SequenceTree4 User's Manual

by Jeremy Magland*

October 10, 2008

Contents

*If you have comments, questions, or error corrections, please send them to Jeremy.Magland@gmail.com

1 Introduction

1.1 What is SequenceTree?

SequenceTree is a pulse sequence programming environment for MRI. You can use it to design, create, visualize, and simulate pulse sequences. Ultimately you can export your sequences to run on a real MRI scanner. SequenceTree and related tools then allow you to view, reconstruct, and process the raw data coming back from the scanner.

SequenceTree is a balance between a what-you-see-is-what-you-get (WYSIWYG) visual user-interface and a fully flexible C++ coding environment. On the one hand, sequence parameters and structure can be changed using simple mouse clicks with the sequence display pane showing changes to the pulses in real time. On the other hand, an integrated C++ coding component provides the user with full control of functionality for all aspects of the pulse sequence, from top level loop nodes down to the lowest level scanner events. All of this takes place in a single graphical user interface – coding, compilation, debugging, and deploying all occur within a single application.

Whereas SequenceTree is scanner-independent (i.e. it is a stand-alone application with no components specific to a particular brand of MRI scanner), an interface module is available that allows SequenceTree sequences to be exported as Siemens-compatible source code. The exported code can then be compiled just like any research sequence programmed within the Siemens IDEA environment, and the resulting sequence will run on a clinical scanner.

1.2 Installation

For installation instructions, visit the online resources, including the SequenceTree Wiki: http://www.thesouthpoles.com/sequencetree/wiki.

The easiest and most reliable way to install SequenceTree is to download the latest Windows binary distribution, found at http://code.google.com/p/sequencetree/downloads/list. You will also need to download Qt 4.4 .dll package and extract it to the SequenceTree bin directory (unless you already have Qt 4.4 or later installed on your system). You can find this .dll package at the same website. You should also install MinGW (and add for example c:\mingw\bin to your path) in order to compile pulse sequences.

SequenceTree should also run on Linux and Mac, at this time you may need to contact the author for help compiling from source.

2 Navigating the User Interface

2.1 Pulse Sequence Files

Each SequenceTree pulse sequence is stored in a single text file with .sts extension. This file contains all parameter values and C++ code as well as other sequence-specific information. Use the file menu to open and save pulse sequence files.

2.2 Web Repository of Sequences

Use the "**Open Sequence From Web...**" option in the file menu to open one of the pulse sequences stored on the Web. You can also store your own sequences on the web using the **"Upload Sequence to Web..."** option in the file menu.

2.3 Sequence Compilation

When a pulse sequence is first opened, the simulation pane is blank because the sequence has not yet been compiled. Press F9 to compile the sequence. Compilation messages will appear in the "Compile" output tab. After compilation, the sequence events will appear in the simulation window. If sequence compilation fails, double click on the error in the Compile tab to go to the relevant source code line. If you need help getting your sequence to compile, contact the author.

2.4 Sequence Simulation

After your sequence has compiled successfully, a plot of sequence events will appear in the simulation window, which is visible when you have selected the "Parameters" tab. Use the horizontal scroll bar to scroll through the event blocks. To save space, SequenceTree automatically fills large empty time gaps with gray box time chunks (for lack of a better term). Also, to save simulation time, SequenceTree automatically loads one part of the sequence at a time, so you may notice short delays as you scroll through the sequence during sequence loading. As you change sequence parameters, the simulation pane will update automatically in real time (no need to recompile).

2.5 Changing Sequence Parameters

To change a sequence parameter, first click on the tree tab in the left pane. Next select the tree node that you want to modify. Ensure you are viewing the "Parameters" tab in the right pane. The parameters for the selected node will appear in the parameter view. When the sequence is compiled, green icons represent *active* parameters, or parameters that can be modified by the user (parameters with a red icon are controlled programmatically within the sequence). Double-click on a parameter to change the value. The simulation pane will update immediately to reflect the parameter change.

Note: The units column displays units for parameters for information purposes only.

2.6 Global Parameters

The "Global" parameters tab provides a convenient way to extract the most important parameters in your sequence, so other users don't need to search the sequence tree for which parameters to modify. To add a global parameter, right-click a sequence parameter (from the parameter view) and select "Link to Global Parameter". You can name the global parameter however you like (it doesn't need to coincide with the actual parameter name). If two sequence parameters are linked to the same global parameter, then they will automatically be linked together (i.e. changes in one will automatically take place in the other). In the "Global" tab, you can changed the order of the global parameters using the up/down push buttons.

2.7 Adding Notes

It's a good idea to document the your sequence. Click on the "Notes" tab and add text! Describe what the sequence is used for, what the parameters mean, and which parameters should be modified.

2.8 Custom RF Pulse Shapes

There are two approaches for using custom RF pulse waveforms in SequenceTree: the functional approach, and the sampled approach.

In the **functional approach**, you supply a C function that returns the waveform shape (real and imaginary parts):

- Step 1: Right-click on the RF pulse node (usually named RF in the tree), select "promote node", and select STSincRF.
- Step 2: Right-click on the RF node once again and select "Customize node". Provide a name for new node type, e.g. CustomRF.
- Step 3: Double-click on the RF node to view the C++ code. Modify the pulseShape function to return the shape of your waveform, as in the following example of a Gaussian pulse shape:

```
void CustomRF::pulseShape(double cycles,double &re,double &im) {

    re=exp(-cycles*cycles);
    im=0;

}
```

As in the STSincRF node type, you can add parameters to your object in order to control the shape of the pulse from the user interface.

The input parameter "cycles" is equal to time in seconds multiplied by bandwidth in Hz (so it is dimensionless), with cycles=0 representing the center of the pulse, or the effective excitation or refocusing time.

