```
% Copyright © 2019 Naturalpoint
%    Not for Redistribution
%    NatNet Matlab Class
%    Requires Matlab 2019a or later
%    This class is a wrapper for the NatNetML assembly and controls the
%    connection and communication between the OptiTrack NatNet Server and
%    Matlab clients.
%    Public Properties:
%        ClientIP - The IP address of the client network.
%        ConnectionType - Multicast of Unicast
%        HostIP - The IP address of the host/server network.
%        IsReporting - state of commandline reporting
%        IsConnected - state of connection to server, updated only when
%                      establishing a connection. (Read-only)
%        Mode - state of the Motive server, Live or Edit. (Read-Only)
%        Version - version number of the natnet libraries. (Read-Only)
%        FrameRate - sampling rate of the Motive hardware. (Read-Only)
%    Public Methods:
%        natnet - Object Constructor
%        connect - Establish a connection to an OptiTrack NatNet Server
%        disconnect - Close a connection to an OptiTrack NatNet Server
%        addlistener - Links a callback function to the server stream.

%        getFrameRate - Returns the server Frame Rate.
%        getModelDescription - Returns a structure of the asset list.
%        getFrame - Returns the latest frame of mocap data.

%        getFrameMetaData - Returns last frame of meta data within mocap
%            frame. Return value is a Matlab struct
%        getFrameMarkerSet - Returns the last frame of MarkerSet data
%            within mocap frame. Return value is a Matlab struct
%        getFrameLabeledMarker - Returns the last frame of LabeledMarker
%            data within the mocap frame. Return value is a Matlab
%            struct.

%        getMode - Returns the state of the server: Live or Edit mode.
%        enable - Enables execution of a linked callback function.
%        disable - Disabled execution of a linked callback function.
%        Server Commands
%        startRecord - Starts a recording on the server.
%        stopRecord - Stops a recording on the server.
%        cycleRecord - Loops recording.
%        liveMode - Changes the server to Live mode.
%        editMode - Changes the server to Edit mode.
%        startPlayback - Starts playback of the server in Edit mode.
%        stopPlayback - Stops playback of the server in Edit mode.
%        setPlaybackStartFrame - Sets the playback loop start frame.
%        setPlaybackEndFrame - Sets the playback loop end frame.
%        setPlaybackLooping - Sets playback to loop.
%        setPlaybackCurrentFrame - Sets the current frame in Edit mode.
%        setTakeName - Sets the pending (to be recorded) take name.
%        setPlaybackTakeName - Opens a recorded take in Edit mode.
```

```
%        setCurrentSession - Changes the current folder in Edit mode.
%        delete - Destructor for the natnet class

classdef natnet < handle
    properties ( Access = public )
        ClientIP
        ConnectionType
        HostIP
        IsReporting
    end % properties ( Access = public )


    properties ( SetAccess = private )
        IsConnected
        Mode
        Version
        FrameRate
    end % properties ( SetAccess = private )


    properties ( Access = private )
        Assembly
        AssemblyPath
        AssemblyVersion
        Client
        IsAssemblyLoaded
        LastAssemblyPathFile
        MaxListeners
        Listener
        CReattempt
    end % properties ( Access = private )


    properties ( Access = private , Dependent = true )
        iConnectionType
    end % properties ( Access = private , Dependant = true )


    methods
        function set.HostIP( obj , val )
            validIP = obj.checkip( val);
            if ( ~isempty( validIP ) )
                obj.disconnect
                obj.HostIP = validIP;
                obj.report( [ 'set host ip address: ' , validIP ] )
            end
        end % set.HostIP


        function set.ClientIP( obj , val )
            validIP = obj.checkip( val );
            if ( ~isempty( validIP ) )
                obj.disconnect
                obj.ClientIP = validIP;
```

```
            obj.report( [ 'set client ip address: ' , validIP ] )
        end
    end % set.ClientIP


    function set.ConnectionType( obj , val )
