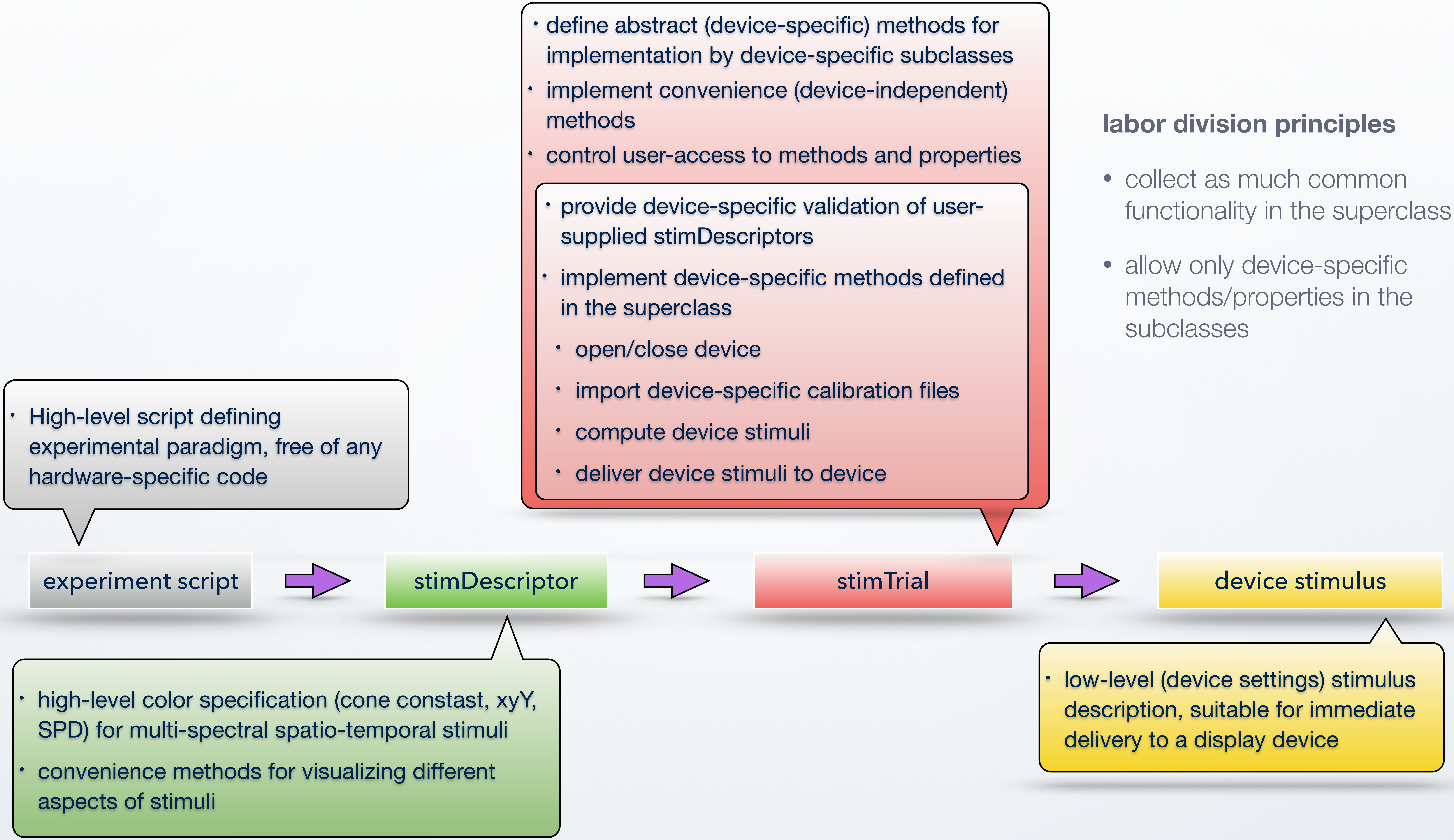


experiment script → stimDescriptor → stimTrial → device stimulus

- High-level script defining experimental paradigm, free of any hardware-specific code

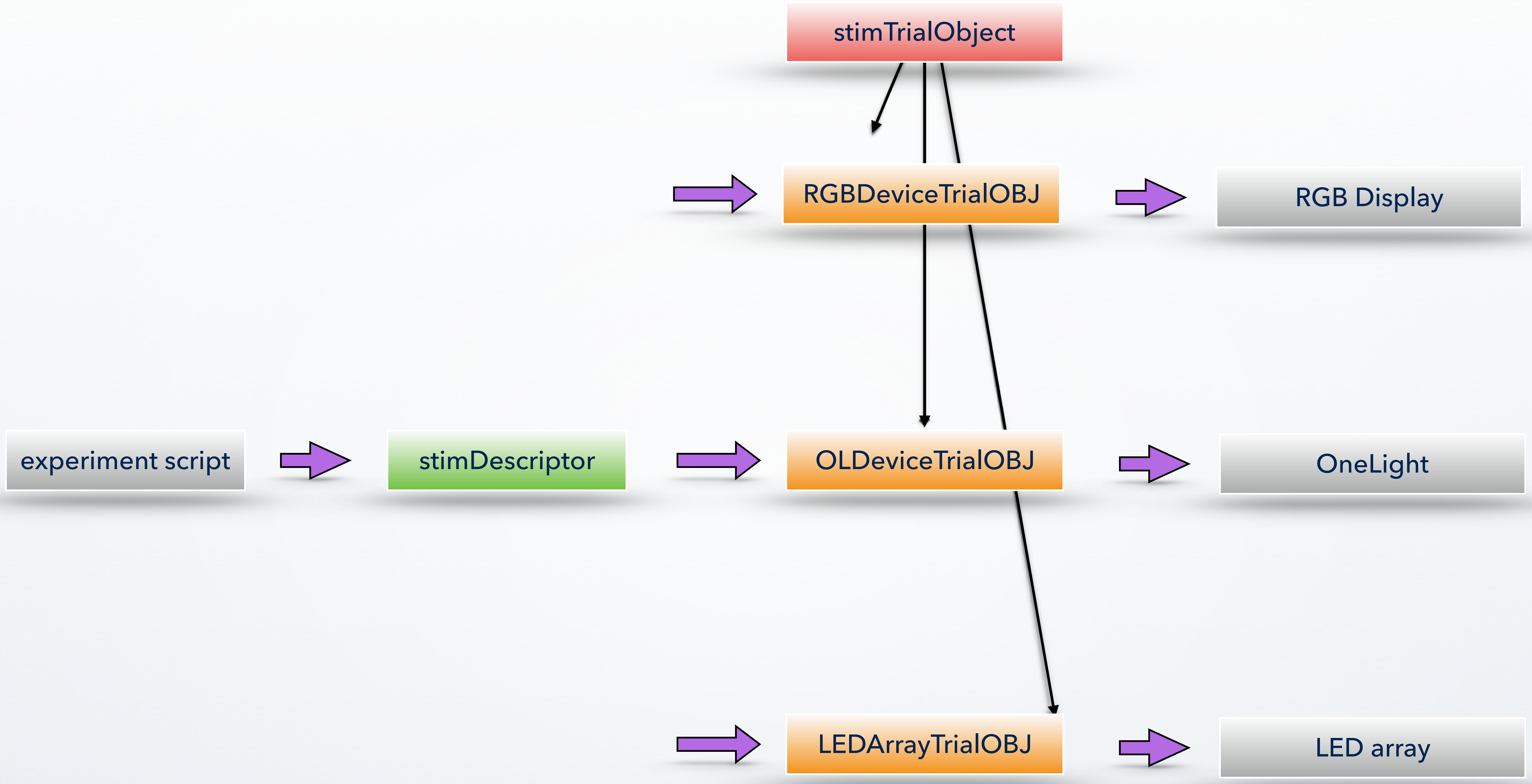
- high-level color specification (cone constast, xyY, SPD) for multi-spectral spatio-temporal stimuli
- convenience methods for visualizing different aspects of stimuli