If you like, you can set pulse_duration in the prepare() method as a function of bandwidth (as is done in STSincRF). If you do not set pulse_duration, it will remain an active parameter in the user interface (i.e. modifiable by the user). You can also set reference_fraction, representing the fraction of the pulse duration that occurs before the center time (or reference time), or again you can leave it as an active parameter.

In the **sampled approach**, you can import a sampled waveform as a resource:

- Step 1: Open MR Pulse Tool from the Tools menu.
- Step 2: Design a pulse and save it to disk.
- Step 3: Import your new RF pulse waveform using the Resources menu. Your pulse will appear as a resource in the tree view.
- Step 4: Right-click on your RF pulse node, select "Promote node", and choose STSampledRF

- Step 5: Set the pulse_shape parameter of the RF pulse node to the name of your RF pulse resource, e.g. "mypulse.mrp".
- Step 6: Compile your sequence. Your sampled pulse should appear in the simulation.

Note that it is important that your imported pulse has the correct bandwidth value (for reference purposes) so that SequenceTree can scale (in time) your pulse appropriately according to the prescribed bandwidth for the pulse sequence.

2.9 Working with Resources

For the purpose of storing arrays of data (e.g. RF pulse waveforms or phase cycling tables), SequenceTree sequences can contain resources. These are displayed under the "Resources" node of the tree view (note: this node does not appear when the sequence has no resources). Use the Resource menu to add a new resource or import an RF pulse shape resource.

To edit a resource, double click on the resource node.

Resources can be accessed from within the C++ code (see Section 5.8). For an example of using resources, see STSampledRF.cpp in code/nodetypes/foundation.

3 SequenceTree Concepts

3.1 Hierarchical Sequence Structure

A SequenceTree sequence consists of a hierarchy of nodes. To each node is associated the following data:

- Node name
- Node type (name of C++ class)
- List of child nodes (a.k.a. children)
- List of parameters

Every sequence contains a single *Root* node at the top of the tree. The children of the Root are *Loop* nodes, controlling the looping or iterative structure of the sequence (the C++ type of a Loop node inherits *STLoop*). The immediate children of a Loop node are *Block* nodes, representing single event blocks (often single repetitions) in the pulse sequence (the C++ type of a Block node inherits *STChain*). The children of a Block node are so-called *Segment* nodes, representing a collection of possibly overlapping scanner events. For example, in the spoiled gradient echo example (spgr.sts), the Block node contains three segments (child nodes) named 'Excite', 'Acquire', and 'Rewind' (see Figure 1). The Excite node has the functionality of the excitation pulse, including a prephasing gradient (Prephase), a slice select gradient (SliceGradient) and a RF pulse (RF). Similarly, the Acquire node has the functionality of reading a single line of k-space, including phase encoding (Encode), a readout gradient (ReadoutGradient), and the readout event itself (Readout). Finally, the Rewind node is in charge of rewinding all imaging gradients and applying an optional spoiler gradient at the end of each repetition.

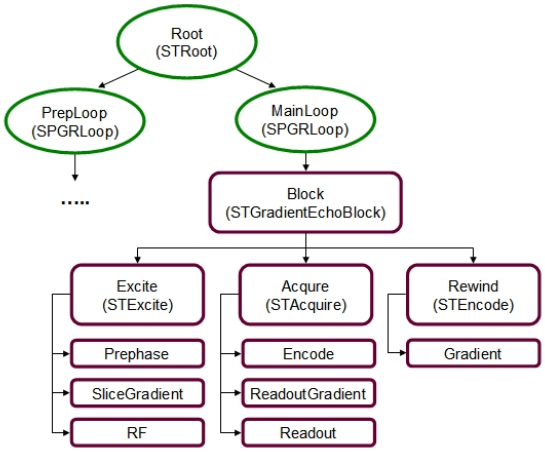

Figure 1: Hierarchical sequence structure for spoiled gradient echo example sequence (spgr.sts).

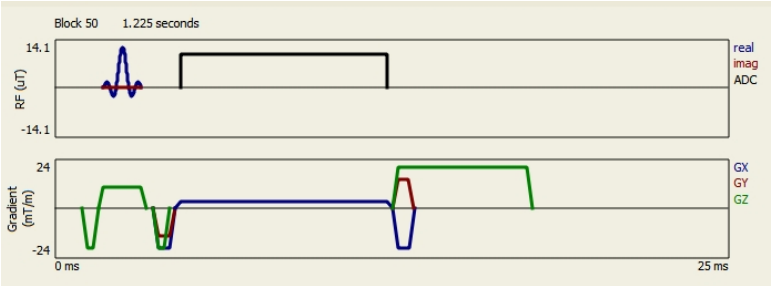

Figure 2: A single repetition in the balanced gradient echo example sequence (spgr.sts).

3.2 Parent/Child Communication

All functionality in a SequenceTree pulse sequence is implemented via parent/child communication between nodes. It is important for the programmer to understand the nature of this communication. Here are some basic principles of parent/child communication:

- Parents can access and modify the parameters of their children, grandchildren, etc. This is usually done within the prepare() method of a block or segment, or within the loopRun() method of a loop.
- Children should *not* modify (or even use) the values of the parameters of their parents, grandparents, etc. The one exception to this rule is that nodes sometimes need to modify the parameters of the Root node (for example to implement a global phase shift, or field-of-view shift during the sequence).
- Each node parameter is either *active* or *passive*. *Active* means that the user is free to modify the value of the parameter from within the parameter view of the user interface (active parameters are marked with a green icon). On the other hand, *passive* parameters (marked with a red icon) cannot be set by the user because they are modified programatically during the sequence, either by the node itself, or by a parent node, grandparent node, etc. SequenceTree automatically detects which parameters are active.
- Anytime a parameter is modified during the sequence, that parameter, as well as the corresponding node (and all of its ancestor nodes), are automatically marked as having been modified. This will signal the SequenceTree framework to call the prepare() method for the modified nodes are run (e.g. sent to the scanner). After prepare() is executed, the modification state is reset.