        obj.disconnect
        if ( strcmpi( 'Multicast' , val ) == 1 )
            obj.ConnectionType = 'Multicast';
            obj.report( 'set multicast' )
        elseif ( strcmpi( 'Unicast' , val ) == 1 )
            obj.ConnectionType = 'Unicast';
            obj.report( 'set unicast' )
        else
            obj.report( 'invalid connection type' )
        end
    end % set.ConnectionType


    function set.IsReporting( obj , val )
        if ( val == 1 )
            obj.IsReporting = 1;
            obj.report( 'reporting enabled' )
        elseif ( val == 0 )
            obj.report( 'reporting disabled' )
            obj.IsReporting = 0;
        end
    end % set.IsReporting


    function iconnectiontype = get.iConnectionType( obj )
        if ( strcmpi( 'Multicast' , obj.ConnectionType ) == 1 )
            iconnectiontype = 0;
        elseif ( strcmpi( 'Unicast' , obj.ConnectionType ) == 1 )
            iconnectiontype = 1;
        end
    end % iconnectiontype
end % methods ( property set and get methods)


methods ( Access = public  )
    function obj = natnet(  )
        obj.getLastAssemblyPath
        obj.MaxListeners = 255;
        obj.Listener{ obj.MaxListeners , 1 } = [ ];
        obj.FrameRate = 0;
        obj.IsAssemblyLoaded = 0;
        obj.IsConnected = 0;
        obj.IsReporting = 0;
        obj.CReattempt = 1;
        obj.HostIP = '127.0.0.1';
        obj.ClientIP = '127.0.0.1';
        obj.ConnectionType = 'Multicast';
        obj.Version = ' ';
```

```
            obj.Mode = ' ';
        end % natnet - constructor


        function connect( obj )
            obj.disconnect
            obj.getAssemblies
            if ( obj.IsAssemblyLoaded == 0 )
                if ( isempty( obj.AssemblyPath ) )
                    obj.setAssemblyPath
                    obj.addAssembly
                    obj.getAssemblies
                else
                    obj.addAssembly
                    obj.getAssemblies
                end
            end
            if ( obj.IsAssemblyLoaded == 0 )
                report( obj , 'natnetml assembly is missing or undefined' )
                return
            end
            try
                obj.Client = NatNetML.NatNetClientML( obj.iConnectionType );
                v = obj.Client.NatNetVersion(  );
                obj.Version = sprintf( '%d.%d.%d.%d', v( 1 ) , v( 2 ) ,
v( 3 ) , v( 4 ) );
                if  ( isempty( obj.Client ) == 1 || obj.IsAssemblyLoaded ==
0 )
                    obj.report( 'client object invalid' )
                    return
                else
                    obj.report( [ 'natnet version: ' , obj.Version ] )
                end
                flg = obj.Client.Initialize( obj.ClientIP , obj.HostIP );
                if flg == 0
                    obj.report( 'client initialized' )
                else
                    obj.report( 'initialization failed' )
                    obj.IsConnected = 0;
                    return
                end
            catch exception
                rethrow(exception)
            end

            [ ~ , rc] = obj.Client.SendMessageAndWait( 'FrameRate' );
            if rc == 0
                obj.IsConnected = 1;
                obj.getFrameRate
                obj.getMode
                obj.getModelDescription;
                return
            elseif( rc == 1 )
                obj.report( 'connection failed due to an internal error' )
```

```
            elseif ( rc == 2 )
                obj.report( 'connection failed due to an external error' )
            elseif ( rc == 3 )
                obj.report( 'connection failed due to a network error' )
            else
                obj.report( 'connection failed due to an unknown error' )
            end
            obj.IsConnected = 0;
        end % connect


        function addlistener( obj , aListenerIndex , functionhandlestring )
            if ~isnumeric( aListenerIndex ) || aListenerIndex < 1 ||
aListenerIndex > obj.MaxListeners
                obj.report( 'invalid index' );
                return
            end

            if isa( functionhandlestring , 'char' ) &&
~isempty( which( functionhandlestring ) )
                functionhandle = str2func ( [ '@( src , evnt)' ,
functionhandlestring , '( src , evnt)' ] );

