Synapse API Manual

Updated: 3/13/18

Synapse API Manual

Tucker-Davis Technologies
11930 Research Circle
Alachua, FL 32615 USA
Phone: (+1)386.462.9622
Fax: (+1)386.462.5365

Notices

The information contained in this document is provided "as is," and is subject to being changed, without notice. TDT shall not be liable for errors or damages in connection with the furnishing, use, or performance of this document or of any information contained herein.

The latest versions of TDT documents are always online at www.tdt.com/support.htm.

Synapse API Manual Table of Contents

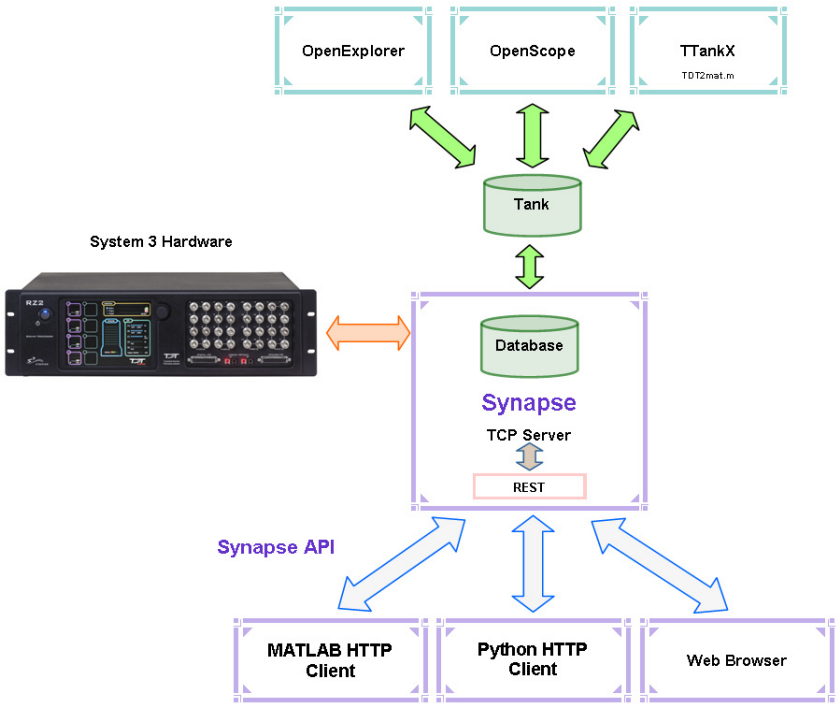
Synapse API Overview

Synapse API is a series of methods that can be used to develop Synapse companion applications using Matlab, Python, C++ or any language that can make HTTP requests. It provides access to System 3 real-time processing devices during an experiment, and lets you control Synapse remotely, mimicking many behaviors a user can perform.

Client applications developed using Synapse API can:

- Control the system mode
- Set tank and block names
- Set subject, user, and experiment, and add entries to the Synapse database
- Read and write gizmo parameters at runtime, including user gizmos
- Issue triggers and retrieve system status

Synapse API is a RESTful interface between a built-in TCP server and user development tools. Client applications can connect to Synapse through direct HTTP requests.



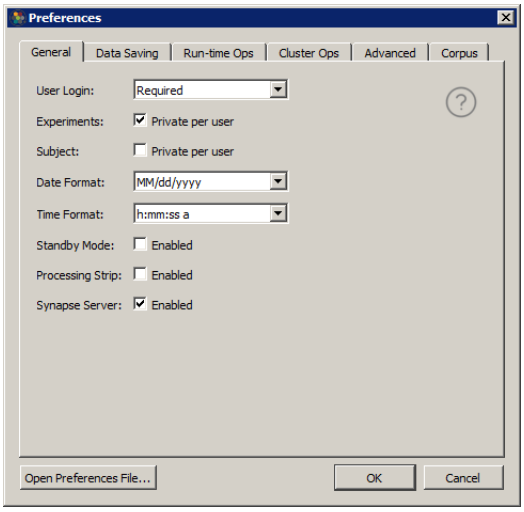
Synapse and Synapse API Functional Diagram

Synapse also uses TDT's TTank data server and data format which provides compatibility with many of the TDT OpenEx client applications, such as OpenExplorer and OpenScope, and includes direct data import into Matlab with TDT2mat.m.

Using Synapse API Tools in Synapse

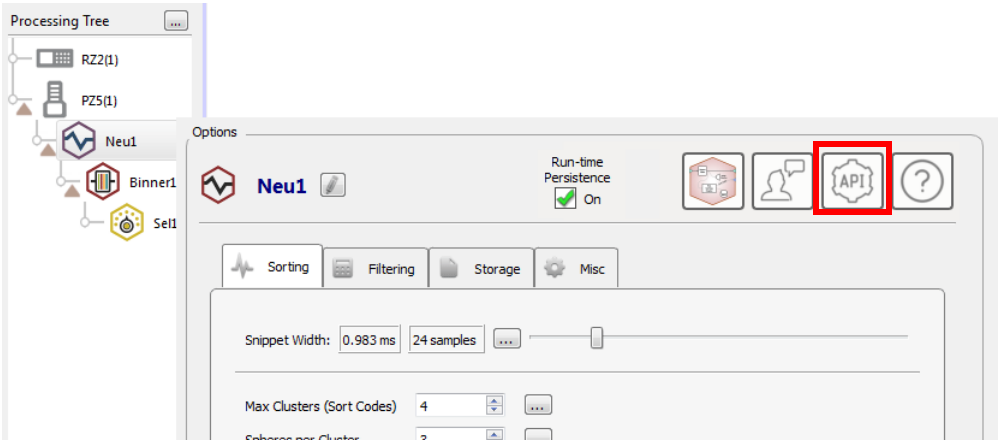
To enable Synapse API tools in the Synapse user interface, you must modify the Synapse preferences to start the built-in TCP Server.