As an example of parent/child node communication, consider the spoiled gradient echo example. The loop node steps through a list of phase encode steps (e.g. PE1=-64 up to +63 in increments of 1). For each step, the loop node communicates with its child (Block) by setting

```
Block->kspace_echo=PE1_dir*PE1+PE2_dir*PE2;
```

specifying the desired location of kspace. Here *PE1_dir=[0,1,0]* is a vector representing the direction of phase encoding, and *PE2=0* (for 2D imaging). The Block then passes this information on to its children in the prepare() method:

```
Acquire->echo_moment=sequence()->kspace2moment(kspace_echo).
```

Here, *sequence()->kspace2moment()* is a built-in function that converts kspace locations to units of $[\mu T/mm] \cdot \mu s$ according to the prescribed field of view.

3.3 Node Parameters

The functionality of a node (as implemented in the corresponding C++ code) depends on the values of the node's parameters. Within the user interface, each node parameter has the following data:

- Type - STInteger, STReal, STVector3, or STIterator

- Name - name of the parameter
- Value - a string representation of the parameter value
- Units - for information purposes only

3.4 Customizing Nodes

There are three kinds of node types in a SequenceTree sequence: foundation node types, user node types, and custom node types. The C++ code for foundation and user node types can be viewed but not modified from within SequenceTree (the code for these types can be found in the code/nodetypes/user and code/nodetypes/foundation directories). On the other hand, the C++ code for custom node types can be modified from within SequenceTree (double-click on the node to edit the code). Nodes with foundation, user, and custom node types are called foundation, user, and custom nodes, respectively [Please excuse the somewhat trivial statement]. Foundation and user nodes can be distinuished from custom nodes in the tree view according to the node icons.

If you need to modify the functionality of a foundation node, you will first need to customize it. Right-click on the node and select "Customize Node". Note that you must first customize the parent node. It is highly encouraged that you only customize nodes toward the top of the tree, such as Loop and Block nodes. This is because the code for low-level nodes is subject to slight modifications in the framework over time, thus they should remain foundation node types whenever possible.

3.5 Iterators

The looping or iterative functionality of a SequenceTree sequence is accomplished using iterators in Loop nodes (as mentioned above, Loop nodes are the immediate children of the sequence's Root node). An iterator is simply a parameter of type *STIterator*, consisting of minimum/maximum values and a step size (or increment). When such a parameter is present in a Loop node, SequenceTree will automatically step through the values of the iterator one by one during sequence execution, calling the loopRun() method at each step. The code within loopRun() is responsible for setting the parameters of its children appropriately, and then calling the run() method of each child at the desired times.

Nested iteration is accomplished by including multiple iterator parameters within a single Loop node. The top-most iterator is always the inner-most nested loop (i.e. the fastest moving). Modification of the nesting order of the iterators can be easily accomplished by simply reording the iterator parameters in the class code.

3.6 Linking Nodes

For convenience, multiple nodes can be linked together in the user interface. This is useful for implementing preparation repetitions. For example, in the spoiled gradient echo example (spgr.sts), the Block of the PrepLoop is linked to the Block of the MainLoop. Therefore, whenever a parameter of either Block (or their descendents) is modified by the user, the same change takes place in the linked Block. It's a way to synchronize the parameters of nodes.

To link two nodes, select the first node, hold down the control button, and left-click the second node.

To unlink nodes, right-click on the nodes and select "unlink node".

3.7 Promoting Nodes

4 Programming Tutorials

4.1 Change sequence parameters

1. Open sequence: File->open, c:\st-4.1.10-standalone\sequences\spgr.sts
2. Compile the sequence by clicking the 'Compile' button
3. Scroll through the sequence event blocks using the scroll bar in the simulation view
4. Click on the 'tree' tab Expand the tree to MainLoop->Block->Acquire->Readout
5. Change the number of readout points by double-clicking on the 'N' parameter and entering '128'
6. Minimize the echo time by selecting the MainLoop->Block node, right-clicking the TE parameter, and selecting 'Minimize Parameter' (Warning: if this causes a sequence error, increment echo time until the error disappears)
7. Change TR by double-clicking the TR parameter and entering '20000'
8. Change the number of phase encodes by selecting the MainLoop node, double-clicking the PE1 parameter, and entering '-64:1:63'
9. Change the spoiler moment using the moment parameter of MainLoop->Block->Rewind
10. Experiment by modifying any sequence parameter that has a green icon, or try modifying the parameters in the 'Global' tab.

4.2 Switch between selective and non-selective excitation pulse

1. Open the sequence used in the previous example
2. Change the RF pulse shape by right-clicking the MainLoop->Block->Excite->RF node and selecting 'Promote node...'.
3. Choose STRF for a simple rectangular pulse.
4. Do the same for the PrepLoop->Block->Excite->RF node.
5. Recompile using the 'compile' button Make the pulse non-selective by modifying the following parameters of MainLoop->Block->Excite:
   gradient_amplitude=(0,0,0)
   slice_thickness=0
6. Set the duration of the rectangular pulse using the pulse_duration parameter of MainLoop->Block->Excite->RF.