                if ~isa( obj.Listener{ aListenerIndex , 1 } ,
'event.listener' )
                    obj.report( [ 'adding listener in slot:' ,
num2str(aListenerIndex ) ] );
                    obj.Listener{ aListenerIndex , 1 } =
addlistener( obj.Client , 'OnFrameReady2' , functionhandle );
                    obj.disable( aListenerIndex );
                elseif isa( obj.Listener{ aListenerIndex , 1 } ,
'event.listener' )
                    obj.report( [ 'replacing listener in slot: ' ,
num2str(aListenerIndex ) ] )
                    obj.disable( aListenerIndex )
                    delete( obj.Listener{ aListenerIndex , 1 } )
                    obj.Listener{ aListenerIndex , 1 } =
addlistener( obj.Client, 'OnFrameReady2' , functionhandle );
                    obj.disable ( aListenerIndex )
                end
            else
                obj.report( [ 'invalid function set for slot: ' ,
num2str( aListenerIndex ) ] )
                return
            end
        end % addlistener


        function getFrameRate( obj )
            [ bytearray , rc ] = obj.sendMessageAndWait( 'FrameRate' );
            if rc == 0
                bytearray = uint8( bytearray );
                obj.FrameRate =  typecast( bytearray , 'single' );
                obj.report( [ 'frame rate: ' , num2str( obj.FrameRate ) , '
```

```
fps' ] )
            else
                obj.FrameRate = 0;
            end
        end % getFrameRate


        function modelDescription = getModelDescription( obj )
            if ( obj.IsConnected == 1 )
                dataDescriptions = obj.Client.GetDataDescriptions(  );
                modelDescription.TrackingModelCount = dataDescriptions.Count;
                report( obj , [ 'number of trackables: ' ,
num2str( dataDescriptions.Count ) ] )
                modelDescription.MarkerSetCount = 0;
                modelDescription.RigidBodyCount = 0;
                modelDescription.SkeletonCount = 0;
                modelDescription.ForcePlateCount = 0;
                modelDescription.DeviceCount = 0;
                modelDescription.CameraCount = 0;

                for i = 1 : modelDescription.TrackingModelCount
                    descriptor = dataDescriptions.Item( i - 1 );
                    % marker sets
                    if ( descriptor.type == 0)
                        modelDescription.MarkerSetCount =
modelDescription.MarkerSetCount + 1;

modelDescription.MarkerSet( modelDescription.MarkerSetCount ).Name =
char( descriptor.Name );

modelDescription.MarkerSet( modelDescription.MarkerSetCount ).MarkerCount =
descriptor.nMarkers;
                        for k = 1 : descriptor.nMarkers

modelDescription.MarkerSet( modelDescription.MarkerSetCount ).Markers( k ).La
bel = char( descriptor.MarkerNames( k ) );
                        end
                    % rigid bodies
                    elseif ( descriptor.type == 1 )
                        modelDescription.RigidBodyCount =
modelDescription.RigidBodyCount + 1;

modelDescription.RigidBody( modelDescription.RigidBodyCount ).Name =
char( descriptor.Name );

modelDescription.RigidBody( modelDescription.RigidBodyCount ).ID =
descriptor.ID ;

modelDescription.RigidBody( modelDescription.RigidBodyCount ).ParentID =
descriptor.parentID;

modelDescription.RigidBody( modelDescription.RigidBodyCount ).OffsetX =
descriptor.offsetx;
```

```
modelDescription.RigidBody( modelDescription.RigidBodyCount ).OffsetX =
descriptor.offsetx;

modelDescription.RigidBody( modelDescription.RigidBodyCount ).OffsetX =
descriptor.offsetx;

modelDescription.RigidBody( modelDescription.RigidBodyCount ).Type =
descriptor.type;
                    % skeletons
                    elseif ( descriptor.type == 2 )
                        modelDescription.SkeletonCount =
modelDescription.SkeletonCount + 1;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Name =
char( descriptor.Name );

modelDescription.Skeleton( modelDescription.SkeletonCount ).ID =
descriptor.ID;

modelDescription.Skeleton( modelDescription.SkeletonCount ).SegmentCount =
descriptor.nRigidBodies;
                        for k = 1 :
modelDescription.Skeleton( modelDescription.SkeletonCount ).SegmentCount

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Name
 = char( descriptor.RigidBodies( k ).Name );