1. In Synapse, click **Menu** and **Preferences**.



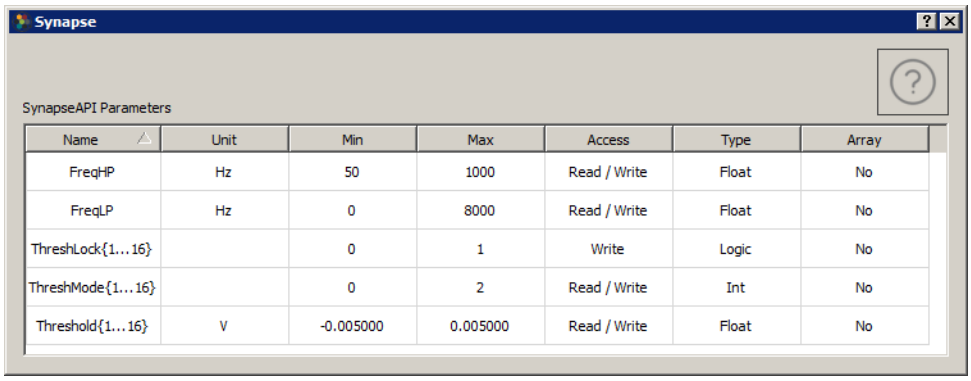
2. Click the **Synapse Server Enabled** check box.
3. Click **OK**.
4. Close and restart Synapse.

Select a gizmo or hardware item in the Processing Tree, and if that item supports the API you will see an API button, pictured below, on the upper right of the Options page for that item.



PCA Spike Sporting Gizmo Options with Synapse API Tools Enabled

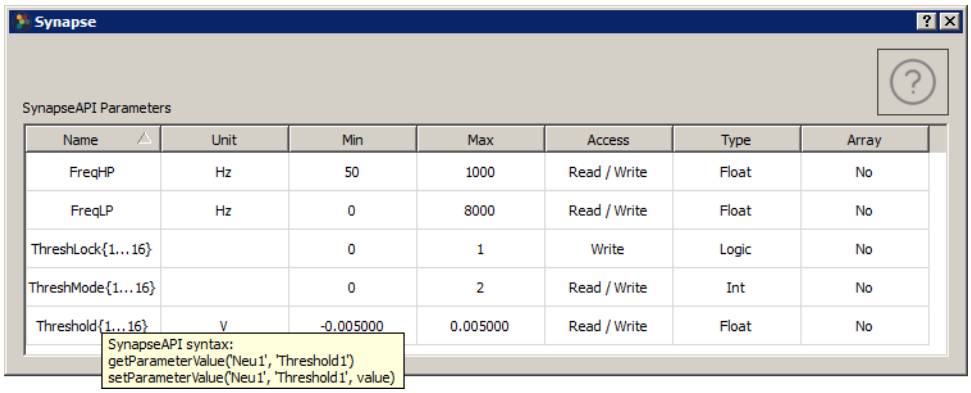
Clicking the API button displays the parameters available for the selected gizmo or hardware object.



PCA Spike Sorting API Dialog

Each row describes the specified parameter with bounded values; including Min and Max values, the type of access allowed through the API, and the data type. The size of the parameter is also displayed if it contains more than one value. For example, you will see the size displayed for the Channel Mapper gizmo API.

The API dialog table is for information purposes only and does not allow you to modify the parameters. You can hover over the Name cell to view the syntax you would use in Python or Matlab to get and set (if allowed) the value of the parameter.

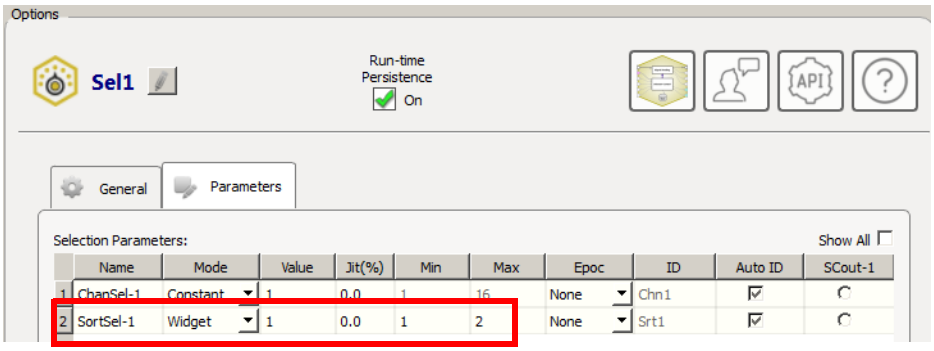


PCA Spike Sorting API Dialog with the Syntax Tooltip Displayed

Parameters that have one entry per channel will condense into a single row for easier viewing. For example, the Threshold parameter of PCA Spike Sorting shown in the table above is actually 16 unique parameters, one for each channel in this example, called 'Threshold1', 'Threshold2', and so on up to 'Threshold16'. You can't get/set all of the Thresholds at once, but instead you make separate calls to get/set each one individually.