4.3 Create a spin-echo sequence with phase encoding before the refocusing pulse

1. Open sequence: File->open, c:\st-4.1.10-standalone\sequences\spgr.sts
2. Save sequence as spin-echo.sts
3. Remove the preparation repetitions by right-clicking on PrepLoop and selecting 'Delete Node'
4. Customize the loop node by right-clicking on MainLoop and selecting 'Customize node'.
5. Enter SpinEchoLoop for the class name.
6. Create a spin echo by right-clicking on MainLoop->Block and selecting 'Change node type'. Then choose STSpinEchoBlock.
7. Compile the sequence by clicking the 'compile' button.
8. Increase TE to 12000 to remove the sequence errors.
9. Customize the block by right-clicking on MainLoop->Block and selecting 'Customize node'.
10. Edit the code for the block by double-clicking the MainLoop->Block node
11. Add the following line of code before 'ST_CHILD(STRefocus,Refocus)':
    ST_CHILD(STEncode, Encode)
12. Add the following line before 'return STChain::prepare();':
    Encode->moment=sequence()->kspace2moment(kspace_echo)*(-1);
13. Recompile

4.4 Implement 3D scanning and switch the order of the phase encoding

1. Open sequence: File->open, c:\st-4.1.10-standalone\sequences\spgr.sts
2. Save the sequence as spgr-avg.sts
3. Add 3D phase encoding by setting PE2 parameter in MainLoop to '-4:1:3'
4. To switch the order of phase encoding you first need to customize the MainLoop (right-click on the main loop and select 'Customize node')
5. Edit the code for the main loop by double-clicking the MainLoop node
6. Swap the following two lines of code using copy-paste
    ST_PARAMETER(STIterator,PE1,0:1:0,)
    ST_PARAMETER(STIterator,PE2,0:1:0,)
7. Recompile and that's it! Note that the order in which STIterator parameters are specified in the loop determines the nesting order of the iteration.

4.5 Implement 'Averaging' in a Basic Sequence

1. Open sequence: File->open, c:\st-4.1.10-standalone\sequences\spgr.sts
2. To add averaging, you first need to customize the MainLoop (right-click on the main loop and select 'Customize node')
3. Edit the code for the main loop by double-clicking the MainLoop node Add the following line of code after the existing iterator parameters
   ST_PARAMETER(STIterator,AVG,1:1:1,)
4. Recompile and that's it! Click on the parameter tab and change the 'AVG' parameter to '1:1:10' for ten averages.

4.6 Create a sequence from 'scratch' and add preparation repetitions

1. Create new sequence: File->new
2. Add a loop node by right-clicking on the Root node and selecting 'Add Loop', choose STCartesianLoop and name it 'MainLoop'. Then compile.
3. Customize your sequence according to your application (see other examples)
4. Insert a loop for prep. reps. by right-clicking on MainLoop and selecting 'Insert Loop'. Be sure to choose the same node type as MainLoop, and name the new node 'PrepLoop'. Then recompile.
5. Link the block nodes between the MainLoop and PrepLoop by selecting MainLoop->Block, Holding down the [Ctrl] key, and left-clicking PrepLoop->Block.This synchronizes parameters between the blocks.
6. Specify the number of preparation repetitions by modifying the parameters of the PrepLoop node.

4.7 Implement centric-ordered phase encoding

1. Open sequence: File->open, c:\st-4.1.10-standalone\sequences\spgr.sts and save sequence as spgr-centric.sts
2. Customize the loop node by right-clicking on MainLoop and selecting 'Customize node'. Enter CentricLoop for the class name.
3. Edit the code by double-clicking the MainLoop->Block node Replace the loopRun() function with the following code:

```
bool CentricLoop::loopRun() {
int iPE1=(int)PE1;
int pe1=0;
if (iPE1%2==0) pe1=-iPE1/2;
else pe1=(iPE1+1)/2;
Block->kspace_echo=PE1_dir*pe1+PE2_dir*PE2;
Block->setReadoutIndex(0,pe1+PE1.numSteps()/2);
return Block->run();
}
```

4. Finally, change the PE1 parameter to '0:1:127' (for 128 lines)

5 Programmer's Reference

5.1 SequenceTree Macros

SequenceTree Macros are critical for communicating information about node types to the user interface, so that parameters and parent/child relationships can be displayed properly even before sequence compilation. These macros must be included in the C++ source file for each node type.

- **ST_CLASS([Class name], [Base class name])** - This macro must be included within the constructor of each SequenceTree class.
- **ST_PARAMETER([Type], [Name], [Default value], [Units])** - define a parameter. Include within the constructor.
- **ST_CHILD([Type], [Name])** - define a child. Include within the constructor.
- **ST_DEFAULT([Parameter], [Value])** - set a default value for a parameter of a child or descendent node. Include within the constructor.
- **ST_ALIGN([Child name], [Alignment type], [Time offset], [Relative child index])** - set the alignment of a child of a *STChain* node. Include within prepare() method. See Section 5.4 for more information.

Note that in the C++ header file of each node type, there is a section marked "this section generated by SequenceTree – do not edit". SequenceTree uses this section to automatically generate class variables corresponding to parameters and children defined using the ST_PARAMETER and ST_CHILD macros in the source file. Any edits to this section will be discarded.

5.2 Access to parameters and children

By examining the header files of SequenceTree node types (recall that all node types are standard C++ classes), one can see that all node parameters and children are simply member variables of the class. Parameters of type STReal and STInteger can, for the most part, be treated like C++ types, double and int respectively. Parameters of type STIterator can also be accessed as a double floating point value in the standard manner, where the value represents the current value of the iterator at a particular time in the sequence. The minimum, maximum, and increment information for an iterator can also be accessed using the minimum(), maximum(), and step() methods.

Note that children are pointer member variables, whereas parameters are standard member variables. For example, the flip angle of the RF pulse in an object of type STGradientEchoBlock can be accessed from within the block using the following notation:

```
double phi = Excite->RF->flip_angle;
```

5.3 Loop node implementation

The following methods can be overloaded in the implementation of a loop node (base class STLoop).

The **prepare()** method is called at the beginning of the sequence. One-time initialization of variables, parameters, and parameters of child and descendent nodes should occur within this function.