- define abstract (device-specific) methods for implementation by device-specific subclasses
- implement convenience (device-independent) methods
- control user-access to methods and properties
  - provide device-specific validation of user-supplied stimDescriptors
  - implement device-specific methods defined in the superclass
    - open/close device
    - import device-specific calibration files
    - compute device stimuli
    - deliver device stimuli to device

- low-level (device settings) stimulus description, suitable for immediate delivery to a display device

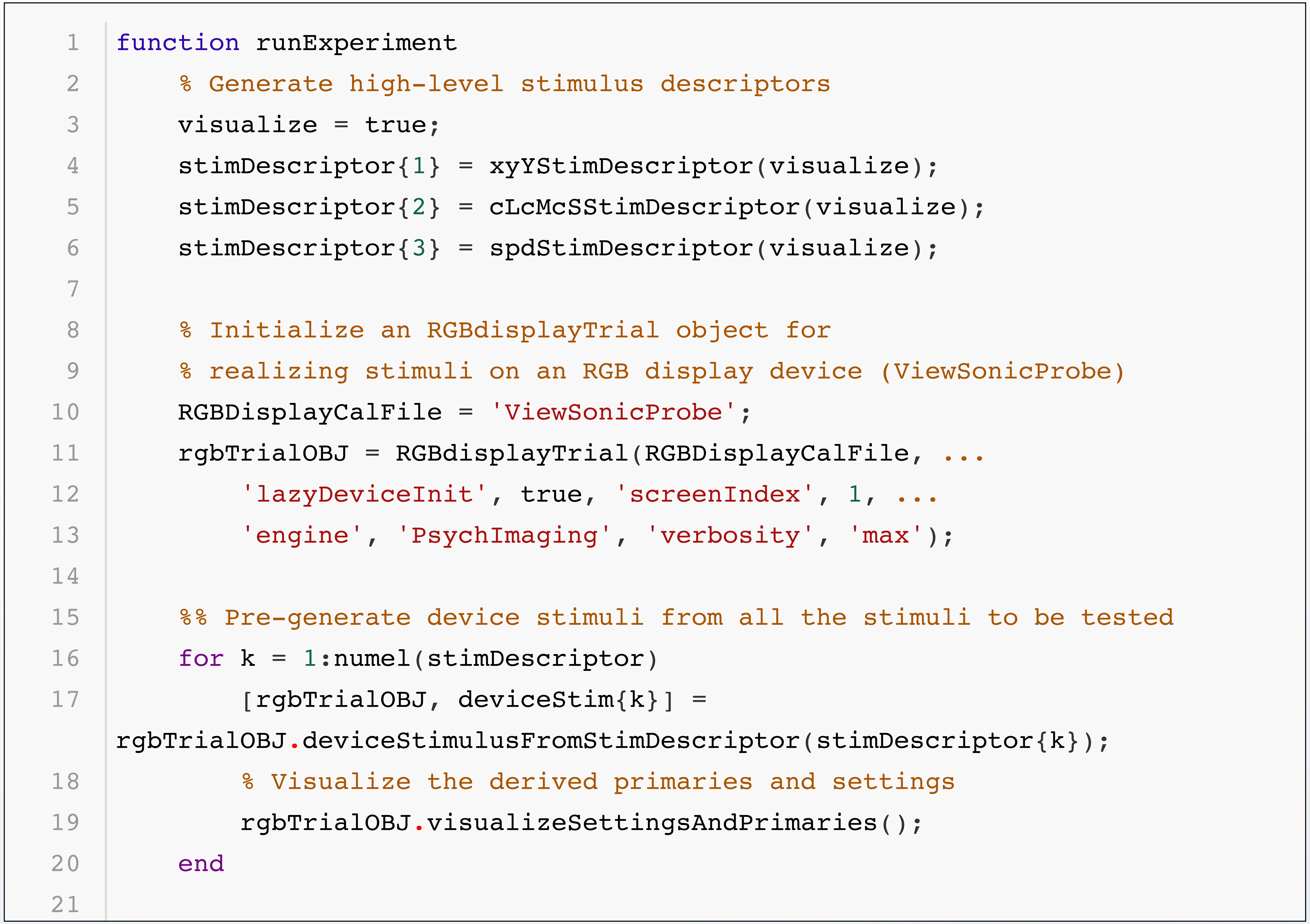
labor division principles

- collect as much common functionality in the superclass
- allow only device-specific methods/properties in the subclasses



