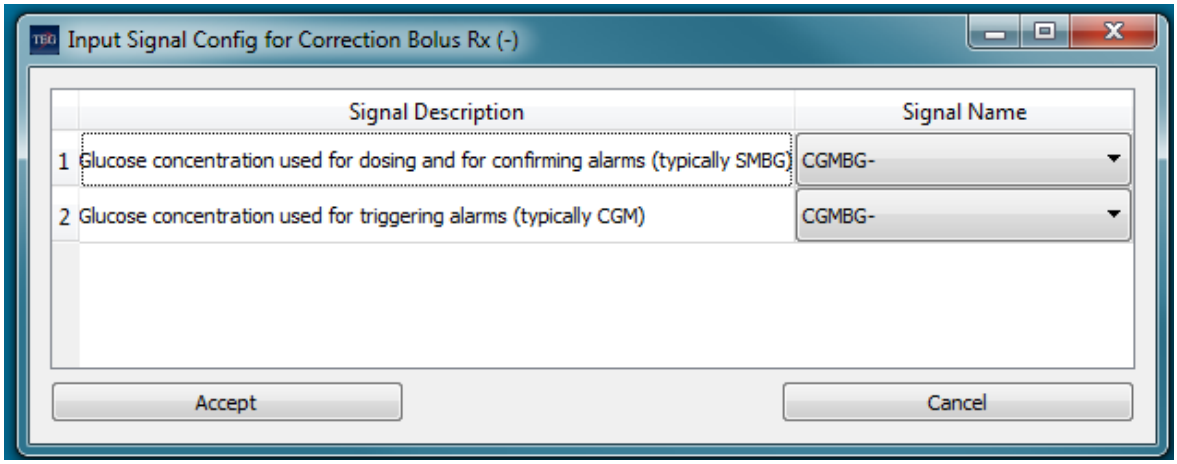


Figure 133: Walkthrough 2 -- Input Signal Config for Correction Bolus Provider (default)

The left column, labeled *Signal Description*, shows the signals that the Correction Bolus Provider needs. The *Signal Name* column contains dropdown controls to allow you to select these signals. The CGM glucose concentration signal is, by default, setup to use the CGMBG signal name. However, so is the SMBG glucose concentration. We'll need to change that to use the SMBG signal name:

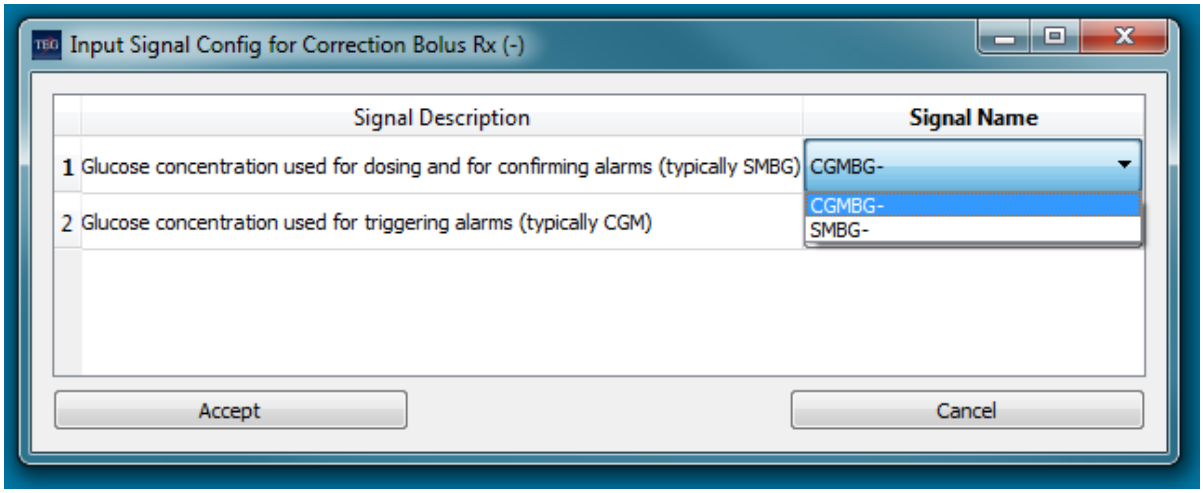


Figure 134: Walkthrough 2 -- Input Signal Config for Correction Bolus Provider

After we have done this, the *Control* tab should look like the following:

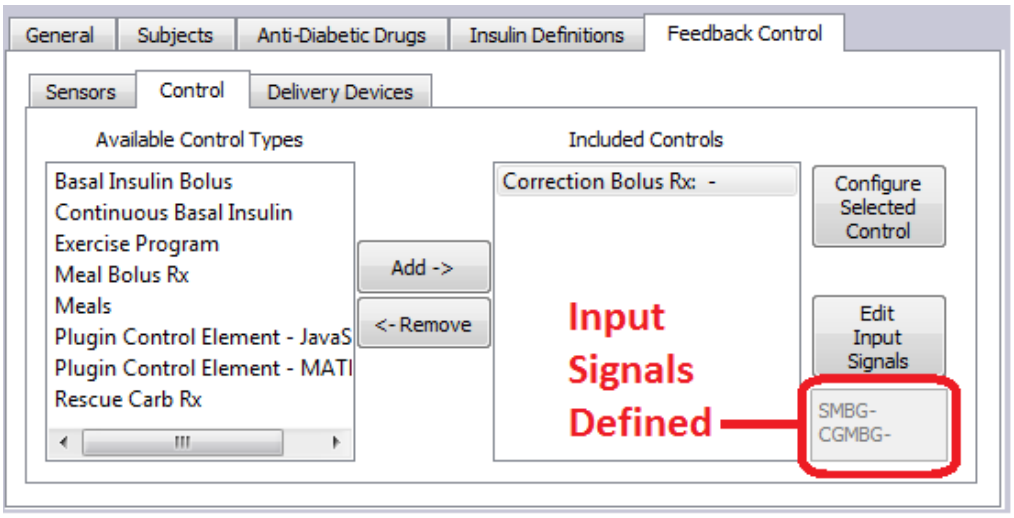


Figure 135: Walkthrough 2 -- Correction Bolus Rx Input Signals

Now press the *Configure Selected Control* button. Ensure that *only* the *Enable alarm-based corrections* checkbox is checked. Enter 150 in the *BG threshold* edit box:

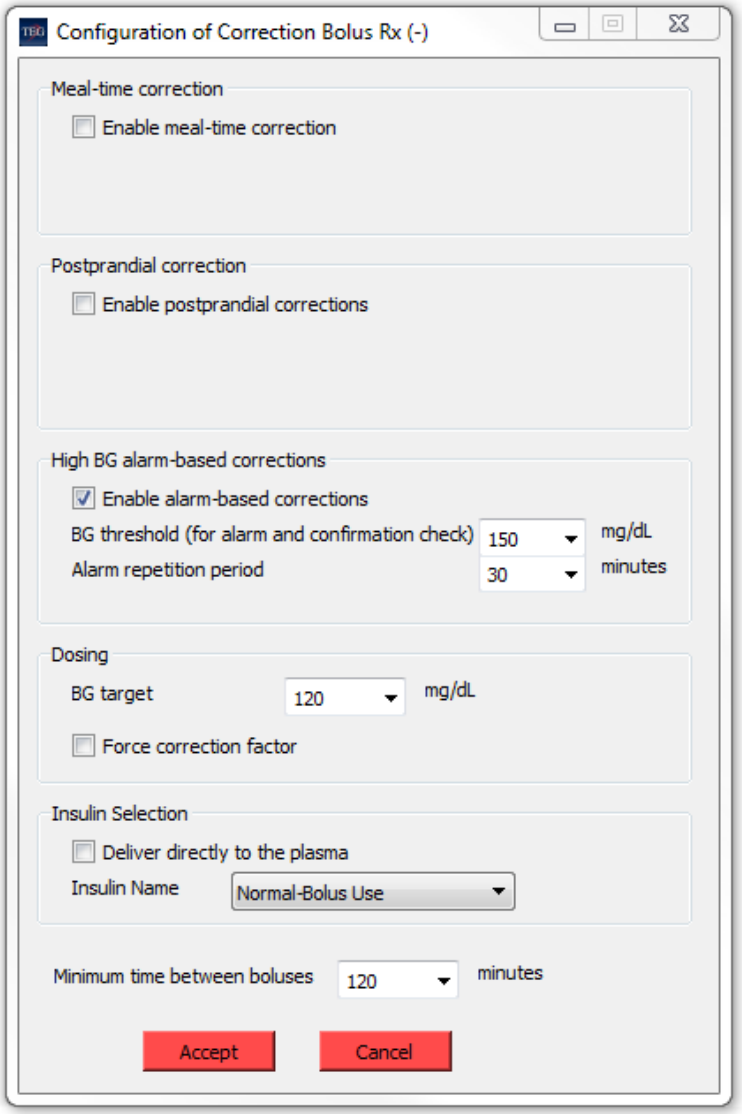


Figure 136: Walkthrough 2 – Correction Bolus Rx Configuration

Press the *Accept* button and re-run the simulation. The Signal Trace Plot should look like this:

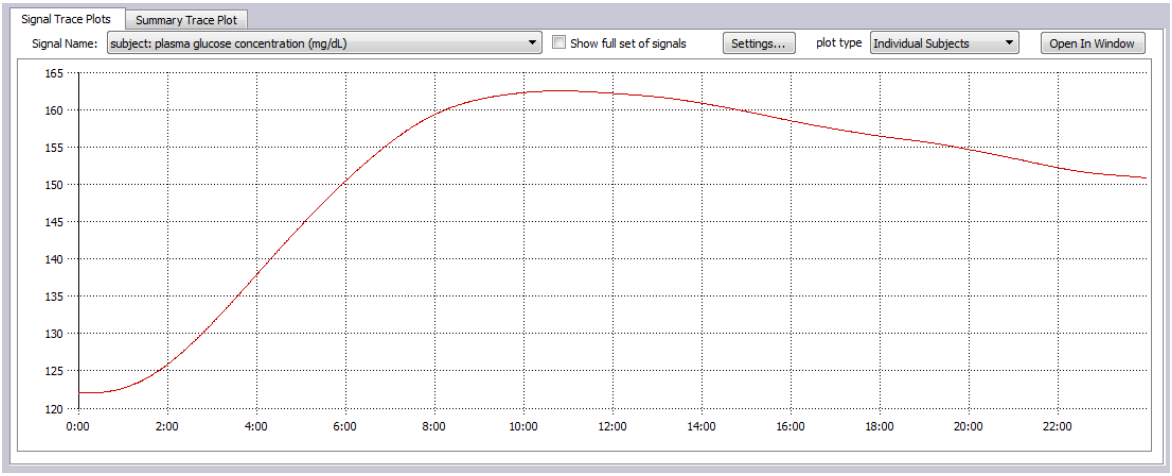


Figure 137: Walkthrough 2 -- BG with Correction Boluses

Note that now, the BG concentration never exceeds 165 mg/dL.

Change the *Signal Trace Plot* to show the Normal Bolus Use input:

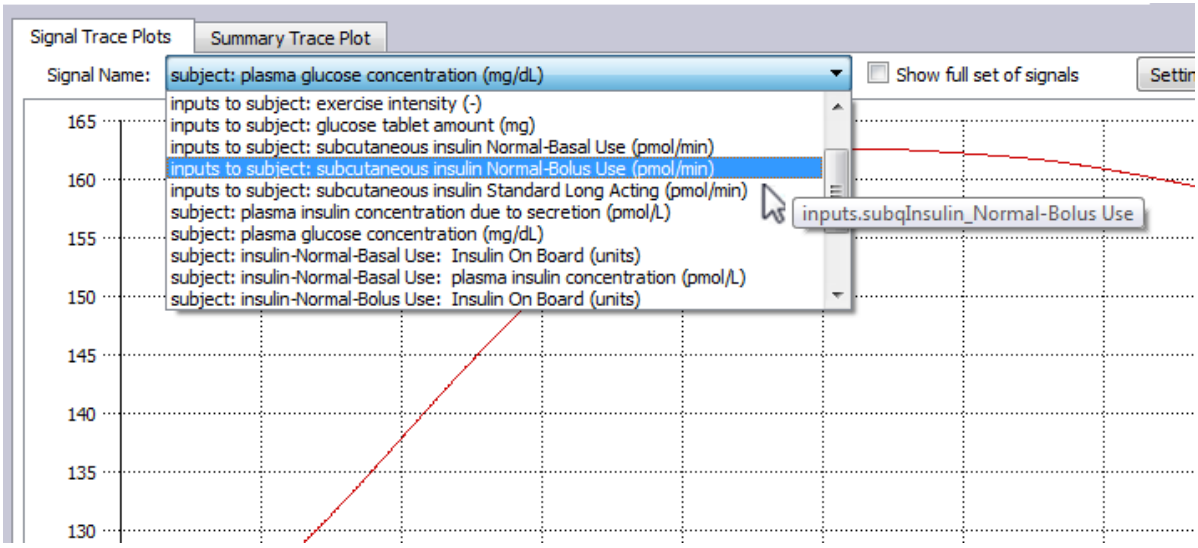


Figure 138: Walkthrough 2 -- Selecting Plot Trace

The display should change to this.

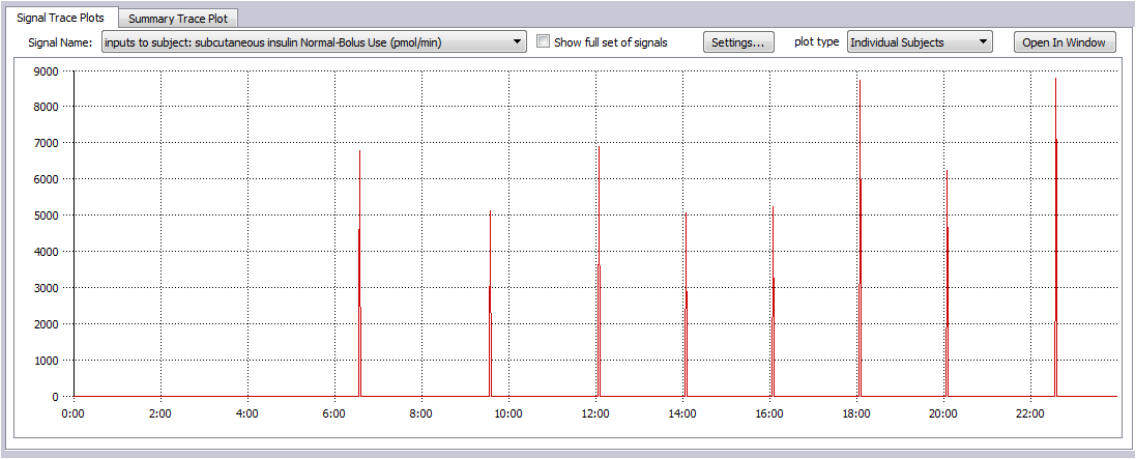


Figure 139: Walkthrough 2 -- Subcutaneous Insulin Normal Bolus Use

To show the effect of these subcutaneous boluses in the bloodstream we can simply change the displayed *Signal Name* to "subject: insulin-Normal-Bolus-Use: plasma insulin concentration (pmol/L)":