The **loopRun()** method is called at each step of the iterators. Within this function, parameters of child and descendent nodes should be set according to the current iterator state, and the run() method of each child should be called at the appropriate times.

5.4 Block node implementation

The following methods can be overloaded in the implementation of a block node (base class STChain).

The **initialize()** method is called at the beginning of the sequence. One-time initialization of variables, parameters, and parameters of child and descendent nodes should occur within this function.

The **prepare()** method is called immediately before the block is run whenever a parameter of this node, a child node, or a descendent node has been modified. In this function, the parameters of child nodes should be set, the alignment of child nodes should be set, and finally STChain::prepare() should be called.

For nodes types that inherit STChain, the **ST_ALIGN** macro is used to set the timing for the children. The syntax for this macro is:

> **ST_ALIGN([Child name], [Alignment type], [Time offset], [Relative child index])**

The alignment type can be one of the following values:

- **ST_ALIGN_LEFT (default)** - Align this child as far to the left as possible, without overlapping the previous child or the beginning of the block. Then offset the start time by "Time offset".
- **ST_ALIGN_RIGHT** - Align this child as far to the right as possible, without overlapping the next child or the end of the block. Then offset the start time by "Time offset".
- **ST_ALIGN_RELATIVE** - Align this child relative to the child at (zero-based) index "Relative child index". Time offset represents the time between the reference time of the relative child and the reference time of this child (see description of referenceTime() below).
- **ST_ALIGN_ABSOLUTE** - The reference time of this child will occur at "Time offset" relative to the beginning of this block.

Note that the "Relative child index" parameter is only relevant for the ST_ALIGN_RELATIVE alignment type.

Each child has a reference time, representing the relevant time for that child. For example, the reference time for the Excite segment usually occurs at the peak of the RF excitation pulse. Similarly, the reference time for the Acquire segment occurs at the *echo time* of the readout. These reference times can be accessed programmatically using, for example, RF->startTime()+RF->referenceTime().

5.5 RF Pulse implementation

5.6 Segment node implementation

Note: *The vast majority of code modifications should occur for Loop nodes, Block nodes, and RF pulse nodes (see Sections 5.3, 5.4, and 5.5). However, it is sometimes necessary to customize a different type of node, although this should be avoided whenever possible.*

The relative start times of the child nodes can be accessed using child(j)->relativeStartTime(), and set using child(j)->setRelativeStartTime(double).

The following methods can be overloaded in a segment node.

The **initialize()** method is called at the beginning of the sequence. One-time initialization of variables, parameters, and parameters of child and descendent nodes should occur within this function.

The **prepare()** method is called immediately before the segment is run whenever a parameter of this node, a child node, or a descendent node has been modified. In this function, the parameters of child nodes should be set, and *setModified(false)* should be called in order to reset the modification state.

The **run()** method is called when the segment is to be run. In the default implementation, run() is called for each child node.

The **duration()** method must return the duration (in microseconds) of the segment.

The **SAR()** method must return the energy of the segment. In the default implementation, it returns the sum of SAR() for the children.

The **referenceTime()** method must return the reference time of the segment in microseconds (relative to the start time of the segment). This information is used by STChain objects to set relative alignments. In the default implementation this function returns child(ind)->relativeStartTime()+child(ind)->referenceTime(), where ind=referenceChildIndex(). The reference child index can be set using setReferenceChildIndex(int).

The **terminalMoment()** method must return the 3D moment at the end of this segment as a function of the moment at the start of the segment, as given by initialMoment(). This information is used by SequenceTree to keep track of the k-space location in order to automatically prephase or rewind gradients appropriately. For example, a segment consisting purely of gradients should return initialMoment()+M, where M is the total gradient moment in the segment, whereas a segment containing an a refocusing RF pulse should return -initialMoment()-M1+M2, where M1 and M2 are the gradient moments occuring before and after the refocusing time respectively.

The **totalGradientMoment()** method should return the total 3D moment of the gradients in the segment, ignoring RF pulses. This information is used by SequenceTree to implement automatic FOV shifts. In the default implementation, it returns the sum of totalGradientMoment for the children.

The **gradientStartTimes()** and **gradientEndTimes()** must return the 3D gradient start and end times in microseconds (relative to the start time of the segment). This information is used by STChain objects to achieve optimal alignments. Note that RF pulse and readout events should never overlap with external segments. Therefore, gradientStartTimes() should never exceed the start time of RF pulses and readout events. Similarly, gradientEndTimes() should never occur before the end time of RF pulses or readout events in the segment.

5.7 Access to Root Node

The root node of the sequence can be accessed in the code of any node using the **sequence()** function. This is useful for offsetting the global transmit/receive phase during the sequence (e.g. for RF spoiling), for shifting the field-of-view (e.g. for multi-slicing), or for accessing special functions associated with the root node (e.g. sequence()->kspace2moment()). For example, the following code can be found in the loop node of spgr.sts (note that *phase_shift* is a parameter of the root node).

```
bool SPGRLoop::loopRun() {
    if (RF_spoiling) {
        sequence()->phase_shift=rand()%360;
    }
    Block->kspace_echo=PE1_dir*PE1+PE2_dir*PE2;
    return Block->run();
}
```

5.8 Resources

For the purpose of storing arrays of data (e.g. RF pulse waveforms or phase cycling tables), SequenceTree sequences can contain resources. These are displayed under the "Resources" node of the tree view (note: this node does not appear when the sequence has no resources). For information about working with resources in the user interface, see Section 2.9.