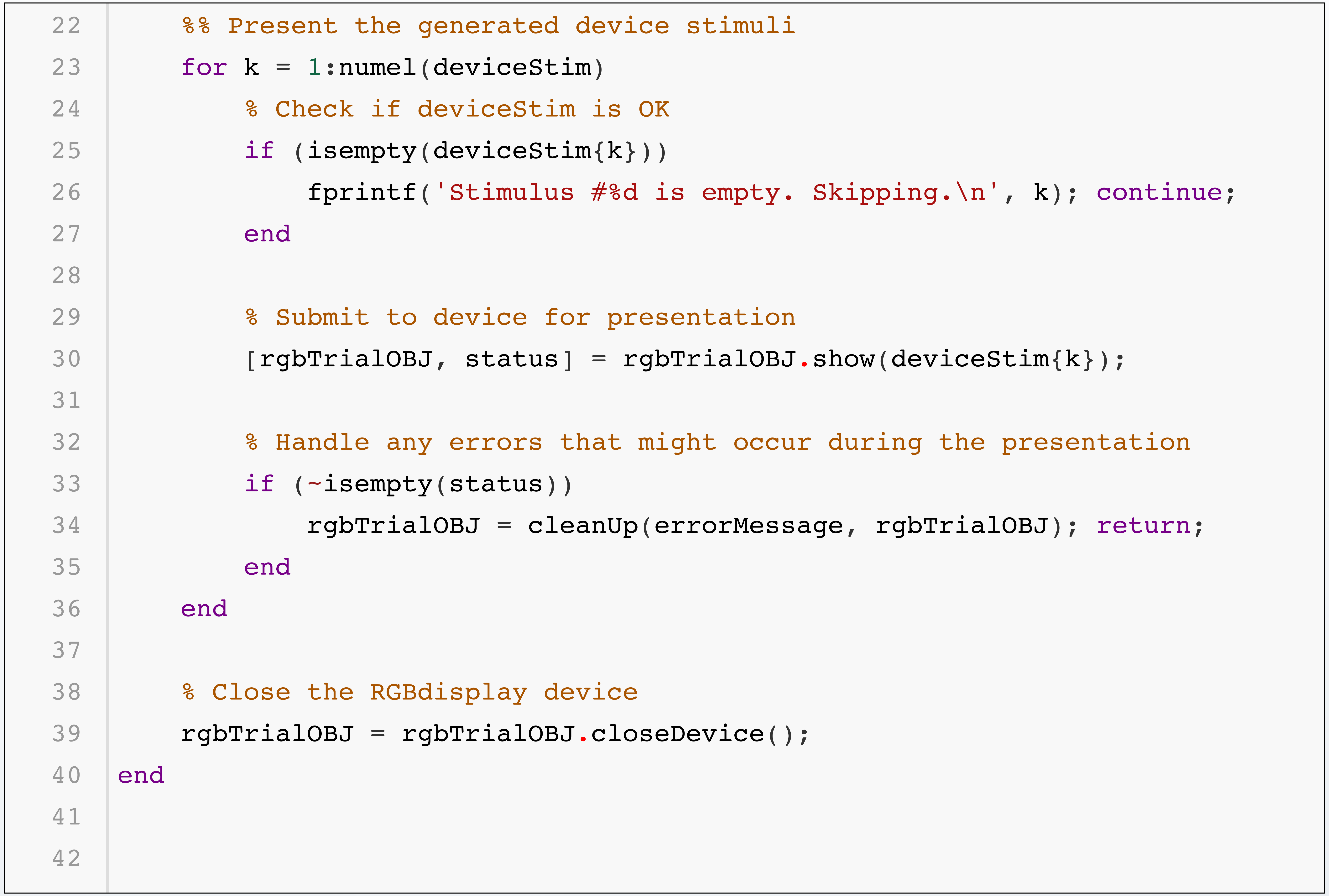
stimTrialObject
RGBDeviceTrialOBJ
RGB Display
experiment script
stimDescriptor
OLDeviceTrialOBJ
OneLight
LEDArrayTrialOBJ
LED array

experiment script



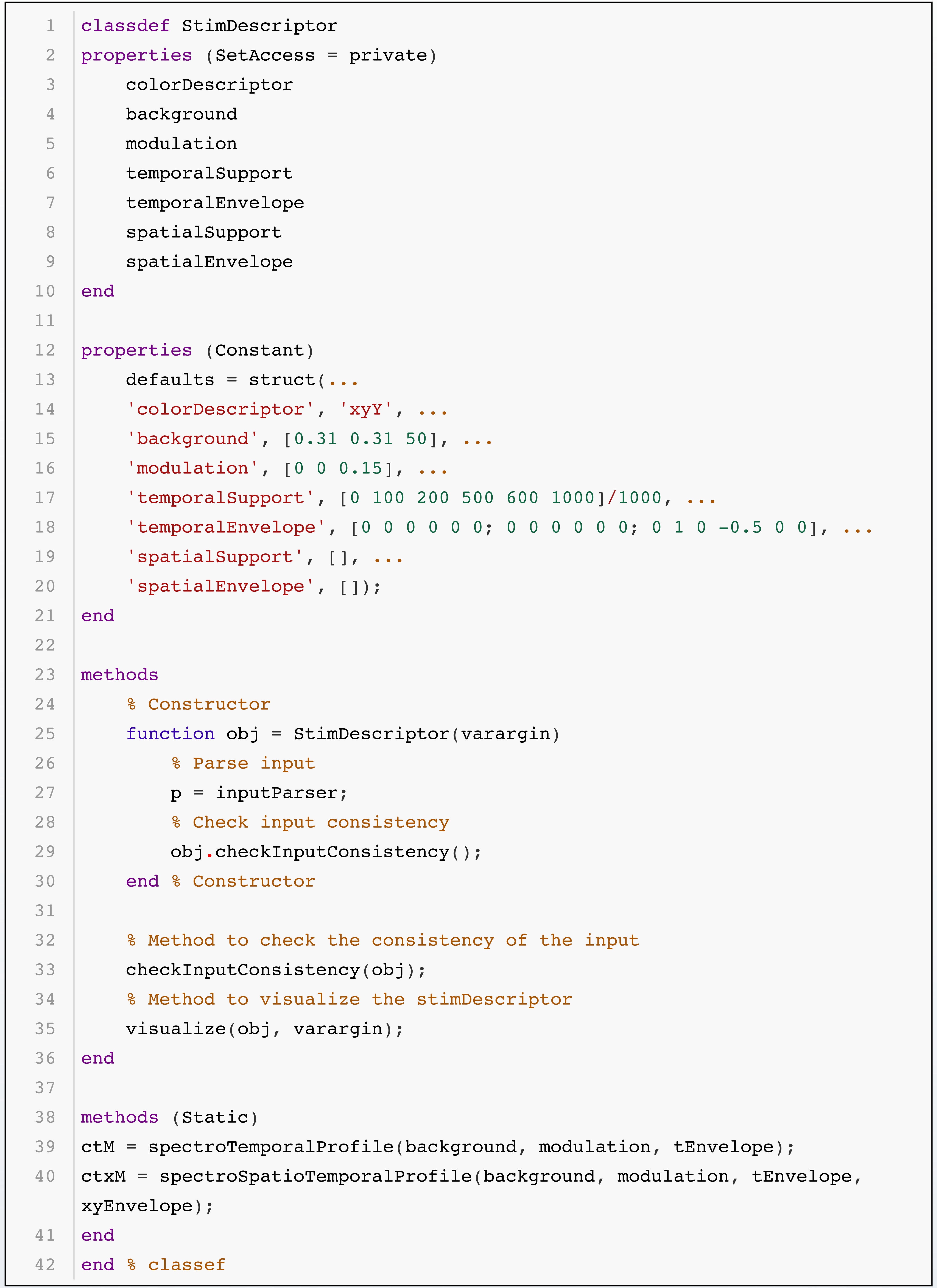
```
function runExperiment
    % Generate high-level stimulus descriptors
    visualize = true;
    stimDescriptor{1} = xyYStimDescriptor(visualize);
    stimDescriptor{2} = cLcMcSStimDescriptor(visualize);
    stimDescriptor{3} = spdStimDescriptor(visualize);

    % Initialize an RGBdisplayTrial object for
    % realizing stimuli on an RGB display device (ViewSonicProbe)
    RGBDisplayCalFile = 'ViewSonicProbe';
    rgbTrialOBJ = RGBdisplayTrial(RGBDisplayCalFile, ...
        'lazyDeviceInit', true, 'screenIndex', 1, ...
        'engine', 'PsychImaging', 'verbosity', 'max');

    %% Pre-generate device stimuli from all the stimuli to be tested
    for k = 1:numel(stimDescriptor)
        [rgbTrialOBJ, deviceStim{k}] = 
rgbTrialOBJ.deviceStimulusFromStimDescriptor(stimDescriptor{k});
        % Visualize the derived primaries and settings
        rgbTrialOBJ.visualizeSettingsAndPrimaries();
    end