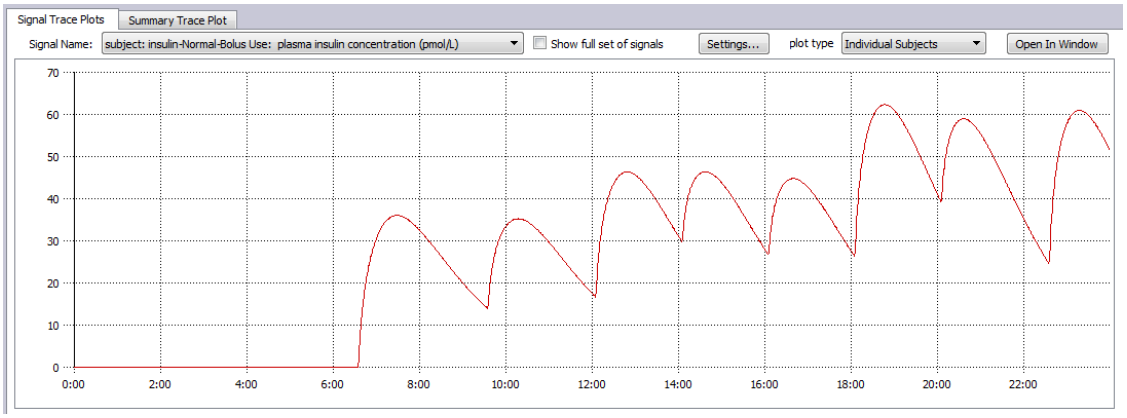


Figure 140: Walkthrough 2 -- Plasma Insulin Concentration

12 Walkthrough 3: A JavaScript Plugin

This example will demonstrate the use of a JavaScript plugin that provides the functionality of the alarm-based correction bolus provider with one modification: If the subject is scheduled to have a moderate or heavy-intensity exercise session within an hour of the bolus, the dosage will be reduced by 50%. Again, we will use Type 1 Adult – adult#001 as our subject.

The JavaScript plugin file PluginReducedBolusControlElement.js is supplied as a part of the DMMS.R installation. You can find this file in the config folder under DMMS.R. Like the alarm-based correction example shown in section 11, this walkthrough also requires an SMBG signal (SMBG) and an Ideal CGM signal (CGMBG).

Name this simulation something like “PluginDemo.” After selecting the subject and setting up the two sensors, the next step is to add an exercise regimen: Go to the control sub-tab and include an Exercise Program in the included controls. Configure four, 15-minute, moderately intensive, exercise sessions at 6, 7, 8 and 9:00 AM.

Now include a Plugin Control Element and configure it to use the implementation file named “PluginReducedBolusControlElement.js” in your project config folder.

Set the input signals for the Plugin Control to SMBG- (SMBG measurement) and CGMBG- (CGM reading). These signal names must match the sensor output signal names you configured earlier.

Now run the simulation. We can examine the output traces to see the relationships among blood glucose, insulin boluses, and exercise sessions (See Figure 141, Figure 142 and Figure 143).

The JavaScript file defines the target BG level as 120 mg/dL and the threshold value to trigger the bolus as 150 mg/dL.