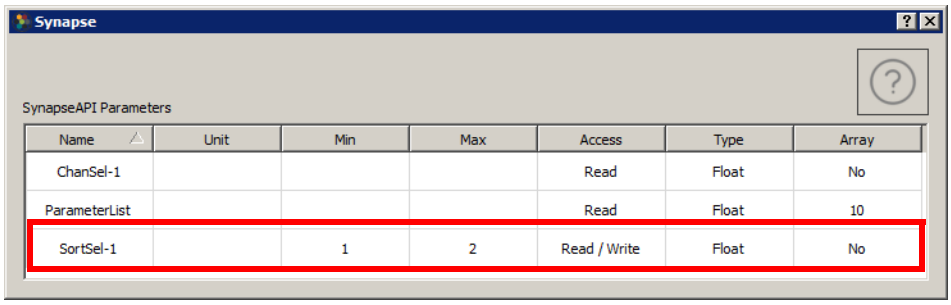
The parameters available for Synapse API access may be controlled during runtime only. For example, the table above displays the parameters available for the PCA Spike Sorting: high-pass and low-pass filter frequency and the spike detection threshold. When using Synapse API to set these values, you will see the widgets update in the Synapse runtime interface with the new values and these changes will also be logged into the Synapse database.

When accessing parameters defined in a bounded Parameters table, the desired parameter must be set to Widget Mode (runtime widget enabled), or else the parameter will not be available as a Synapse API parameter and cannot be modified at runtime. Below, the SortSel-1 parameter of the Selector gizmo is set to Widget Mode.



Selector Options Parameter Table with SortSel-1 Set to Widget Mode

In this mode it then becomes a valid Synapse API parameter, and the API dialog shows the same Min/Max values defined in the Parameter Table.



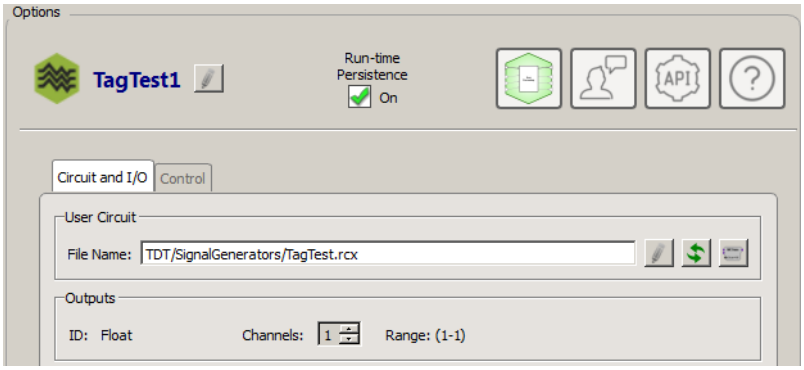
Selector Gizmo API Dialog After SortSel-1 Set to Widget Mode

Using Synapse API Tools With User Gizmos

Any parameter in a User Gizmo that is defined by a matching parameter tag and gizmoControl macro pair is also accessible with Synapse API and will appear in the API dialog.

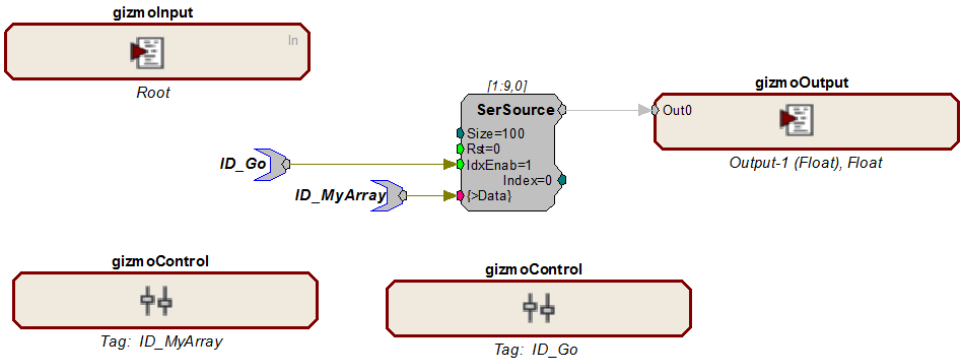
Note: See the User Gizmos section of the Synapse Manual for information on building and using User gizmos.

The example below shows the TagTest gizmo, which is available in the TDT > SignalGenerators category of Custom gizmos. This gizmo outputs an array of numbers controlled by the parameter called 'MyArray' and turns the output on and off using a parameter called 'Go'. Add the TagTest gizmo to your experiment, followed by a Stream Data Storage gizmo, to follow along with the example below.



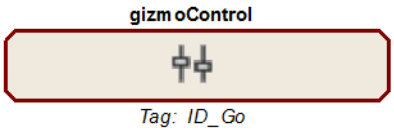
TagTest1 User Gizmo Setup

The RPvdsEx circuit used to implement the gizmo (TagTest.rcx) contains four gizmo builder macros: gizmoInput, gizmoOutput, and two gizmoControls. The gizmoControl macros create the parameter specifications for this custom gizmo.