```

experiment script



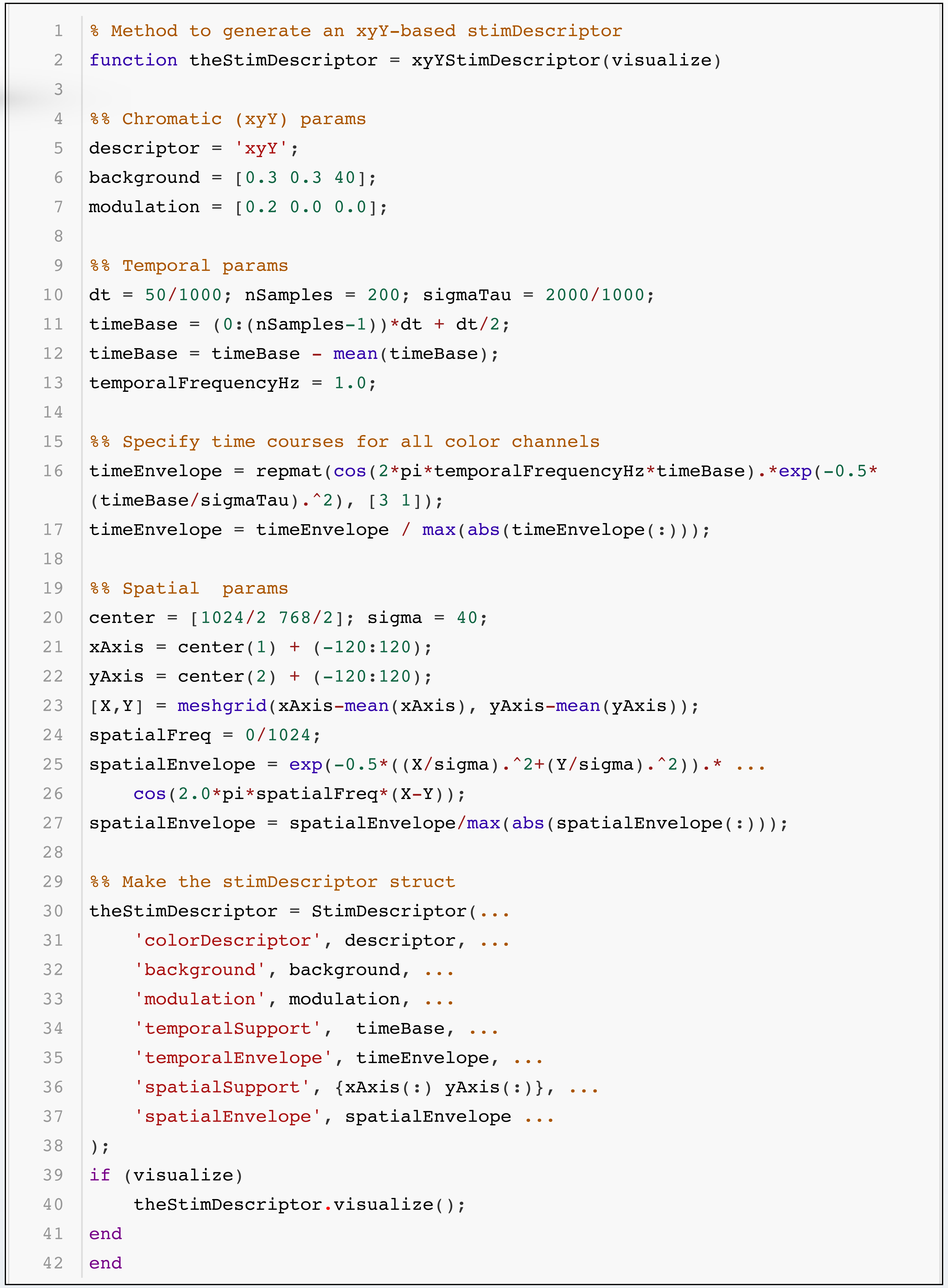
```
    %% Present the generated device stimuli
    for k = 1:numel(deviceStim)
        % Check if deviceStim is OK
        if (isempty(deviceStim{k}))
            fprintf('Stimulus #%d is empty. Skipping.\n', k); continue;
        end

        % Submit to device for presentation
        [rgbTrialOBJ, status] = rgbTrialOBJ.show(deviceStim{k});

        % Handle any errors that might occur during the presentation
        if (~isempty(status))
            rgbTrialOBJ = cleanUp(errorMessage, rgbTrialOBJ); return;
        end
    end

    % Close the RGBdisplay device
    rgbTrialOBJ = rgbTrialOBJ.closeDevice();
end
```