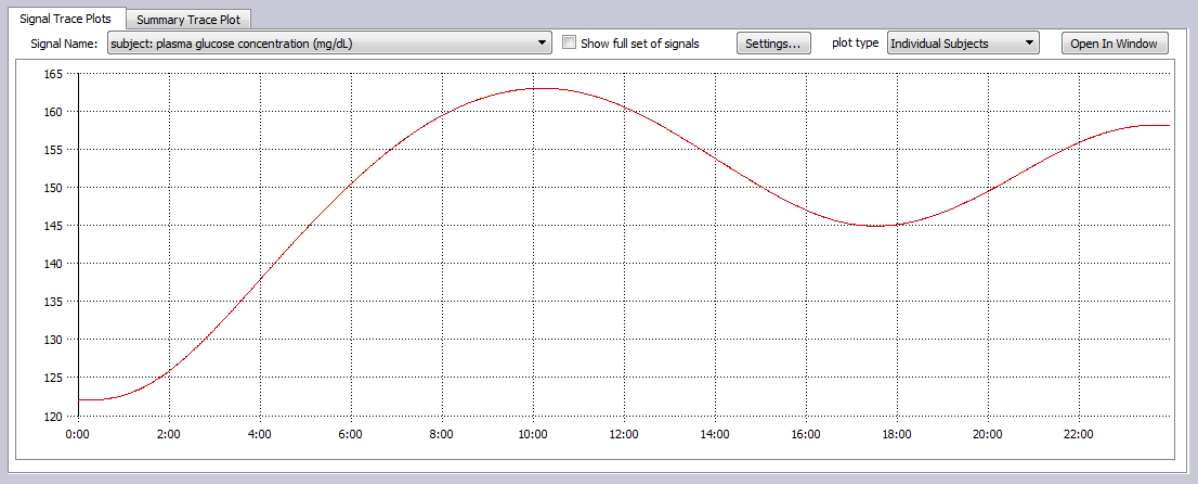


Figure 141: Walkthrough 3 -- Plasma BG Trace

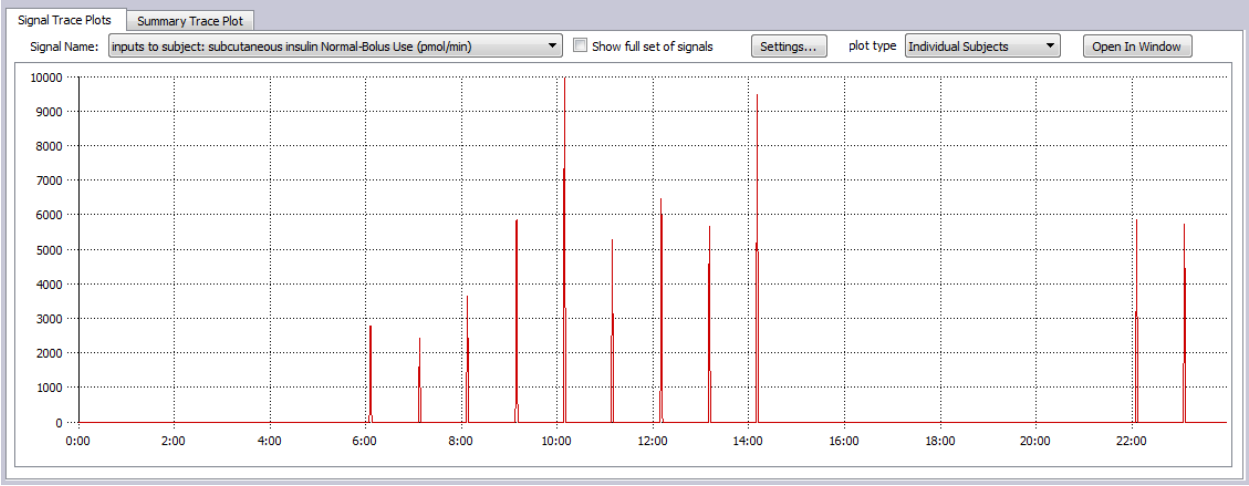


Figure 142: Walkthrough 3 -- Correction Boluses

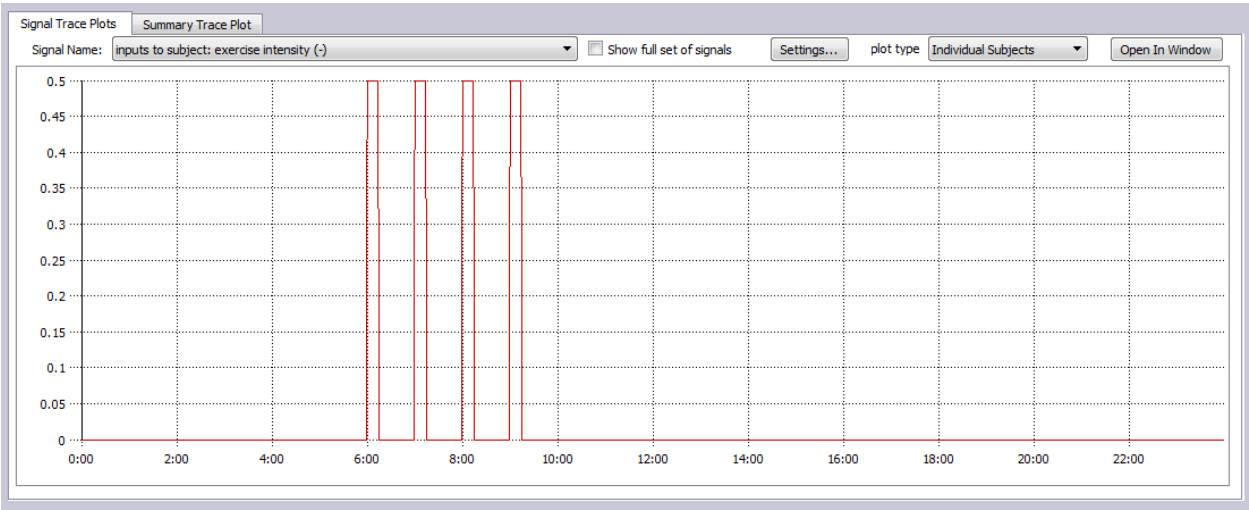


Figure 143: Walkthrough 3 -- Exercise Program

Note that while 11 boluses were delivered over the 24-hour simulation period, only the first three, which were delivered in advance of a scheduled exercise session, were affected by the action of the plugin.

To get an idea of how the insulin from these boluses migrates through the body, we can look at two other output traces, Plasma Insulin Concentration, and Insulin On Board.

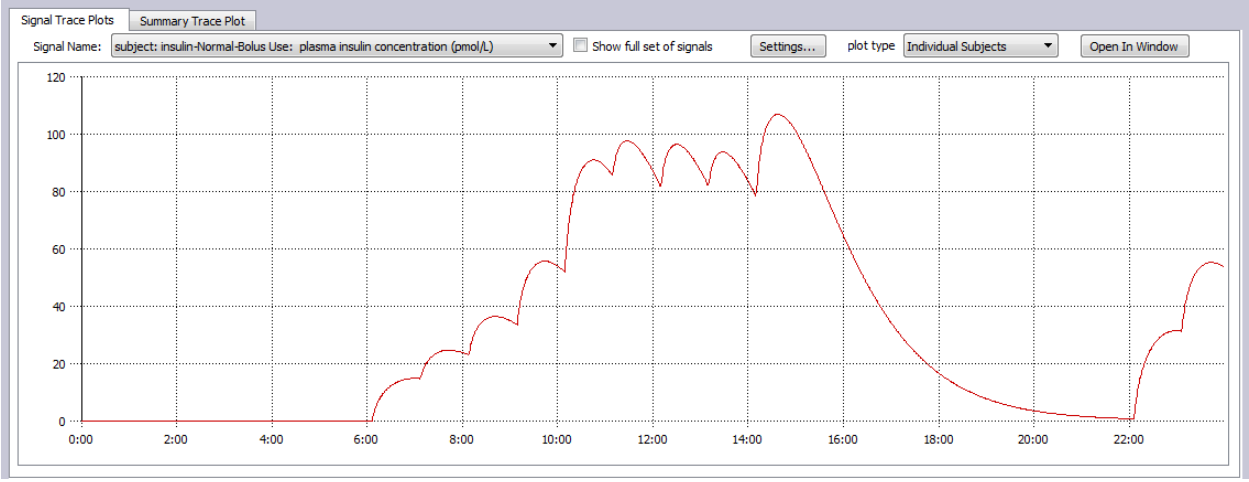


Figure 144: Walkthrough 3 -- Plasma Insulin

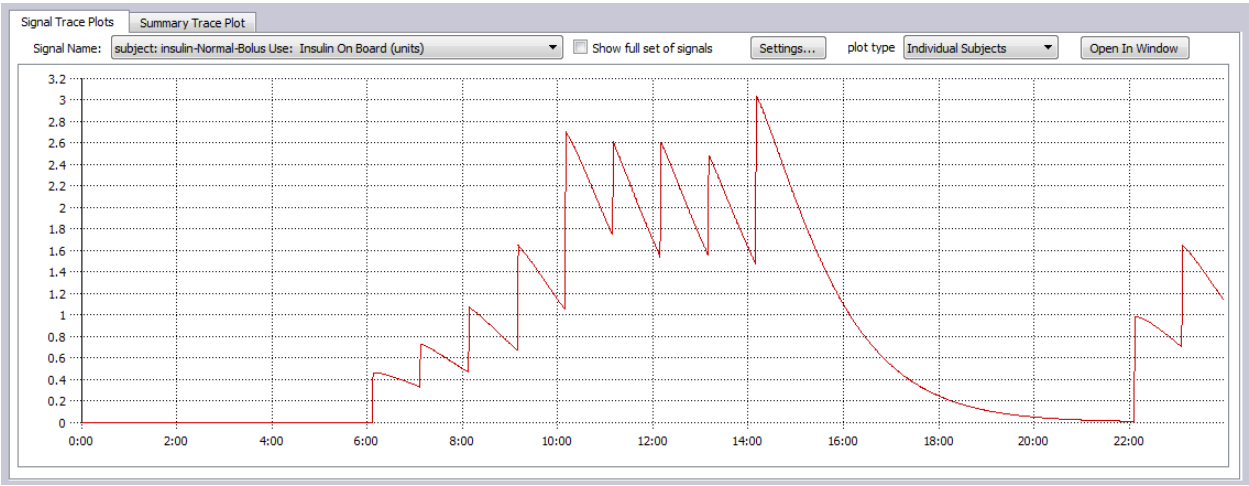


Figure 145: Walkthrough 3 -- IOB

The behavior of this plugin is governed by several variables defined in the script. These variables are used as program constants: they are initialized when declared, but never changed by the program. You may want to copy the file, change some of the values, and note the effect on the output.