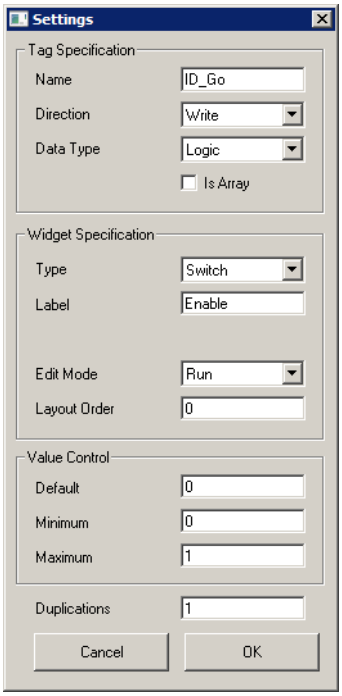
Example User Gizmo RPvdsEx Circuit with Logic and Array Controls

The first gizmoControl macro (ID_Go) turns the array test output on and off. By attaching the TagTest gizmo to a Stream Data Storage gizmo you can verify this at runtime.



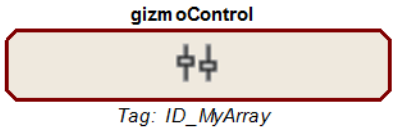
gizmoControl Macro

In the macro settings for ID_Go, the parameter tag is defined as a logic value, and the runtime interface is a switch widget.



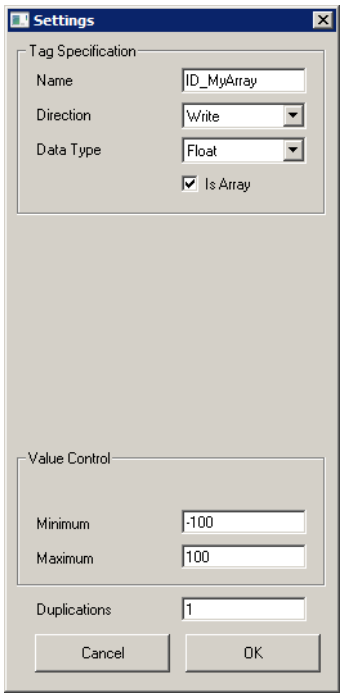
gizmoControl Macro Settings for 'Go' parameter

The circuit also includes a gizmoControl to define an array of values that can be written to dynamically and are sent as output when the test is running.



gizmoControl Macro

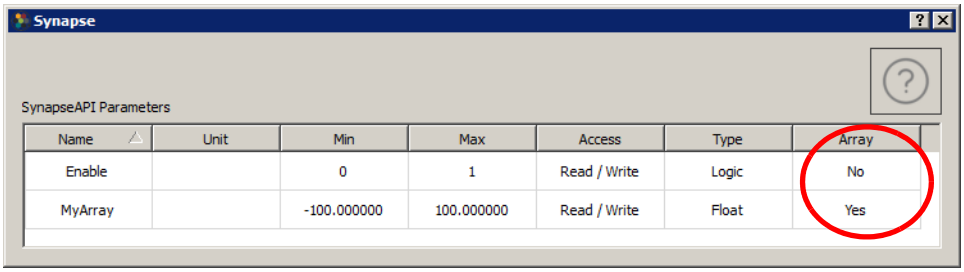
In the macro settings, the parameter tag is defined and the 'Is Array' checkbox is selected.



gizmoControl Macro Settings for 'MyArray' parameter

The bounds (Minimum and Maximum) for the parameter are applied to each element in the array whenever the API is writing the array values.

Because this user gizmo includes gizmoControl macros, the API button  is shown in the Options area and the Synapse API Dialog can be displayed.



TagTest Synapse API Parameters

Notice that the 'MyArray' parameter is recognized as an Array. For User gizmos, the size of any parameter array can only be determined programmatically at runtime using getParameterSize, or by manually examining the circuit.

Here is example Matlab code for interacting with the user gizmo parameters.

```
% create Synapse API connection
syn = SynapseAPI('localhost');

% switch into a runtime mode (Preview in this case)
if syn.getMode() < 1, syn.setMode(2), end

% get all info on the 'MyArray' parameter
GIZMO = 'TagTest1';
PARAMETER = 'MyArray';
info = syn.getParameterInfo(GIZMO, PARAMETER)

% get the array size (should be 100)
sz = syn.getParameterSize(GIZMO, PARAMETER)

% write values 1 to 50 in first half of buffer
result = syn.setParameterValues(GIZMO, PARAMETER, 1:50, 50)

% read all values from buffer
syn.getParameterValues(GIZMO, PARAMETER, sz)

% get all info on the 'Go' parameter
PARAMETER = 'Go';
info = syn.getParameterInfo(GIZMO, PARAMETER)

% flip the switch
result = syn.setParameterValue(GIZMO, PARAMETER, 1)

% check the value
value = syn.getParameterValue(GIZMO, PARAMETER);
fprintf('value = %d\n', value);

% also verify visually that the switch slipped in the run
% time interface. This state change will be logged just
% like any other variable change and saved with the runtime
% state.
```

Using Synapse API to Record Notes

The Run-time Notes feature built into the RZ and RX objects in Synapse lets you record timestamps and notes during a recording that are saved to disk.

You can assign common notes to buttons that are easily clicked by the user. There is also a text field to enter custom notes. The SynapseAPI gives you a way to record notes through your custom application.

Unlike other gizmos that use the API, the RZ and RX objects do not have the API button so their API parameters are not easily visible in the Synapse window.