stimDescriptor

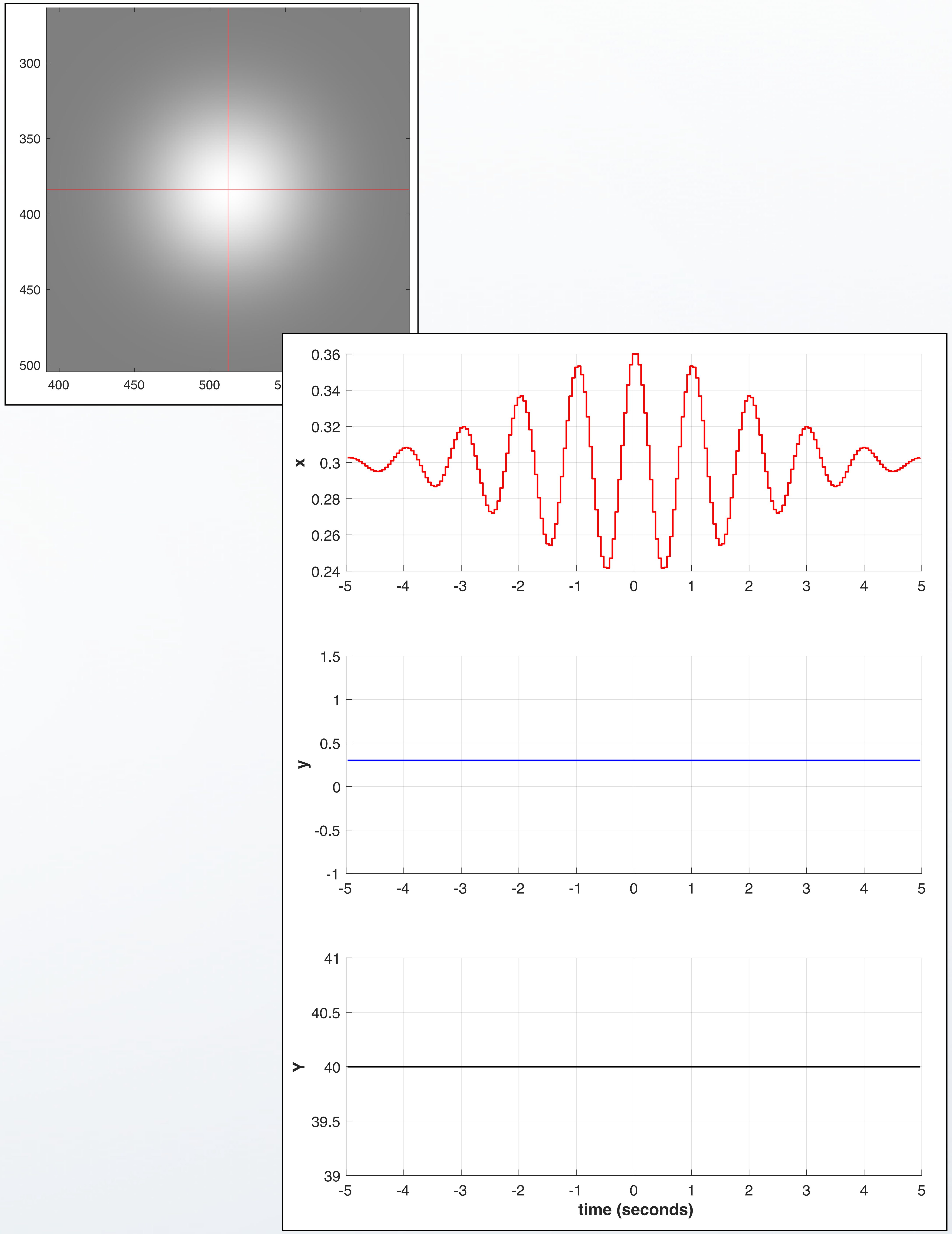


```
classdef StimDescriptor
properties (SetAccess = private)
    colorDescriptor
    background
    modulation
    temporalSupport
    temporalEnvelope
    spatialSupport
    spatialEnvelope
end

properties (Constant)
    defaults = struct(...
    'colorDescriptor', 'xyY', ...
    'background', [0.31 0.31 50], ...
    'modulation', [0 0 0.15], ...
    'temporalSupport', [0 100 200 500 600 1000]/1000, ...
    'temporalEnvelope', [0 0 0 0 0 0; 0 0 0 0 0 0; 0 1 0 -0.5 0 0], ...
    'spatialSupport', [], ...
    'spatialEnvelope', []);
end

methods
    % Constructor
    function obj = StimDescriptor(varargin)
        % Parse input
        p = inputParser;
        % Check input consistency
        obj.checkInputConsistency();
    end % Constructor

    % Method to check the consistency of the input
    checkInputConsistency(obj);
    % Method to visualize the stimDescriptor
    visualize(obj, varargin);
end

methods (Static)
ctM = spectroTemporalProfile(background, modulation, tEnvelope);
ctxM = spectroSpatioTemporalProfile(background, modulation, tEnvelope,
xyEnvelope);
end
end % classef
```