Variable Name	Default Value	Units	Range	Notes
pMolesPerUnit	6000	none	>0	This is the factor for normal insulin. Typically not changed.
bgTarget	120	mg/dL	>0	Target blood glucose level for subject. Used to calculate bolus size.
bgThreshold	150	mg/dl	>bgTarget	Blood glucose level used to trigger delivery of bolus.

minimumTimeSinceLastBolus	60	minutes	>0	No bolus delivered if most recent one was within this window.
exerciseTimeHorizon	120	minutes	>0	Any required bolus will be adjusted if an exercise session is scheduled within this window.
exerciseIntensityThreshold	0.3	none	0-1	Algorithm will adjust bolus size only if upcoming exercise is above this level.
exerciseCorrectionFactor	0.5	none	0-1	Factor to be used when adjusting bolus size.

Table 10: Walkthrough 3 -- Program Variables

13 Determining Which Version of DMMS.R is Installed

To determine the version designation of DMMS.R installed on a Windows platform, open the *Help* tab on the dialog and select *About DMMS.R*. The following message will appear:

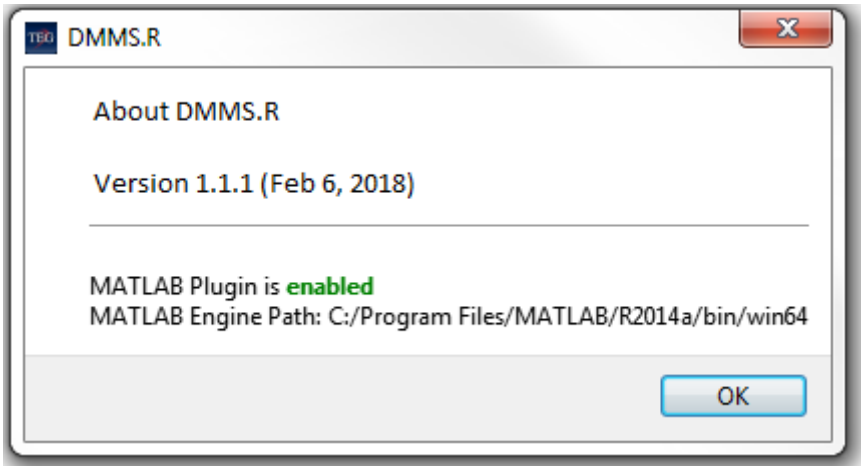


Figure 146: About DMMS.R (Windows)

14 Glossary of Terms

This section defines a few terms and abbreviations used in this manual with which the reader may not be familiar. It does not attempt to cover all terms used in the treatment or simulation of Diabetes Mellitus.

Item	Description
BG	Blood Glucose
BMI	Body Mass Index
Bolus	A dose of insulin given over a short time (<3 minutes)
Carbohydrate Ratio (carb ratio)	The amount of carbohydrate (gms) that is covered by 1 unit of insulin (usual range is 10-15 gm for adults)
CF	See Correction Factor
CHO	Carbohydrates
CGM	Continuous Glucose Monitor
Correction Factor	Also known as the insulin sensitivity factor- the amount of insulin needed to bring the blood glucose level into target range when glucose levels are too high.
CR	See Carbohydrate Ratio
Drug Class	A classification applied to drugs within the **drug database**, where there is no interaction between drugs of different classes. Drug classes may include Sensitizers, SGLT-2 Inhibitors, Secretagogues, and Alpha Glucosidase Inhibitors. The DMMS can perform a simulation using any number of drugs, as long as no two selected drugs are from the same class.
Drug Database	The library of drugs whose effects on a subject are represented in the DMMS as a constant alteration (through the duration of a simulation run) of the subject's defined metabolic characteristics. The database can specify several drugs (e.g., metformin), and, for each of these, the effect resulting from several different dosages. It also can group the drugs into **drug classes**, where there is no interaction between drugs of different classes.
Glucagon	A counter-regulatory hormone to be given when insulin-overdose has occurred to assist with glucose recovery from hypoglycemia
Insulin Analog	A specific type of insulin, with PK/PD characteristics defining the behavior of the insulin *after it has been released* into subcutaneous space or the bloodstream. The characteristics defining how the insulin is released (into either the bloodstream or subcutaneous space) may be established by combining the **insulin analog** with a bioavailability value and any release profile defined

	in the **insulin release database**, thus specifying an **insulin-release pair**. An **insulin analog** is represented by specifying an **insulin class**, along with the set of values for the PK/PD equation parameters defined for that class. Note that this differs slightly, in concept, from the conventional use of the term "insulin analog," in which release characteristics are considered. Within the DMMS, release characteristics and post-release PK/PD characteristics are treated independently to maximize flexibility in defining insulins.
Insulin Class	A class of **insulin analogs**, all of which can be modelled with the same post-release PK/PD equations, possibly with different parameters applied to these equations. The coefficients of the applicable functions may vary from one **insulin analog** to another within the given class. Note that the **insulin class** does not address release characteristics of an insulin analog – only the PK/PD characteristics of the insulin once it has been released into either subcutaneous space or the bloodstream. The release characteristics are defined independently (see **insulin release database**).
Insulin Database	Database defining a set of **insulin analogs** available for use in the simulation, where each **insulin analog** is defined by an **insulin class** along with a set of parameter values specifically required for that class. A given **insulin analog** defined in the **insulin database** may be combined with any release definition found in the **insulin release database** and a bioavailability value, to fully specify an insulin, including its delivery mechanism.
Insulin on board	See IOB
Insulin Release Database	Database defining a set of insulin release definitions, where each describes the profile of release over time, resulting from a bolus. Each release definition is specified by defining release rate as a piecewise linear function of time. The database will contain a sequence of timestamped values, each representing the proportion of the bolus (potentially reduced based on a specified bioavailability) "released" to the body per unit of time, at a specified time after the bolus is provided. The piecewise linear function is defined by interpolating between the points provided in this way. The actual "proportion of the bolus" values included in the database may be scaled arbitrarily, as the DMMS will automatically adjust these to ensure that, over the full duration of release, the full bolus is accounted for.
Insulin-release pair	The combination of an **insulin analog** defined in the **insulin database** and an insulin release definition from the **insulin release**