To access a resource within C++ code, use the following notation:

```
STResource *R=sequence()->resource("resource_name");
```

or

```
STResource *R=sequence()->resource("resource_parent->resource_child");
```

Be sure to verify that R is not NULL before accessing this pointer! Resources can contain a single real number or an array of numbers. To access a single number use

```
double value=R->getDouble();
```

To access an array use

```
double value=R->getDoubleList(index);
```

The size of the array is given by

```
long N=R->doubleListCount();
```

For an example of using resources, see STSampledRF.cpp in code/nodetypes/foundation.

6 Foundation Node Type Reference

6.1 Foundation Loops

- **STCartesianLoop** - Basic loop for standard 2D or 3D Cartesian scanning of k-space
  - PE1 and PE2 (unitless) are the phase encode iterators
  - PE1_dir and PE2_dir (unitless) are the directions of phase encoding
  - readout_dir (unitless) is the readout direction
- **STRadialLoop** - Basic loop for standard 2D Radial (or 3D hybrid radial) scanning of k-space
  - angle (degrees) is the iterator controlling the view angle.
  - readout_dir1 and readout_dir2 (unitless) are the two orthogonal directions for 2D radial scanning.
  - PE (unitless) is the iterator controlling the optional Cartesian phase encoding for 3D hybrid radial.
  - PE_dir (unitless) is the direction of the Cartesian phase encoding for 3D hybrid radial.

6.2 Foundation Blocks

- **STGradientEchoBlock** - Basic block (single repetition) for simple gradient echo acquisitions
  - TE (microseconds) is the echo time, or the time between excitation and acquisition.
  - TR (microseconds) is the duration of the block.
  - kspace_dir (unitless) is the direction in k-space of the readout.
  - kspace_echo (unitless) is the location in k-space at the echo time.
  - excite_time (microseconds) is the start time of the excitation pulse segment.
- **STSpinEchoBlock** - Basic block (single repetition) for simple spin echo acquisitions
  - TE (microseconds) is the echo time, or the time between excitation and acquisition.
  - TR (microseconds) is the duration of the block.
  - kspace_dir (unitless) is the direction in k-space of the readout
  - kspace_echo (unitless) is the location in k-space at the echo time.

6.3 Foundation Segments

- **STExcite** - Fundamental segment for RF pulse excitation, including an optional slice select gradient and an optional prephasing gradient. For a non-selective pulse, set gradient_amplitude=(0,0,0) and slice_thickness=0.
  - gradient_amplitude ($(\mu T/mm) \cdot \mu s$) is the amplitude for the slice select gradient, or (0,0,0) for a non-selective pulse.
  - slice_thickness (mm) is thickness of the selected slice, or 0 for a non-selective pulse.
  - bandwidth (Hz) is the bandwidth of the pulse. It is automatically set by the framework for selective pulses, according to gradient_amplitude and slice_thickness.
  - prephase (0 or 1) determines whether a prephasing gradient is included before the selective pulse. This is useful for balanced steady-state imaging.
  - Note: Use parameters of the *SliceGradient* child to set the ramp time for the slice select gradient.
  - Note: Control the shape, flip angle, and other aspects of the excitation pulse using the *RF* child.
- **STAcquire** - Fundamental segment for signal readout, including a readout gradient and automatic encoding gradients.
  - echo_moment ($(\mu T/mm) \cdot \mu s$) is the desired location in k-space at the reference time of the acquisition. Prephasing encoding gradients will automatically be set to achieve echo_moment at the appropriate time.
  - moment_per_point ($(\mu T/mm) \cdot \mu s$) is the desired k-space step between successive readout points. This controls the amplitude of the readout gradient.
  - Note: Use parameters of the *ReadoutGradient* child to control the ramp times of the readout gradient.
  - Note: Use parameters of the *Readout* child to control the dwell time and the number of points to acquire.
- **STEncode** - Fundamental segment for moving to a specific location in 3D k-space using a collection of X, Y, and/or Z gradients. A node of this type is often used at the end of repetitions to implement a spoiler gradient.
  - moment ($(\mu T/mm) \cdot \mu s$). If do_rewind=1, this is the desired location in k-space to move to. If do_rewind=0, this is the absolute moment to encode - which will offset the current k-space location.
  - do_rewind (0 or 1) specifies whether or not *moment* is defined relative to the center of k-space.
  - Note: Use the *maxamp* and *ramprate* parameters of the *Gradient* child to control the amplitude and ramp time of the gradients.

- **STRefocus** - Fundamental segment for a refocusing RF pulse, including an optional slice select gradient and optional crusher gradients. For a non-selective pulse, set gradient_amplitude=(0,0,0) and slice_thickness=0.
  - gradient_amplitude ($(\mu T/mm) \cdot \mu s$) is the amplitude for the slice select gradient, or (0,0,0) for a non-selective pulse.
  - slice_thickness (mm) is thickness of the selected slice, or 0 for a non-selective pulse.
  - bandwidth (Hz) is the bandwidth of the pulse. It is automatically set by the framework for selective pulses, according to gradient_amplitude and slice_thickness.
  - crusher_moment ($(\mu T/mm) \cdot \mu s$) determines the size and direction of each crusher gradient surrounding the RF pulse.
  - flip_angle (degrees) is the flip angle of the RF pulse
  - Note: Use parameters of the *SliceGradient* child to set the ramp time for the slice select gradient.
  - Note: Control the shape and other aspects of the refocusing pulse using the *RF* child.
  - Note: Use the *maxamp* and *ramprate* parameters of the *Crusher1* and *Crusher2* children to control the amplitude and ramp time for the crusher gradients.
- **STRF** - A single RF excitation pulse. Nodes of type STRF are usually not direct children of block nodes, but occur as a child of an STExcite or STRefocus node. This is also the base class for other RF pulse objects, including STSampledRF, STSincRF and custom RF types.
  - flip_angle (degrees) is the flip angle for the pulse.
  - pulse_duration (microsec) is the total duration of the pulse. In the case of STSincRF, for example, this parameter is automatically set according to the *num_lobes_left*, *num_lobes_right*, and *bandwidth* parameters.
  - reference_fraction (unitless) is the fraction of the pulse duration that occurs before the reference time. This is used internally by SequenceTree for purposes of timing events and rephasing/prephasing gradients.
  - bandwidth (Hz) is the bandwidth of the pulse. This parameter is often set by the parent object (STExcite or STRefocus) in the case of a spatially selective pulse, as a function of prescribed gradient amplitude and slice thickness.
  - time_step (microseconds) is the time step for sampling the RF waveform (default=10 microseconds)
  - phase (degrees) is a phase offset to the pulse, independent of other internal phase offsets.
  - frequency (Hz) is a frequency offset to the pulse, independent of other internal frequency offsets.