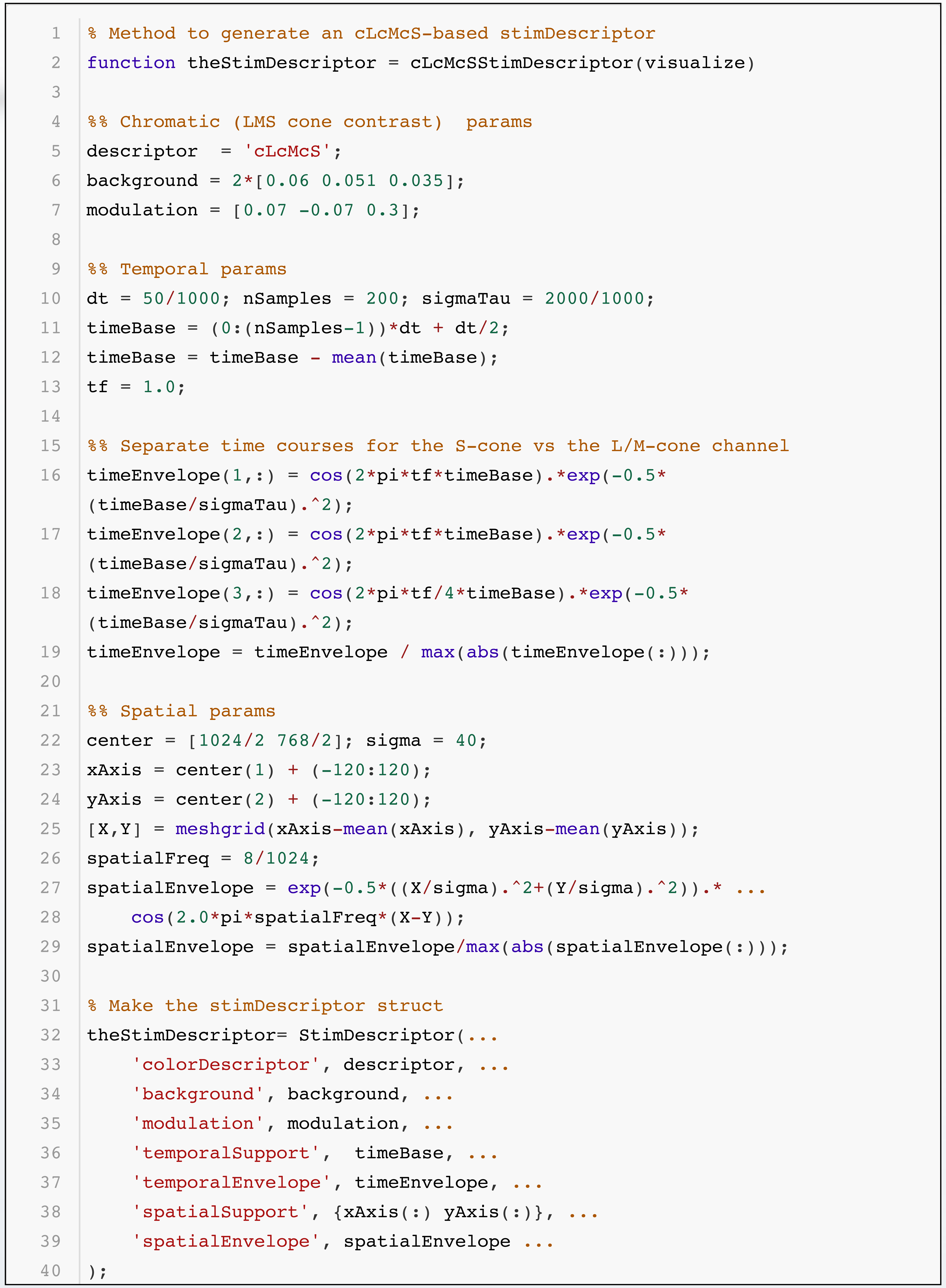
stimDescriptor



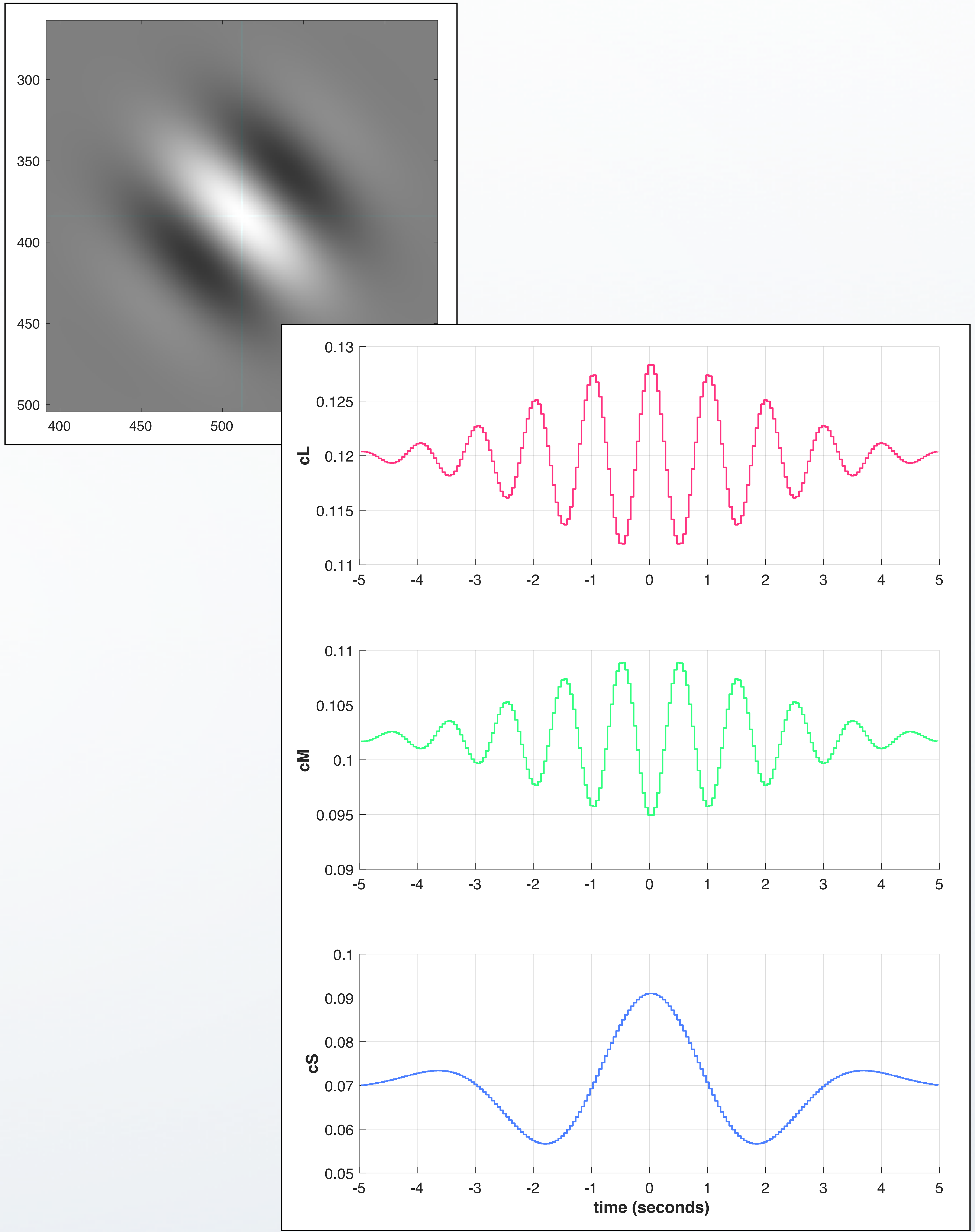
```
% Method to generate an xyY-based stimDescriptor
function theStimDescriptor = xyYStimDescriptor(visualize)

%% Chromatic (xyY) params
descriptor = 'xyY';
background = [0.3 0.3 40];
modulation = [0.2 0.0 0.0];

%% Temporal params
dt = 50/1000; nSamples = 200; sigmaTau = 2000/1000;
timeBase = (0:(nSamples-1))*dt + dt/2;
timeBase = timeBase - mean(timeBase);
temporalFrequencyHz = 1.0;

%% Specify time courses for all color channels
timeEnvelope = repmat(cos(2*pi*temporalFrequencyHz*timeBase).*exp(-0.5*
(timeBase/sigmaTau).^2), [3 1]);
timeEnvelope = timeEnvelope / max(abs(timeEnvelope(:)));

%% Spatial  params
center = [1024/2 768/2]; sigma = 40;
xAxis = center(1) + (-120:120);
yAxis = center(2) + (-120:120);
[X,Y] = meshgrid(xAxis-mean(xAxis), yAxis-mean(yAxis));
spatialFreq = 0/1024;
spatialEnvelope = exp(-0.5*((X/sigma).^2+(Y/sigma).^2)).* ...
    cos(2.0*pi*spatialFreq*(X-Y));
spatialEnvelope = spatialEnvelope/max(abs(spatialEnvelope(:)));

%% Make the stimDescriptor struct
theStimDescriptor = StimDescriptor(...
    'colorDescriptor', descriptor, ...
    'background', background, ...
    'modulation', modulation, ...
    'temporalSupport',  timeBase, ...
    'temporalEnvelope', timeEnvelope, ...
    'spatialSupport', {xAxis(:) yAxis(:)}, ...
    'spatialEnvelope', spatialEnvelope ...
);
if (visualize)
    theStimDescriptor.visualize();
end
end
```