	database and a specified bioavailability. Note that the release specified may be "none", which would indicate that the insulin to be used is not of a "slow release" variety, meaning that the full amount of an injection (after accounting for bioavailability) will be introduced instantaneously to the applicable human-model compartment (subcutaneous space or plasma, depending on whether the dose is to be delivered via IV).
IOB	Insulin on board. This is defined as the total amount of insulin within all compartments of the **subject state** in which this insulin can continue to influence other compartments of the **subject state**. This includes subcutaneous insulin, insulin in the plasma, and insulin in the liver. The time that the insulin spends in these compartments (related to what is generally called "duration of insulin action") will depend on the **insulin analog** being used, and on the **subject parameters** that apply. **IOB** can be divided into two sub-categories, as follows: **Bolus IOB**: The portion of the IOB that is attributable to insulin provided as a meal bolus or correction bolus. **Basal IOB**: The portion of the IOB that is attributable to insulin provided as basal insulin – whether this is delivered "continuously" (e.g., via a pump) or as a basal bolus of long-acting insulin. In the latter case, the insulin is not considered to be part of the IOB until it has been "released into the body," as defined by the applicable insulin release definition in the **insulin release database**.
Normal insulin	Any insulin that has the release and post-release PK/PD characteristics of a typical rapid-acting insulin. In the DMMS.R's GUI, all **insulin-release pairs** and **insulin analogs** shown as "normal" have these characteristics.
scenario	The set of selections that define what the *in-silico* subject or subjects experience through the course of a single simulation run, as well as the settings defining the set of *in silico* subjects and the time span of the run. The selections defining what is experienced by the subjects includes: • Devices (e.g., pumps and sensors) to be used • Lifestyle choices (e.g., meals and exercise) • Treatment protocols (e.g., meal & correction bolus algorithms, hypoglycemia treatment logic, etc.) • Oral medications (drugs selected from the **drug database**) • Options for simulation algorithms Within the context of DMMS.R, a study arm is fully defined by a scenario, but a given scenario, or variations on a scenario, may

	apply to multiple arms. Thus, one may say “Arm 2 uses the same scenario as arm 1, but with different meal sizes”.
Sensor database	An xml file containing configurations of several commonly used sensors that can be modelled via the “standard sensor” supported by DMMS.
Simulation state	The set of values that fully define the simulated world at an instant in time. This includes the **subject state** for all subjects, the time-related information for the simulation (day of week, time of day, day of year), and the state of all configured feedback elements. Given a **simulation state**, the set of subject inputs, and the set of **subject parameters**, all future states of the simulation can be determined.
Simulation time interval	The interval of simulated time between two consecutive iterations of the simulation, where iterations represent each point at which the complete simulation state is defined.
Simulation run	The execution of a simulation as defined by a single scenario, spanning the time defined within that scenario, and including all subjects defined in that scenario.
Standard Long-Acting Insulin	An **insulin-release pair** with a release profile that results in uniform distribution of insulin throughout a 24-hour period, and, once in the body, having post-release PK/PD characteristics identical to those of **normal insulin**.
Subject	The *in-silico* patient in the simulation
Subject database	A set of data defining metabolic and biometric characteristics of each *in silico* subject that can be represented in the DMMS. The subject database is represented in an XML file and a set of csv files that organize subjects into populations grouped by such things as diabetes type (Type 1, Type 2, or pre-diabetic) and age group (adult, adolescent, or child).
Subject parameters	The set of metabolic and biometric characteristics of an *in-silico* subject, as defined in the **subject database**.
Subject state	The set of values defining the complete state of the subject’s metabolic system, such that the **subject state** at any future time can determined by the current state, all subject inputs occurring between the current time and the applicable future time, and the **subject parameters**.

Abbott

15 Contact Information

For additional information about, or assistance with, this or other Epsilon Group products please contact:

mlEpsilonInfo@abbott.com

Index

Appendix A: Output Data Definition

Output data signals prefaced with “ctrlOutputs” are generated by “control elements” as shown in Figure 3. These are the intended subject inputs, which may be subsequently modified by a delivery element, if such an element is defined to operate on the applicable subject input.

The signals prefixed with “inputs” represent the subject inputs as actually delivered to the subject. These will be identical to the corresponding ctrlOutputs signal if no delivery element handles the associated subject input, otherwise, they may differ (e.g., as a result of noise introduced by the delivery element).

Table 11 contains definitions of simulation default data as recorded in .CSV output files.

Table 12 defines additional output associated with user-defined insulins. Data resulting from user definitions will be merged into Table 11, as shown by the light-green rows ( ). Note that when any user-defined insulins are used in the simulation, or when any sensors are included, the column IDs (‘A’,’B’,’C’…) will be adjusted to accommodate the extra columns.

Default Column ID	Signal Name	Units	Description
A	time (min)	minutes	time since start of simulation
B	ctrlOutputs.mealCHO	mg	meal carbohydrates delivered within the current iteration of the simulation
C	ctrlOutputs.fullMealCarbsExpectedAtStart	mg	total carbohydrates expected for the full meal
D	ctrlOutputs.glucOrDextIvInj	mg	glucose/dextrose IV injection delivered within the current iteration of the simulation
E	ctrlOutputs.glucagonSqInj	mg	glucagon subcutaneous injection delivered within the current iteration of the simulation
F	ctrlOutputs.exerciseIntensity	(none)	exercise intensity as a fraction of "full exertion" (0.0 for none, 1.0 for full)
G	ctrlOutputs.glucoseTab	mg	The carbohydrate amount intended to be delivered in a glucose tablet.
H	ctrlOutputs.subqInsulin_Normal-Basal Use	pmol	amount of Normal Basal Insulin injected subcutaneously within the current iteration of the simulation

I	ctrlOutputs.subqInsulin_Normal-Bolus Use	pmol	amount of Normal Bolus Insulin injected subcutaneously within the current iteration of the simulation
Control output data resulting from any user-defined subcutaneously delivered insulins will be inserted here. There will be a new column for each new insulin, other than slow-release insulins, defined:			
	ctrlOutputs.subqInsulin_<new insulin name>	pmol	amount of user-defined Insulin injected subcutaneously within the current iteration of the simulation
J	ctrlOutputs.ivInsulin_Normal-Basal Use	pmol	amount of Normal Basal Insulin delivered directly to the plasma within the current iteration of the simulation
K	ctrlOutputs.ivInsulin_Normal-Bolus Use	pmol	
Control output data resulting from any user-defined insulins delivered directly to the plasma will be inserted here. Note that this data set will be repeated for each new insulin, other than slow-release insulins, defined:			
	ctrlOutputs.ivInsulin_<new insulin name>	pmol	amount of user-defined Insulin delivered directly to the plasma within the current iteration of the simulation
L	ctrlOutputs.slowRelInsulin_Standard Long Acting	pmol	amount of Standard Long-Acting inulin delivered within the current iteration of the simulation
Control output data resulting from any user-defined slow-release insulins delivered directly to the plasma will be inserted here. Note that this data set will be repeated for each new slow-release insulin defined:			
	ctrlOutputs.slowRelInsulin_<new insulin name>	pmol	amount of user-defined slow-release Insulin delivered within the current iteration of the simulation
M	inputs.mealCHO	pmol	meal carbohydrates within the current iteration of the simulation
N	inputs.fullMealCarbsExpectedAtStart	mg	total carbohydrates expected for the full meal
O	inputs.glucOrDextIvInj	mg/min	glucose/dextrose IV injection within the current iteration of the simulation
P	inputs.glucagonSqInj	mg	glucagon subcutaneous injection delivered within the current iteration of the simulation