  - pulse_type (1, 2, or 3) specifies whether the pulse is an excitation pulse (1), a refocusing pulse (2), or other (3). SequenceTree uses this information to keep track of k-space location throughout the sequence for purpose of automatically setting gradients for dephasing and prephasing.
  - gradient_amplitude (mm) is used internally by SequenceTree (when STRF is a child of STExcite or STRefocus) for purpose of automatic FOV shifts.

- **STReadout** - A single readout event. Nodes of type STReadout are usually not direct children of block nodes, but occur as a child of an STAcquire node.
  - dwell_time (microseconds) is the dwell time for the readout (time spent acquiring a single point).
  - N (unitless) is the number of points to be read.
  - reference_fraction (unitless) is the fraction of the readout duration that occurs before the reference (or echo) time. This is used internally by SequenceTree to set timing and automatically compute gradient moments for prephasing and dephasing.
  - actual_reference_fraction (unitless) is for information purposes only, indicating the effective reference fraction after adjustments have been made according to the gradient raster time.
  - reference_sample (unitless) is for information purposes only, indicating the effective sample number at echo time.
  - phase (degrees) is a phase offset to the readout, independent of other internal phase offsets.
  - frequency (Hz) is a frequency offset to the readout, independent of other internal frequency offsets.
  - gradient_amplitude (mm) is used internally by SequenceTree (when STReadout is a child of STAcquire) for purpose of automatic FOV shifts.

- **STGradientAmp** - A 3D gradient object with fixed amplitude used during a selective RF pulse or during a readout. Nodes of type STGradientAmp are usually not direct children of block nodes, but occur as a child of STAcquire, STExcite, or STRefocus.
  - ramp_time_1 and ramp_time_2 (microsec) are the rampup and rampdown times, respectively.
  - plateau_time (microsec) is the plateau time.
  - amplitude ($\mu T/mm$) is the 3D amplitude.

- **STGradientMom** - A 3D gradient object with fixed zeroth moment used during used for phase encoding, prephasing, dephasing or spoiling. Nodes of type STGradientMom should almost always occur as a child of an STEncode object.
  - ramp_times_1, plateau_times, ramp_times_2, and start_times are all used internally by SequenceTree. Rather than setting these parameters directly, STEncode uses the setMoment() method of STGradientMom. Note that STGradientMom should almost always occur as a child of an STEncode object.

- alignment (0, 1, or 2) specifies whether the gradients are to be aligned to the left (0), right (1), or center (2) relative to one another.
- always_min_dur (0 or 1) specifies whether the duration should always be minimal. By default this parameter is set to 0, indicating that a fixed duration is used throughout the sequence (good for phase encoding tables).
- maxamp ($\mu T/mm$) is the maximum gradient amplitude to be used. By default this parameter is 0, in which case the maxamp parameter of the root node is used instead.
- ramprate ($(\mu T/mm)/\mu s$) is the maximum slew rate to be used. By default this parameter is 0, in which case the ramprate parameter of the root node is used instead.

7 Virtual Scanner (MRI Simulator)

[To be finished]

8 Export to Scanner

8.1 Export to Siemens Scanner

SequenceTree sequences can be exported as IDEA-compatible source code, compiled within IDEA framework, and then run on a Siemens Magnetom MRI scanner.

If you work at the Hospital of the University of Pennsylvania, some convenient tools exist for exporting to the scanner. These are accessible by logging in remotely to the MaglandMRI workstation. Contact Jeremy for more information.

Note that the export-to-Siemens module is not included with the open source distribution of SequenceTree. Contact Jeremy to obtain this module.

In general, follow these instructions for exporting your sequence.

- Open your sequence within SequenceTree (make sure it compiles) and select "Tools->Export to Scanner".
- Specify the desired name of the sequence, the export directory (must exist, no spaces please), and the Siemens software version (e.g. siemens_vb15).
  - Note: If you have IDEA installed on your computer, you will probably want to specify the export directory as something like: z:\n4\pkg\MrServers\MrImaging\seq
- After clicking OK, SequenceTree will create a subdirectory with source code for your sequence. If IDEA is installed on your computer, you can compile this sequence just like any other research sequence (i.e. using SDE shell). Otherwise, you can copy the exported directory to a different computer that has IDEA installed.
- If the sequence fails to compile, first consult the list of possible reasons below. If you can't find the problem in that list, try compiling a simpler sequence, such as spgr.sts from the web. If you continue to have trouble, please report the issue to the author.

Some possible reasons for compilation errors (from within IDEA)

- You forgot to "return true" in a function that has return type "bool". For example, in loopRun(). This is a frequent problem, because some compilers (such as MinGW) do not complain about this issue, whereas others consider it as an error.

8.2 Interface to Siemens User Interface at Scanner

SequenceTree always automatically connects to Field of View and Field-of-View Position, including oblique scan positions. Just remember that X, Y, and Z directions in SequenceTree always correspond to Readout, Phase Encode, and Slice directions at the scanner, respectively. Currently, you cannot change the dimensions of the field of view at the scanner, but this functionality will be added in the future.