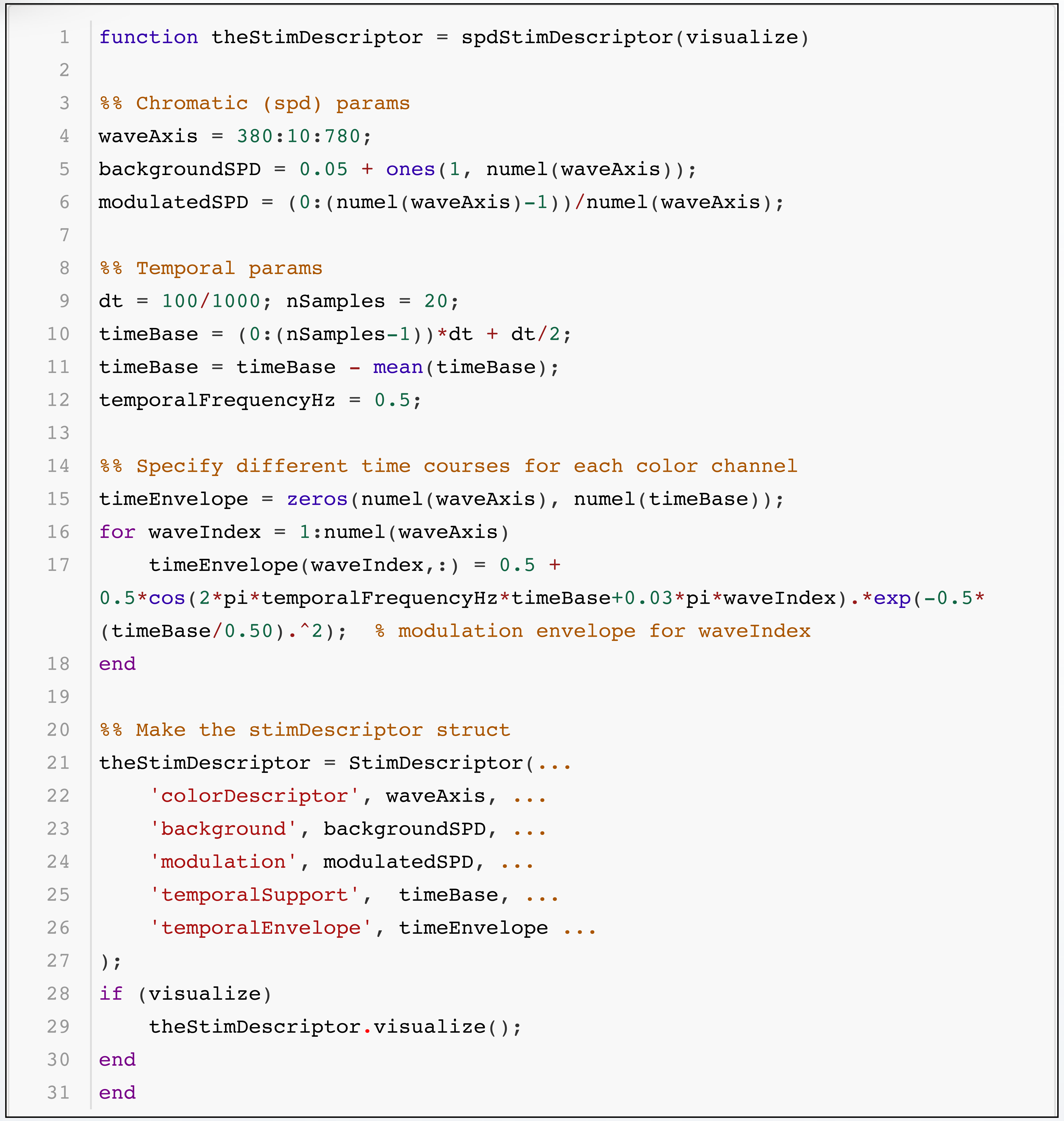
stimDescriptor



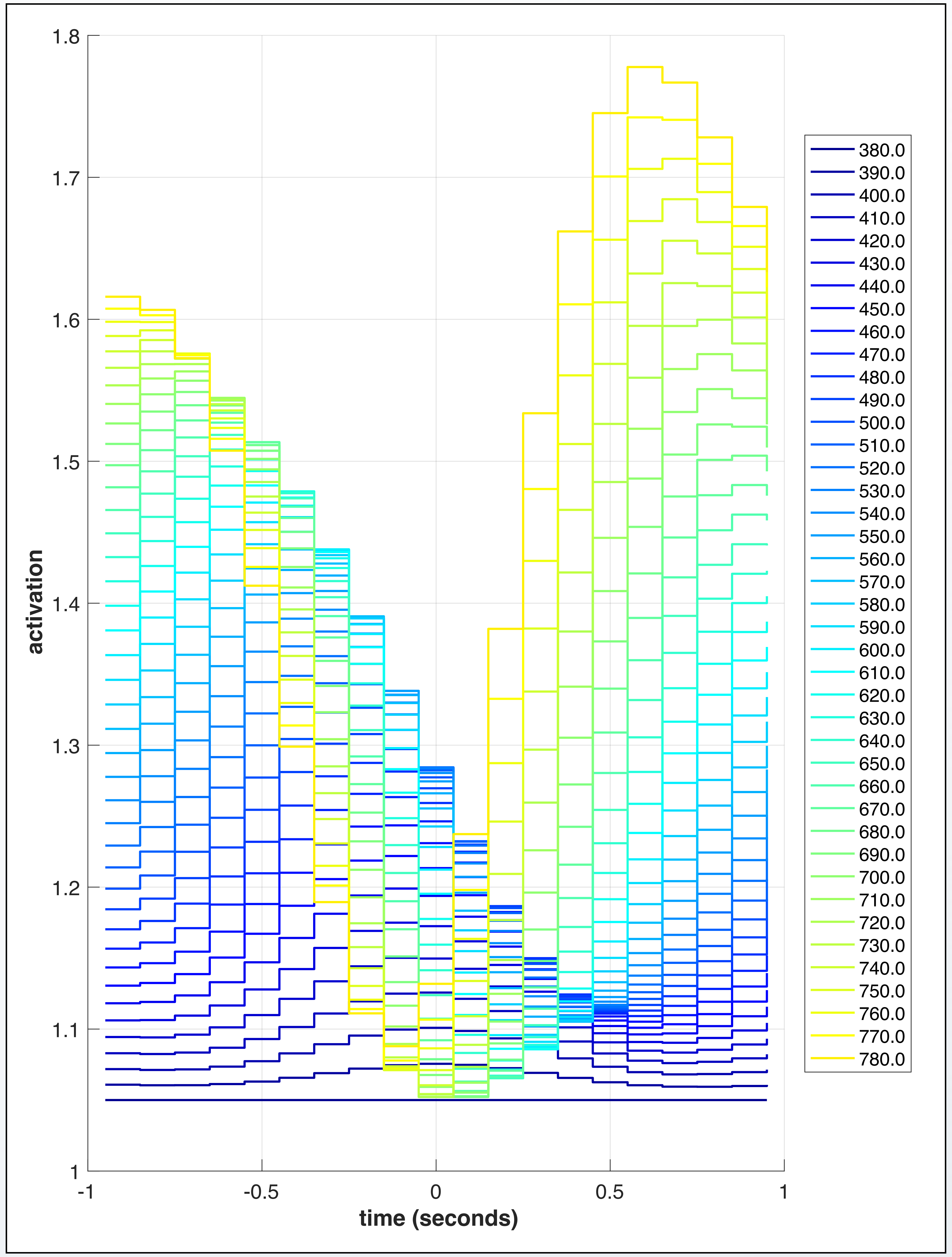
```
% Method to generate an cLcMcS-based stimDescriptor
function theStimDescriptor = cLcMcSStimDescriptor(visualize)

%% Chromatic (LMS cone contrast)  params
descriptor  = 'cLcMcS';
background = 2*[0.06 0.051 0.035];
modulation = [0.07 -0.07 0.3];

%% Temporal params
dt = 50/1000; nSamples = 200; sigmaTau = 2000/1000;
timeBase = (0:(nSamples-1))*dt + dt/2;
timeBase = timeBase - mean(timeBase);
tf = 1.0;

%% Separate time courses for the S-cone vs the L/M-cone channel
timeEnvelope(1,:) = cos(2*pi*tf*timeBase).*exp(-0.5*(timeBase/sigmaTau).^2);
timeEnvelope(2,:) = cos(2*pi*tf*timeBase).*exp(-0.5*(timeBase/sigmaTau).^2);
timeEnvelope(3,:) = cos(2*pi*tf/4*timeBase).*exp(-0.5*(timeBase/sigmaTau).^2);
timeEnvelope = timeEnvelope / max(abs(timeEnvelope(:)));

%% Spatial params
center = [1024/2 768/2]; sigma = 40;
xAxis = center(1) + (-120:120);
yAxis = center(2) + (-120:120);
[X,Y] = meshgrid(xAxis-mean(xAxis), yAxis-mean(yAxis));
spatialFreq = 8/1024;
spatialEnvelope = exp(-0.5*((X/sigma).^2+(Y/sigma).^2)).* ...
    cos(2.0*pi*spatialFreq*(X-Y));
spatialEnvelope = spatialEnvelope/max(abs(spatialEnvelope(:)));

% Make the stimDescriptor struct
theStimDescriptor= StimDescriptor(...
    'colorDescriptor', descriptor, ...