Q	inputs.exerciseIntensity	(none)	exercise intensity as a fraction of "full exertion" (0.0 for none, 1.0 for full)
R	inputs.glucoseTab	mg	The carbohydrate amount delivered to the subject in a glucose tablet.
S	inputs.subqInsulin_Normal-Basal Use	pmol	amount of Normal Basal Insulin injected subcutaneously within the current iteration of the simulation
T	inputs.subqInsulin_Normal-Bolus Use	pmol	amount of Normal Bolus Insulin injected subcutaneously within the current iteration of the simulation
Input data resulting from any user-defined subcutaneously-delivered insulins will be inserted here. Note that this data set will be repeated for each new insulin, other than slow-release insulins, defined:			
	inputs.subqInsulin_<new insulin name>	pmol	amount of user-defined subcutaneously-injected insulin delivered to the subject within the current iteration of the simulation
U	inputs.ivInsulin_Normal-Basal Use	pmol	amount of Normal Basal Insulin delivered directly to the plasma within the current iteration of the simulation
V	inputs.ivInsulin_Normal-Bolus Use	pmol	amount of Normal Bolus Insulin delivered directly to the plasma within the current iteration of the simulation
Input data resulting from any user-defined insulins, delivered directly to the plasma, will be inserted here. Note that this data set will be repeated for each new insulin, other than slow-release insulins, defined:			
	inputs.ivInsulin_<new insulin name>	pmol	amount of user-defined insulin, delivered directly to the plasma within the current iteration of the simulation
W	inputs.slowRelInsulin_Standard Long Acting		amount of user-defined insulin, delivered directly to the plasma within the current iteration of the simulation
Control output data resulting from any user-defined slow-release insulins delivered directly to the plasma will be inserted here. Note that this data set will be repeated for each new slow-release insulin defined:			

	inputs.slowRelInsulin_<new insulin name>	pmol	amount of user-defined slow-release Insulin delivered within the current iteration of the simulation
X	subjE.Gs conc	mg/Dl	Subcutaneous glucose concentration
Y	subjE.Ip conc secreted	pmol/L	Plasma insulin concentration for the normal human insulin originating from the subject's pancreas.
Z	subjE.EGP rate	mg/kg/min	Endogenous glucose production rate
AA	subjE.G app rate	mg/kg/min	Rate of appearance of glucose
AB	subjE.G util rate	mg/kg/min	Glucose utilization rate at all tissues
AC	subjE.I app rate	pmol/L/min	Rate of appearance of insulin
AD	subjE.Gp conc	mg/dL	Plasma glucose concentration
AE	subjE.Glucagon app rate	ng/L/min	Rate of appearance of glucagon
AF	subj.Qsto1	mg	carbs in first (solid) phase of the stomach
AG	subj.Qsto2	mg	carbs in 2nd (liquid) phase of the stomach
AH	subj.Qgut	mg	carbs in intestine
AI	subj.Gp	mg/kg	glucose mass in plasma and rapidly equilibrating (insulin-independent) tissues
AJ	subj.Gt	mg/kg	glucose mass in slowly equilibrating (insulin-dependent) tissues
AK	subj.IpSecreted	pmol/kg	Insulin mass in the plasma, for the normal human insulin originating from the subject's pancreas
AL	subj.X	pmol/L	insulin action on glucose utilization
AM	subj.I1st	pmol/L	1st compartment used in delayed insulin signal for insulin action on glucose production
AN	subj.Id	pmol/L	delayed insulin signal for insulin action on glucose production, Id
AO	subj.IlSecreted	pmol/kg	insulin mass in the liver, for the normal human insulin originating from the subject's pancreas

AP	subj.Gs	mg/kg	subcutaneous glucose
AQ	subj.Y	pmol/kg /min	Static component of insulin secretion
AR	subj.Ipo	pmol/kg	amount of insulin in the portal vein
AS	subj.SRSH	ng/L/min	Delayed static glucagon secretion
AT	subj.H	ng/L	Plasma glucagon concentration
AU	subj.XH	ng/L	Delayed glucagon action on endogenous glucose production (EGP)
AV	subj.CPep1	pmol/L	C-peptide, 1st compartment
AW	subj.CPep2	pmol/L	C-peptide, 2nd compartment
AX	subj.Hsc1	mg/kg	1st compartment of subcutaneous glucagon
AY	subj.Hsc2	mg/kg	2nd compartment of subcutaneous glucagon
AZ	subj.Isc1	pmol/kg	1st compartment of subcutaneous insulin
BA	subj.Isc2	pmol/kg	2nd compartment of subcutaneous insulin
BB	subj.exEffect	(none)	exercise effect -- acts as a multiplier on Vmx
BC	subj.QatMealStart	mg	carbs in the stomach (subj.Qsto1 + subj.Qsto2) at start of most recent meal
BD	subj.QtotOfMeal	mg	total carbs in the meal which was most recently started
BE	subj.lastGlucTab	mg	carbs in the most recently delivered glucose tablet
BF	subj.timeSinceGlucTab	minutes	time since the most recently delivered glucose tablet
BG	insulin-Normal-Basal Use.Ip	pmol/kg	insulin in the plasma
BH	insulin-Normal-Basal Use.Il	pmol/kg	insulin in the liver

BI	insulin-Normal-Basal Use.Isc1	pmol/kg	1st compartment of subcutaneous insulin
BJ	insulin-Normal-Basal Use.Isc2	pmol/kg	2nd compartment of subcutaneous insulin
BK	insulin-Normal-Basal Use.IscSlowRel	pmol/kg	Not used – only applies to slow-release insulins
BL	insulin-Normal-Basal Use.IivSlowRel	pmol/kg	Not used – only applies to slow-release insulins
BM	insulin-Normal-Basal Use.lastBolusAge	minutes	Not used – only applies to slow-release insulins
BN	insulin-Normal-Basal Use.IOB	units	derived state value for total insulin on board
BO	insulin-Normal-Basal Use.IpConc	pmol/L	normal basal insulin concentration in plasma
BP	insulin-Normal-Bolus Use.Ip	pmol/kg	insulin in the plasma
BQ	insulin-Normal-Bolus Use.Il	pmol/kg	insulin in the liver
BR	insulin-Normal-Bolus Use.Isc1	pmol/kg	1st compartment of subcutaneous insulin
BS	insulin-Normal-Bolus Use.Isc2	pmol/kg	2nd compartment of subcutaneous insulin
BT	insulin-Normal-Bolus Use.IscSlowRel	pmol/kg	Not used – only applies to slow-release insulins
BU	insulin-Normal-Bolus Use.IivSlowRel	pmol/kg	Not used – only applies to slow-release insulins
BV	insulin-Normal-Basal Use.lastBolusAge	minutes	Not used – only applies to slow-release insulins
BW	insulin-Normal-Bolus Use.IOB	units	derived state value for total insulin on board
BX	insulin-Normal-Bolus Use.IpConc	pmol/L	normal insulin concentration in plasma
BY	insulin-Standard Long Acting.Ip	pmol/kg	insulin mass in plasma
BZ	insulin-Standard Long Acting.Il	pmol/kg	insulin mass in the liver

CA	insulin-Standard Long Acting.Isc1	pmol/kg	1st compartment of subcutaneous insulin
CB	insulin-Standard Long Acting.Isc2	pmol/kg	2nd compartment of subcutaneous insulin
CC	insulin-Standard Long Acting.IscSlowRel	pmol/kg	compartment for long-acting insulins that release into the 1st subcutaneous compartment, typically used for a basal bolus
CD	insulin-Standard Long Acting.IivSlowRel	pmol/kg	compartment for long-acting insulins that release slowly into the plasma
CE	insulin-Standard Long Acting.lastBolusAge	minutes	time since the most recently delivered bolus of Standard Long-Acting insulin
CF	insulin-Standard Long Acting.IOB	units	derived state value for total insulin on board
CG	insulin-Standard Long Acting.IpConc	pmol/L	standard long-acting insulin concentration in plasma
Concentration data, and IOB, for each user-defined insulin will be inserted here. For details on these data see Table 12.			
Data resulting from any included sensors will be inserted here. Each sensor will result in one column containing the output signal values for that sensor:			
	Sensors.<output signal name>, where “output signal name” is the name you supplied when configuring the sensor output.	Varies with sensor configuration.	Value of signal output from this sensor.
Data contributed by certain types of control elements will be inserted here. This will include user-defined output signals of JavaScript plugin control elements (see section 9.1.2):			
	Sensors.CEsig.<control element type>.<plugin name>.<signal name>. For JavaScript plugin control elements’ signals, “control element type” will be “Plugin Control Element - JavaScript.”	Depends on control element	Value of signal output from this control element.