The 'Run-time Notes' mode must be set to 'Notes File' or 'Notes File + Epocs'. The API communicates with the 'RecordingNotes' tab at run-time during a Recording only; note capture does not work during Preview mode. Here is some example code:

```
GIZMO = 'RecordingNotes';
BUTTON = 'Button';
NOTE = 'Note';

% log note associated with first button
syn.setParameterValue(GIZMO, BUTTON, 0)

% log note associated with second button
syn.setParameterValue(GIZMO, BUTTON, 1)

% write a custom note
syn.setParameterValue(GIZMO, NOTE, 'My Custom Note')
```

Synapse API Methods

Mode Control and System Status

SynapseAPI

Description: Creates an instance of Synapse API and connects to the Synapse server through port 24414. By default it connects to Synapse running on your local machine (localhost), but can optionally connect to Synapse running on a remote machine.

Prototype: `syn = SynapseAPI()`

Sample Code: Open a connection to the Synapse server.

MATLAB

```
% connect to locally running Synapse
syn = SynapseAPI();
% this is equivalent to SynapseAPI('localhost')
% or you can connect to a remote server on your network
syn = SynapseAPI('10.1.0.55');
% show the current system mode
syn.getModeStr()
```

Python

```
# connect to locally running Synapse
syn = SynapseAPI()
# this is equivalent to SynapseAPI('localhost')
# or you can connect to a remote server on your network
syn = SynapseAPI('10.1.0.55')
```

getMode

Description: Returns the current system mode of Synapse as an integer. This call can be used in conjunction with setMode to control the operational mode of your entire system. The Synapse modes — including Idle, Standby, Preview, and Record — are described in the Synapse Manual.

Prototype: `iMode = getMode()`

Returns: `0 (Idle), 1 (Standby), 2 (Preview), 3 (Record)`

Sample Code: Open a connection to the Synapse server, and if the mode is Record (3) the routine runs.

MATLAB

```
syn = SynapseAPI();
if syn.getMode() == 3
   %Start Routine
end
```

Python

```
syn = SynapseAPI()
if syn.getMode() == 3:
   #Start Routine
```

getModeStr

Description: Returns the current system mode of Synapse as a string. See description of getMode.

Prototype: `sMode = getModeStr()`

Returns: 'Idle', 'Standby', 'Preview', 'Record'

Sample Code: Open a connection to the Synapse server, and if the mode is Record (3) the routine runs.

MATLAB

```
syn = SynapseAPI();
if syn.getModeStr() == 'Record'
   %Start Routine
end
```

Python

```
syn = SynapseAPI();
if syn.getModeStr() == 'Record':
   #Start Routine
```

setMode

Description: Sets the system mode of Synapse. The possible modes include: Idle, Standby, Preview, and Record.

Prototype: `bSuccess = setMode(iNewMode)`

Arguments: 0 (Idle), 1 (Standby), 2 (Preview), 3 (Record)

Returns: 0 (fails), 1 (succeeds)

Sample Code: This code sample opens a connection to the Synapse server. If the Synapse mode is not Record mode (3), setMode places Synapse in Record mode.

MATLAB

```
syn = SynapseAPI();
if syn.getMode() ~= 3
```

```
   syn.setMode(3)
end
```

Python

```
syn = SynapseAPI();
if syn.getMode() != 3:
   syn.setMode(3)
```

setModeStr

Description:	Sets the system mode of Synapse. The possible modes include: Idle, Standby, Preview, and Record.
Prototype:	`bSuccess = setMode(sNewMode)`
Arguments:	'Idle', 'Standby', 'Preview', 'Record'
Returns:	0 (fails), 1 (succeeds)
Sample Code:	This code sample opens a connection to the Synapse server. If the Synapse mode is not 'Record' mode, setModeStr places the Synapse in 'Record' mode.

MATLAB

```
syn = SynapseAPI();
if strcmp(syn.getModeStr(),'Record') ~= 1
   syn.setModeStr('Record')
end
```

Python

```
syn = SynapseAPI();
if syn.getModeStr() != 'Record':
   syn.setModeStr('Record')
```

getSystemStatus

Description:	Returns a structure containing system state information. This same information is found in the lower left corner of the Synapse main window.
Prototype:	`tStatus = getSystemStatus()`
Returns:	errorCount number of errors recordSecs number of seconds recorded rateMBps data rate as Mb per second sysLoad IO load percentage uiLoad UI load percentage
Sample Code:	Start a recording, wait 5 seconds, then retrieve system status information.

MATLAB

```
syn.setMode(3)
pause(5)
```

```
tStatus = syn.getSystemStatus()
```

Python

```
syn.setMode(3)
time.sleep(5)
tStatus = syn.getSystemStatus()
```

getSamplingRates

Description: Returns a structure containing the sampling rates for each device in the Processing Tree.

Prototype: `tSamplingRates = getSamplingRates()`

Sample Code: Retrieve the device sampling rate for an RZ6 processor.

MATLAB

```
result = syn.getSamplingRates();
sf = result.RZ6_1
```

Python

```
result = syn.getSamplingRates()
sf = result['RZ6(1)']
```

Gizmos and Parameters

getGizmoNames

Description: Returns a cell array of all gizmos in the running experiment. This can be used with getParameterNames and get and set parameter methods to change parameters in runtime.

Prototype: `cGizmos = getGizmoNames()`

Sample Code: Retrieve the gizmo names in current experiment.

MATLAB

```
gizmo_names = syn.getGizmoNames()
if numel(gizmo_names) < 1
   error('no gizmos found')
end
```

Python

```
gizmo_names = syn.getGizmoNames()
if len(gizmo_names) < 1:
   error('no gizmos found')
```

getGizmoParent

Description: Return the name of the parent processor for a particular gizmo. Useful if you have multiple processors and want to know the sampling rate a particular gizmo is running at.