    'background', background, ...
    'modulation', modulation, ...
    'temporalSupport',  timeBase, ...
    'temporalEnvelope', timeEnvelope, ...
    'spatialSupport', {xAxis(:) yAxis(:)}, ...
    'spatialEnvelope', spatialEnvelope ...
);
```



stimDescriptor



```
function theStimDescriptor = spdStimDescriptor(visualize)

%% Chromatic (spd) params
waveAxis = 380:10:780;
backgroundSPD = 0.05 + ones(1, numel(waveAxis));
modulatedSPD = (0:(numel(waveAxis)-1))/numel(waveAxis);

%% Temporal params
dt = 100/1000; nSamples = 20;
timeBase = (0:(nSamples-1))*dt + dt/2;
timeBase = timeBase - mean(timeBase);
temporalFrequencyHz = 0.5;

%% Specify different time courses for each color channel
timeEnvelope = zeros(numel(waveAxis), numel(timeBase));
for waveIndex = 1:numel(waveAxis)
    timeEnvelope(waveIndex,:) = 0.5 + 0.5*cos(2*pi*temporalFrequencyHz*timeBase+0.03*pi*waveIndex).*exp(-0.5*(timeBase/0.50).^2);  % modulation envelope for waveIndex
end

%% Make the stimDescriptor struct
theStimDescriptor = StimDescriptor(...
    'colorDescriptor', waveAxis, ...
    'background', backgroundSPD, ...
    'modulation', modulatedSPD, ...
    'temporalSupport',  timeBase, ...
    'temporalEnvelope', timeEnvelope ...
);
if (visualize)
    theStimDescriptor.visualize();
end
end
```


activation
time (seconds)