All other parameters (TE, TR, resolution, bandwidth, etc.) are hardcoded into the sequence and cannot be modified at the scanner, unless they are manually added as scanner directives in the notes tab. You can add parameters to the special card of the scanner user interface using the following procedure:

Step 1: Select the sequence parameter you want add to the special card, and make sure it is linked to a global parameter (see Section 2.6). This parameter must be of type STReal or STInteger (STVector and STString parameters cannot be exported at this time).

Step 2: In the notes tab, add directive lines in the following format:

```
== Siemens User Interface Directives =====
UIDOUBLE NAME=TE RANGE=1:0.1:100 LABEL="TE" UNITS="ms" FACTOR=0.001
UIDOUBLE NAME=TR RANGE=10:10:3000 LABEL="TR" UNITS="ms" FACTOR=0.001
UIDOUBLE NAME=flip_angle RANGE=0:5:180 LABEL="Flip angle" UNITS="deg"
UIDOUBLE NAME=slice_thickness RANGE=2:2:50 LABEL="Slice thickness" UNITS="mm"
UICHECKBOX NAME=RF_spoiling LABEL="RF Spoiling"
============================
```

- **The fields in each line must be separated by tabs.**
- **You are limited to 13 special card parameters.**
- The **NAME** field must match the name of the global parameter.
- The **RANGE** field specifies the range of values for the parameter in the format min:step:max.
- The **LABEL** field specifies the label for the parameter on the user interface.
- The (optional) **UNITS** field specifies the units for the parameter on the user interface
- The (optional) **FACTOR** field can be used to convert between units. For example, if the global parameter TE is stored in microseconds in SequenceTree, then FACTOR=0.001 converts it to milliseconds on the Siemens user interface.
- **UIDOUBLE** can be used for parameters of type STReal or STInteger
- **UICHECKBOX** can be used for parameters that take on values 0 and 1 (0 means unchecked, 1 means checked)

- **Important:** Parameters with certain names, including "TE", "TR", and "flip_angle", are automatically added to the standard locations on the user interface, and are not added to the special card. It is required that TE and TR have units of milliseconds, and that flip_angle has units of degrees.

8.3 Cardiac/Respiratory Gating

To add cardiac or respiratory gating functionality to your sequence, put the following code somewhere within the loopRun() function:

```
int channel_number=1;
long halt_delay=100; //microseconds
scanner()->scannerCommand("waitForTrigger",&channel_number,&halt_delay);
```

When this code is encountered, during the sequence, the scan will pause until it receives a trigger from the channel specified by "channel_number". Once the trigger is received, the scan will pause for additional delay (according to halt_delay in microseconds) and then resume with the sequence.

To set up cardiac or respiratory gating, you must also specify the trigger source from the appropriate user interface card at the scanner.

Within the SequenceTree GUI, "waitForTrigger" events appear as magenta rectangles in the sequence simulation view.

8.4 SequenceTree Controller (Parsing Raw Data)

The SequenceTree Controller is an executable program that is distributed with SequenceTree (bin\st4controller.exe on Windows). This program is very useful for automatically parsing raw data at the scanner. To automatically parse raw data, use the following steps:

- When you transfer your sequence binary files to the scanner (e.g. seqname.dll, seqname.o750, etc.) be sure to also transfer the [seqname].sts and [seqname].exe files (all should go in the %CustomerSeq% folder on the Host computer). These files can be found in the exported sequence directory (i.e. where the sequence was compiled within SDE shell).
- Run st4controller.exe (found in bin directory of SequenceTree) on a companion computer that can see the Host computer over the network. Select "Tools->Controller Configuration" within st4controller and specify the directories for sequence binaries and raw data (it's a good idea to mount the host computer on a network drive).
- In the "Sequences" tab you should see a list of the pulse sequences you have exported to the scanner.
- For VA30 and similar release versions, make sure the rawdata2disk flag has been set. On VB15 and similar release versions, use the "twix" tool to save raw data to the raw data directory specified in st4controller.exe.
- Immediately after you run a scan (and after the data has been saved to the raw data directory), the raw data file should appear in the "Raw Data" tab.

The program shows the size of the file, which pulse sequence was run, and how many receive channels were used. Click on the new raw data file and click the "Parse" button. This will open the results viewer with your raw data in .mda (multi-dimensional array) format. Double-click a file to view the raw k-space data array. You can use the Process menu to perform simple reconstruction using 1D, 2D, or 3D FFT. For more advanced processing, use ChainLink, Matlab, or other program (hint: ChainLink is :) recommended).

- From within the results viewer click "Save Experiment..." to save your data to a folder.
- It's a good idea to erase raw data files when you are done with them. Click the "Open Folder" button to open the raw data file, and then delete the old files.

Alternatively, you can read the raw data using the "Tools->Read raw data..." menu option within SequenceTree. For this method you must open the appropriate sequence, compile it within SequenceTree, and make sure the correct parameters are set. The advantage of st4controller is that it automatically detects the sequence and sequence parameters that were actually run.

Note: When using st4controller, be sure to parse the raw data immediately after you run the scan (and before you run the next scan).

Note: You can also use st4controller on a personal workstation (not at the scanner). Just place the [seqname].sts, [seqname].exe, and [seqname].st4parameters files in a designated sequence directory, and the raw data in a designated raw data directory, and set the st4controller configuration appropriately. The [seqname].st4parameters file is created automatically during the scan (at the scanner). But you can also make this file on your own computer – just open and compile the sequence within SequenceTree, set the appropriate parameters, and then rename the tmp\sequence_parameters_in.txt file (found in SequenceTree folder).

9 MR Pulse Tool

[To be finished]

Index