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).ID
= descriptor.RigidBodies( k ).ID;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Pare
ntID = descriptor.RigidBodies( k ).parentID;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Offs
etX = descriptor.RigidBodies( k ).offsetx;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Offs
etY = descriptor.RigidBodies( k ).offsety;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Offs
etZ = descriptor.RigidBodies( k ).offsetz;

modelDescription.Skeleton( modelDescription.SkeletonCount ).Segment( k ).Type
 = descriptor.RigidBodies( k ).type;
                        end
                    elseif ( descriptor.type == 3 )
                        modelDescription.ForcePlateCount =
modelDescription.ForcePlateCount + 1;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Serial =
char( descriptor.Serial );

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).ID =
descriptor.ID;
```

```
modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Width =
descriptor.Width;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Length =
descriptor.Length;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Origin.X =
descriptor.OriginX;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Origin.Y =
descriptor.OriginY;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Origin.Z =
descriptor.OriginZ;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).CalibrationMa
trix = descriptor.CalMatrix;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).CornerCount
= descriptor.Corners.Length;
                        for i = 1 :
modelDescription.ForcePlate( modelDescription.ForcePlateCount ).CornerCount/3

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Corner( i ).X
 = descriptor.Corners( i*3-2 );

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Corner( i ).Y
 = descriptor.Corners( i*3-1 );

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Corner( i ).Z
 = descriptor.Corners( i*3 );
                        end

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).PlateType =
descriptor.PlateType;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).ChannelDataTy
pe = descriptor.ChannelDataType;

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).ChannelCount
= descriptor.ChannelCount;
                        for i = 1 :
modelDescription.ForcePlate( modelDescription.ForcePlateCount ).ChannelCount

modelDescription.ForcePlate( modelDescription.ForcePlateCount ).Channel( i ).
Name = char( descriptor.ChannelNames( i ) );
                        end
                    elseif (descriptor.type == 4)
                        modelDescription.DeviceCount =
modelDescription.DeviceCount + 1;
                    elseif (descriptor.type == -1 || descriptor.type == 5)
                        modelDescription.CameraCount =
modelDescription.CameraCount + 1;
```

```
                    else
                        report( obj , 'invalid asset type' )
                    end
                end
                report( obj , [ 'number of markersets: ' ,
num2str( modelDescription.MarkerSetCount ) ] )
                report( obj , [ 'number of rigid bodies: ' ,
num2str( modelDescription.RigidBodyCount ) ] )
                report( obj , [ 'number of skeletons: ' ,
num2str( modelDescription.SkeletonCount ) ] )
                report( obj , [ 'number of force plates: ' ,
num2str( modelDescription.ForcePlateCount ) ] )
                report( obj , [ 'number of devices: ' ,
num2str( modelDescription.DeviceCount ) ] )
                report( obj , [ 'number of cameras: ' ,
num2str( modelDescription.CameraCount ) ] )
            else
                obj.report( 'connection not established' )
            end
        end % getModelDescription


        function frameOfMocapData = getFrame( obj )
            if ( obj.IsConnected == 1 )
                data = obj.Client.GetLastFrameOfData(  );
                frameOfMocapData = data;
            else
                obj.report( 'connection not established' )
            end
        end % getFrame


        function frameOfMetaData = getFrameMetaData( obj )
            persistent data
            if ( obj.IsConnected == 1 )
                data = obj.Client.GetLastFrameOfData(  );
                t.Frame = data.iFrame;
                t.Timestamp = data.fTimestamp;