Prototype: `sGizmoParent = getGizmoParent(sGizmoName)`

Sample Code: Get the gizmo's sampling rate by querying the parent processor.

MATLAB

```
parent = syn.getGizmoParent('aStim1');
rates = syn.getSamplingRates();
gizmoRate = rates.(parent);
```

Python

```
parent = syn.getGizmoParent('aStim1')
rates = syn.getSamplingRates()
gizmoRate = rates[parent]
```

getGizmoInfo

Description: Returns a cell array of containing the gizmo type, category, description, and icon (string of base64-encoded text representing the icon).

Prototype: `cGizmoInfo = getGizmoInfo(sGizmoName)`

Sample Code: Retrieve the gizmo info in current experiment.

MATLAB

```
gizmo_info = syn.getGizmoInfo('aStim1')
```

Python

```
gizmo_info = syn.getGizmoInfo('aStim1')
```

getParameterNames

Description: Returns a cell array of parameters for the specified gizmo. This can be used with getGizmoNames and get/set parameter methods to change parameters at runtime.

Prototype: `sParameters = getParameterNames(sGizmo)`

Sample Code: Retrieve all parameter names of all gizmos

MATLAB

```
gizmo_names = syn.getGizmoNames()
for i = 1:numel(gizmo_names)
   gizmo = gizmo_names{i}
   params = syn.getParameterNames(gizmo);
end
```

Python

```
gizmo_names = syn.getGizmoNames()
for gizmo in gizmo_names:
   params = syn.getParameterNames(gizmo)
```

getParameterInfo

Description: Returns a structure containing parameter information.

Note: the same information is displayed in a table in the Synapse designtime interface when you click the API button on the gizmo options tab.

Prototype: `tParameterInfo = getParameterInfo(sGizmo, sParameter)`

Returns:

Name	name of the parameter
Unit	the units label for this parameter
Min	the minimum allowed value for this parameter
Max	the maximum allowed value for this parameter
Type	the parameters data type. 'Float', 'Int', 'Logic'
Array	During designtime: If the parameter is an array of known size, this field contains the number of elements in this parameter. If the size is unknown (for example, User gizmo tag arrays or Legacy Hal parameter tags), this field contains 'Yes'. Otherwise it contains 'No' for any scalar parameters. During runtime: this field will always contain the size of the parameter if it is an array (>1), or 'No' if it is not.

Sample Code: Retrieve all parameter info of all gizmos.

MATLAB

```
gizmo_names = syn.getGizmoNames()
for i = 1:numel(gizmo_names)
   gizmo = gizmo_names{i}
   params = syn.getParameterNames(gizmo)
   for i = 1:numel(params)
      param = params{i};
      info = syn.getParameterInfo(gizmo, param);
   end
end
```

Python

```
gizmo_names = syn.getGizmoNames()
for gizmo in gizmo_names:
   params = syn.getParameterNames(gizmo)
   for param in params:
      info = syn.getParameterInfo(gizmo, param)
```

getParameterSize

Description: Returns the size of the specified parameter from the specified gizmo. This can be used with getGizmoNames and getParameterNames.

Prototype: `dValue = getParameterSize(sGizmo, sParameter)`

Sample Code: Retrieve all parameter sizes of all gizmos

MATLAB

```
gizmo_names = syn.getGizmoNames()
for i = 1:numel(gizmo_names)
   gizmo = gizmo_names{i}
   params = syn.getParameterNames(gizmo)
   for i = 1:numel(params)
      param = params{i};
      size = syn.getParameterSize(gizmo);
   end
end
```

Python

```
gizmo_names = syn.getGizmoNames()
for gizmo in gizmo_names:
   params = syn.getParameterNames(gizmo)
   for param in params:
      size = syn.getParameterSize(gizmo, param)
```

getParameterValue

Description: Returns the value of the specified parameter from the specified gizmo. This can be used with getGizmoNames and getParameterNames.

Prototype: `dValue = getParameterValue(sGizmo, sParameter)`

Sample Code: Retrieve a General Purpose Filter gizmo high pass frequency.

MATLAB

```
syn.setMode(2)
val = syn.getParameterValue('Filt1', 'HighPassFreq')
```

Python

```
syn.setMode(2)
val = syn.getParameterValue('Filt1', 'HighPassFreq')
```

setParameterValue

Description: Sets the value of the specified parameter. This can be used with getGizmoNames and getParameterNames. When setting

a parameter value, the value is bounded by the min/max values of the parameter. See getParameterInfo.

Prototype: `bSuccess = setParameterValue(sGizmo, sParameter, dValue)`

Returns: 0 (fails), 1 (succeeds)

Sample Code: Retrieve a General Purpose Filter high pass frequency, then increments it by 1.

MATLAB

```
val = syn.getParameterValue('Filt1', 'HighPassFreq')
syn.setParameterValue('Filt1', 'HighPassFreq', val + 1)
val = syn.getParameterValue('Filt1', 'HighPassFreq')
```

Python

```
val = syn.getParameterValue('Filt1', 'HighPassFreq')
syn.setParameterValue('Filt1', 'HighPassFreq', val + 1)
val = syn.getParameterValue('Filt1', 'HighPassFreq')
```

getParameterValues

Description: Returns the values of the specified parameter array.