	"plugin name" is the name given to the element (see section 6.4.6.2.3) "signal name" is the name of the signal defined within the control element. For JavaScript plugin control elements, this name is defined by the user (see the description of the outSignalName function in section 9.1.2).		
CH	simulation.minPastSimStart	(none)	minutes elapsed since the start of the simulation
CI	simulation.minPastMidnight	(none)	minutes elapsed since midnight
CJ	simulation.weekday	(none)	Day of week, 0 to 6, where 0 = Monday
CK	simulation.dayOfYear	(none)	Day of year, 0 to 366

Table 11: Result Data Definition

Column Label	Units	Description
insulin-<new insulin name>.Ip	pmol/kg	user-defined insulin mass in plasma
insulin-<new insulin name>.Il	pmol/kg	user-defined insulin mass in liver
insulin-<new insulin name>.Isc1	pmol/kg	1st compartment of subcutaneous user-defined insulin
insulin-<new insulin name>.Isc2	pmol/kg	2nd compartment of subcutaneous user-defined insulin
insulin-<new insulin name>.IscSlowRel	pmol/kg	compartment for long-acting user-defined insulins that release into the 1st subcutaneous compartment, typically used for a basal bolus
insulin--<new insulin name>.IivSlowRel	pmol/kg	compartment for long-acting user-defined insulins that release slowly into the plasma
Insulin--<new insulin name>.lastBolusAge	Minutes	time since the most recently delivered bolus of the user-defined insulin

insulin-<new insulin name>.IOB	units	derived state value for total user-defined insulin on board
insulin--<new insulin name>.IpConc	pmol/L	standard long-acting user-defined insulin concentration in plasma

Table 12: User-defined Insulin Concentrations

Appendix B: Running Simulations from the Command Line Interface

An individual simulation run can be performed from the command line interface provided configuration details have been previously saved to a configuration file (see Section 5.4). Use the following syntax:

For Windows:

```
DMMS.R <config file> <log file> <results dir>
```

Here, the following definitions apply:

- <config file> is the name of the configuration file that defines the simulation's "Rx Plan."
- <log file> is the name of a text file which should be used to log the overall completion status of the simulation. If the file already exists, it will be appended with the status for the current run.
- <results dir> is an optional parameter, which specifies the directory into which the simulation results (signal history files) are to be written. If this parameter is omitted, the results will be placed in the directory that would be used if the simulation were run from the GUI (based on the user settings made via the Tools/Settings… menu).

For Windows:

Note that this syntax will only work if the DMMS.R executable is in the command line interface's search path. Otherwise, the fully qualified path for the DMMS must be used, for example:

```
C:\Program Files\The Epsilon Group\DMMS.R\simulator\DMMS.R <config file> <log file> <results dir>
```

For each simulation run, the log file will have an entry identifying the configuration file, and, if the simulation cannot be started or completed, the reason for the failure, including:

- Failure to open or interpret the configuration file
- Invalid (e.g., incompatible) settings within the configuration (e.g., a type 2 subject in a simulation that must know the subject's ideal carb ratio)
- Failure to save the results (e.g., due to an attempt to write to a file that is already open in another program)
- Unexpected program failure during the simulation

In the Windows environment, batch files can be made to invoke multiple simulations, either in sequence or in parallel. For example, one that runs 3 simulations in sequence could contain the following:

```
dmms.r config_T1_test1.xml logMultiple.txt
```

```
dmms.r config_T1_test2.xml logMultiple.txt
dmms.r config_T1_test3.xml logMultiple.txt
```

A batch file that invokes the same 3 simulations in parallel would contain:

```
start dmms.r config_T1_test1.xml logParallel1.txt .\results1
start dmms.r config_T1_test2.xml logParallel2.txt .\results2
start dmms.r config_T1_test3.xml logParallel3.txt .\results3
```

Note that, when running in parallel, it is recommended that one of the following apply:

- An independent results directory should be explicitly specified for each invocation in the batch file (as in the examples above).
- The independent configurations should all have a different simulation name (specified, via the DMMS.R's GUI, under the "General" tab).

Otherwise, if we do not explicitly specify a results directory and all configuration files indicate the same simulation name, there are two possible outcomes (neither desirable):

- If we indicate "Automatically add subdirectories for results on each new simulation run" via the GUI's Tools/Settings… menu, then it will be difficult to know which simulation run corresponds to a given result folder, since the order in which the simulations complete cannot be anticipated.
- If we do not indicate "Automatically add subdirectories for results on each new simulation run," then results from the different runs will attempt to overwrite each other in the same directory.

Appendix C: MATLAB Signal History Data Format

The signal history of a DMMS.R simulation, when exported to a MATLAB data file (.mat file), contains two structure arrays with the following content:

- SimSettings: A 1x1 struct with the following fields:
  - simDurationMinutes: number of minutes in the simulation
  - useUniversalRandomSeed: logical value – true if the simulation was configured with the *Use random seed common to all elements* checkbox checked in the Randomization settings (see section 3.7).
  - universalSeed: numeric value -- the value of the base universal seed configured in the Randomization settings (see section 3.7).
- Subjects: A 1x***n*** struct array, where ***n*** is the number of subjects included in the simulation, with the following fields:
  - SubjectName: A character array containing the subject's name (e.g., 'adult#005')
  - Signals: A 1x1 struct with one field of ***p***x1 double values for each of the signal values selected for inclusion in the export (see section 7.7.2). Here, ***p*** is the number of minute-by-minute samples of any given signal in the simulation (1 more than the duration of the simulation, in minutes).
  - Params: A 1x1 struct with the following fields:
    - name: character array containing the subject's name
    - type1: logical value – true if the subject has type 1 diabetes

- CR: numeric value representing the subject's optimal carbohydrate ratio, in g/Unit of insulin. This value will only be meaningful for type 1 subjects
- CF: numeric value representing the subject's optimal correction factor, in mg/dL/Unit of insulin. This value will only be meaningful for type 1 subjects
- Gb: numeric value – the subject's basal blood glucose concentration, in mg/dL
- BW: numeric value – the subject's bodyweight, in kg
- OGTT (provided only for type 2 and prediabetic subjects): numeric value – the results of a 2-hour oral glucose tolerance test for the subject, in mg/dL
- dailyBasalInsulin (provided only for type 1 subjects): numeric value – the optimal daily basal insulin for the subject, in Units (i.e., the amount which holds the subject's blood glucose concentration at Gb when fasting)

- Survival: A 1x1 struct with the following fields:
  - Alive: logical value – true if the subject is alive at the end of the simulation
  - timeAlive: numeric value – the number of minutes that the subject remained alive in the simulation