Prototype: `fValues = getParameterValues(sGizmo, sParameter, count=-1, offset=0)`

Sample Code: Retrieve the map array from a Channel Mapper gizmo.

MATLAB

```
currMap = syn.getParameterValues('Map1', 'ChanMap')
```

Python

```
currMap = syn.getParameterValues('Map1', 'ChanMap')
```

setParameterValues

Description: Sets the values of the specified parameter array. When setting a parameter array, all values are bounded by the min/max values of the parameter. See getParameterInfo.

Prototype: `bSuccess = setParameterValues(sGizmo, sParameter, values, offset=0)`

Returns: 0 (fails), 1 (succeeds)

Sample Code: Set the map array of a Channel Mapper gizmo.

MATLAB

```
currMap = syn.getParameterValues('Map1', 'ChanMap')
defaultMap = 1:numel(currMap);
syn.setParameterValues('Map1', 'ChanMap', defaultMap);
```

Python:

```
currMap = syn.getParameterValues('Map1', 'ChanMap')
defaultMap = 1:len(currMap)
```

```
syn.setParameterValues('Map1', 'ChanMap', defaultMap)
```

Lab Management

getKnownExperiments

getKnownSubjects

getKnownUsers

getKnownTanks

getKnownBlocks

Description: Returns a cell array of known experiments, subjects, users, tanks, blocks.

Prototype:

```
cExperiments = getKnownExperiments()
cSubjects = getKnownSubjects()
cUsers = getKnownUsers()
cTanks = getKnownTanks()
cBlocks = getKnownBlocks()
```

Sample Code: Retrieve existing experiments, subjects, and users from the Synapse database.

MATLAB

```
result = syn.getKnownExperiments()
if numel(result) < 1
   error('no experiments found')
end
result = syn.getKnownSubjects()
if numel(result) < 1
   error('no subjects found')
end
result = syn.getKnownUsers()
if numel(result) < 1
   error('no users found')
end
```

Python

```
result = syn.getKnownExperiments()
if len(result) < 1:
   error('no experiments found')
result = syn.getKnownSubjects()
if len(result) < 1:
```

```
   error('no subjects found')
result = syn.getKnownUsers()
if len(result) < 1:
   error('no users found')
```

getCurrentExperiment

getCurrentSubject

getCurrentUser

getCurrentTank

getCurrentBlock

Description: Returns the name of the current experiment, subject, user, tank path, block name.

Prototype:
```
sExperiment = getCurrentExperiment()
sSubject = getCurrentSubject()
sUser = getCurrentUser()
sTank = getCurrentTank()
sBlock = getCurrentBlock()
```

Sample Code: Retrieve the current experiment, user, tank, and block name.

MATLAB
```
currUser = syn.getCurrentUser()
currExperiment = syn.getCurrentExperiment()
currSubject = syn.getCurrentSubject()
currTank = syn.getCurrentTank()
currBlock = syn.getCurrentBlock()
```

Python
```
currUser = syn.getCurrentUser()
currExperiment = syn.getCurrentExperiment()
currSubject = syn.getCurrentSubject()
currTank = syn.getCurrentTank()
currBlock = syn.getCurrentBlock()
```

createSubject

Description: Creates a subject with the given name, description, and icon. Allowed icons are ‘mouse’, ‘rat’, ‘monkey’, ‘marmoset’, ‘human’, ‘bat’, ‘owl’, ‘bird’, ‘ferret’, ‘gerbil’, ‘guinea-pig’, ‘rabbit’, ‘pig’, ‘cat’, ‘dog’, ‘fish’, ‘dolphin’, ‘snake’, ‘shark’, ‘duck’, ‘cow’, ‘goat’, ‘horse’.

Prototype:
```
bSuccess = createSubject(sName, sDesc, sIcon)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Create a new subject, and set it as the current subject before recording.

MATLAB

```
nextSubj = 'ABC123';
syn.createSubject(nextSub, 'Control', 'mouse')
syn.setCurrentSubject(nextSub)
syn.setMode(3)
```

Python

```
nextSub = 'ABC123';
syn.createSubject(nextSub, 'Control', 'mouse')
syn.setCurrentSubject(nextSub)
syn.setMode(3)
```

createTank

Description: Creates a tank at the given path.

Prototype: `bSuccess = createTank(sTankPath)`

Returns: 0 (fails), 1 (succeeds)

Sample Code: Create a new tank, set it as the current tank, and start recording.

MATLAB

```
nextTank = 'C:\TDT\NEXTTANK'
syn.createTank(nextTank)
syn.setCurrentTank(nextTank)
syn.setMode(3)
```

Python

```
nextTank = 'C:\TDT\NEXTTANK'
syn.createTank(nextTank)
syn.setCurrentTank(nextTank)
syn.setMode(3)
```

setCurrentExperiment

setCurrentSubject

setCurrentUser

Description: In designtime, change the currently selected experiment, subject, user.

Prototype:

```
bSuccess = setCurrentExperiment(sExperiment)
bSuccess = setCurrentSubject(sSubject)
```

```
bSuccess = setCurrentSubject(sSubject,
sPassword)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Find first known experiment, subject, and user, and select them.

MATLAB

```
result = syn.knownExperiments()
syn.setCurrentExperiment(result{1})
result = syn.knownSubjects()
syn.setCurrentSubject(result{1})
result = syn.knownUsers()
syn.setCurrentUser(result{1})
```

Python

```
result = syn.knownExperiments()
syn.setCurrentExperiment(result[0])
result = syn.knownSubjects()
syn.setCurrentSubject(result[0])
result = syn.knownUsers()
syn.setCurrentUser(result[0])
```

setCurrentTank

Description: Switches to the specified tank. Note that the 'Auto Tank Naming' option in the Preferences menu must be disabled for this to succeed.

Prototype: `bSuccess = setCurrentTank(sTank)`

Returns: 0 (fails), 1 (succeeds)

Sample Code: Set the tank name before the next recording.

MATLAB

```
syn.setMode(0)
nextTank = 'C:\TDT\SYNTANK';
syn.setCurrentTank(nextTank)
syn.setMode(3)
```

Python

```
syn.setMode(0)
nextTank = 'C:\TDT\SYNTANK';
syn.setCurrentTank(nextTank)
syn.setMode(3)
```

setCurrentBlock

Description: Switches to the specified block. Note that the 'Block Naming' option in the Preferences menu must be set to 'Prompt' for this to succeed.

Prototype:
```
bSuccess = setCurrentBlock(sBlock)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Set the block name before the next recording.

MATLAB

```
syn.setMode(0)
nextBlock = 'MyBlockName';
syn.setCurrentBlock(nextBlock)
syn.setMode(3)
syn.currentBlock()
```

Python

```
syn.setMode(0)
nextBlock = 'MyBlockName';
syn.setCurrentBlock(nextBlock)
syn.setMode(3)
```

getSubjectMemos

getExperimentMemos

getUserMemos

Description: Retrieves the log entries from the Synapse database for the specified subject, experiment, or user. These can optionally be filtered by time using the STARTTIME and STOPTIME parameters which are formatted like %Y%m%d%H%M%S, e.g. January 22, 2017 9:53 am would be 20170122095300.

Prototype:
```
cNotes = getSubjectMemos(sSubject)
cNotes = getUserMemos(sUser)
cNotes = getExperimentMemo(sExperiment)
```

Returns: array of the filtered notes

Sample Code: Retrieve specific notes from the Synapse database.

MATLAB

```
currSubject = syn.currentSubject()
cNotes = syn.getSubjectMemos(currSubject)
cNotes = syn.getSubjectMemos(currSubject, 'STARTTIME',
   20170101000000, 'ENDTIME', 20171231235959)
currUser = syn.currentUser()
cNotes = syn.getUserMemos(currUser)
currExperiment = syn.currentExperiment()
```

```
cNotes = syn.getExperimentMemos(currExperiment)
```

Python

```
currSubject = syn.currentSubject()
notes = syn.getSubjectMemos(currSubject)
cNotes = syn.getSubjectMemos(currSubject,
   startTime='20170101000000', endTime='20171231235959')
currUser = syn.currentUser()
notes = syn.getUserMemos(currUser)
currExperiment = syn.currentExperiment()
notes = syn.getExperimentMemos(currExperiment)
```

appendSubjectMemo

appendExperimentMemo

appendUserMemo

Description: Adds a database entry linked to this specified subject, experiment, user, just as if the user typed it in the Logs dialog for each of these items.

Prototype:

```
bSuccess = appendSubjectMemo(sSubject, sMemo)
bSuccess = appendUserMemo(sUser, sMemo)
bSuccess = appendExperimentMemo(sExperiment, sMemo)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Add memos to current subject, user, and experiment.

MATLAB

```
currSubject = syn.currentSubject()
syn.appendSubjectMemo(currSubject, 'Sub memo from Matlab')
currUser = syn.currentUser()
syn.appendUserMemo(currUser, 'User memo from Matlab')
currExperiment = syn.currentExperiment()
syn.appendExperimentMemo(currExperiment, 'Exp memo1')
```

Python

```
currSubject = syn.currentSubject()
syn.appendSubjectMemo(currSubject, 'Sub memo from Python')
currUser = syn.currentUser()
syn.appendUserMemo(currUser, 'User memo from Python')
currExperiment = syn.currentExperiment()
syn.appendExperimentMemo(currExperiment, 'Exp memo1')
```

Persistence

getPersistModes

Description: Returns a cell array of the allowed persistence modes settable from the API. Currently this is 'Last', 'Best', and 'Fresh'.

Prototype:
```
cModes = getPersistModes()
```

getPersistMode

Description: Returns the current persistence mode ('Last', 'Best', 'Fresh', or 'User').

Prototype:
```
sMode = getPersistMode()
```

setPersistMode

Description: Sets the current persistence mode ('Last', 'Best', or 'Fresh'). Synapse must be in Idle mode.

Prototype:
```
bSuccess = setPersistMode(sMode)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Set persistence to 'Fresh' before entering run mode.

MATLAB
```
syn.setPersistMode('Fresh')
syn.setModeStr('Preview')
```

Python
```
syn.setPersistMode('Fresh')
syn.setModeStr('Preview')
```

Miscellaneous Utilities

issueTrigger

Description: Fires the named software trigger to all devices. Used only by User Gizmos that contain a TrgIn component.

Prototype:
```
bSuccess = issueTrigger(iTriggerId)
```

Returns: 0 (fails), 1 (succeeds)

Sample Code: Send software trigger 1 to all devices.

MATLAB
```
syn.issueTrigger(1)
```

Python
```
syn.issueTrigger(1)
```

getError

Description: Returns the previous error (if any) generated by a failure of one of the Synapse API functions.

Prototype:

```
sError = getError()
```