                t.CameraMidExposureTimestamp =
data.CameraMidExposureTimestamp;
                t.CameraDataReceivedTimestamp =
data.CameraDataReceivedTimestamp;
                t.TransmitTimestamp = data.TransmitTimestamp;
                t.Recording = data.bRecording;
                t.TrackingModelsChanged = data.bTrackingModelsChanged;
                t.Timecode = data.Timecode;
                t.TimecodeSubframe = data.TimecodeSubframe;
                t.MarkerCount = data.nMarkers;
                t.OtherMarkerCount = data.nOtherMarkers;
                t.MarkerSetCount = data.nMarkerSets;
                t.RigidBodyCount = data.nRigidBodies;
                t.SkeletonCount = data.nSkeletons;
                t.ForcePlateCount = data.nForcePlates;
                t.DeviceCount = data.nDevices;
```

```
                frameOfMetaData = t;
            else
                obj.report( 'connection not established' )
            end
        end % getFrameOfMetaData


        function frameOfLabeledMarkerData = getFrameLabeledMarker( obj )
            if ( obj.IsConnected == 1 )
                data = obj.Client.GetLastFrameOfData(  );
                t.Frame = data.iFrame;
                t.Timestamp = data.fTimestamp;
                t.MarkerCount = data.nMarkers;
                for i = t.MarkerCount : -1 : 1
                    t.LabeledMarker( i ).AssetID =
int16( bitshift( data.LabeledMarkers( i ).ID , -16 ) );
                    t.LabeledMarker( i ).MemberID = int16( bitshift
( bitshift( data.LabeledMarkers( i ).ID , 16 ) , -16 ) );
                    t.LabeledMarker( i ).X = data.LabeledMarkers( i ).x;
                    t.LabeledMarker( i ).Y = data.LabeledMarkers( i ).y;
                    t.LabeledMarker( i ).Z = data.LabeledMarkers( i ).z;
                    t.LabeledMarker( i ).Size =
data.LabeledMarkers( i ).size;
                    %t.LabeledMarker( i ).Occluded =
logical( bitget( data.LabeledMarkers( i ).parameters , 1 ) );
                    t.LabeledMarker( i ).PointCloudSolved =
logical( bitget( data.LabeledMarkers( i ).parameters , 2 ) );
                    t.LabeledMarker( i ).AssetSolved =
logical( bitget( data.LabeledMarkers( i ).parameters , 3 ) );
                    t.LabeledMarker( i ).AssetMarker =
logical( bitget( data.LabeledMarkers( i ).parameters , 4 ) );
                    t.LabeledMarker( i ).UnlabeledMarker =
logical( bitget( data.LabeledMarkers( i ).parameters , 5 ) );
                    t.LabeledMarker( i ).ActiveMarker =
logical( bitget( data.LabeledMarkers( i ).parameters , 6 ) );
                    t.LabeledMarker( i ).Residual =
data.LabeledMarkers( i ).residual;
                end
                frameOfLabeledMarkerData = t;
            else
                obj.report( 'connection not established' )
            end
        end % getFrameLabeledMarker


        function frameOfMarkerSetData = getFrameMarkerSet( obj )
            if ( obj.IsConnected == 1 )
                data = obj.Client.GetLastFrameOfData(  );
                t.Frame = data.iFrame;
                t.Timestamp = data.fTimestamp;
                t.MarkerSetCount = data.nMarkerSets;
                for i = t.MarkerSetCount : -1 : 1
                    t.MarkerSet( i ).MarkerSetName =
string( data.MarkerSets( i ).MarkerSetName );
```

```
                    t.MarkerSet( i ).MarkerCount =
data.MarkerSets( i ).nMarkers;
                    for j = t.MarkerSet( i ).MarkerCount : -1 : 1
                        %t.MarkerSet( i ).Marker( j ).AssetID =
int16( bitshift( data.MarkerSets( i ).Markers( j ).ID , -16 ) );
                        %t.MarkerSet( i ).Marker( j ).MemberID =
int16( bitshift ( bitshift( data.MarkerSets( i ).Markers( j ).ID , 16 ) ,
-16 ) );
                        t.MarkerSet( i ).Marker( j ).X =
data.MarkerSets( i ).Markers( j ).x;
                        t.MarkerSet( i ).Marker( j ).Y =
data.MarkerSets( i ).Markers( j ).y;
                        t.MarkerSet( i ).Marker( j ).Z =
data.MarkerSets( i ).Markers( j ).z;
                        %t.MarkerSet( i ).Marker( j ).Size =
data.MarkerSets( i ).Markers( j ).size;
                        %t.MarkerSet( i ).Marker( j ).Occluded =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 1 ) );
                        %t.MarkerSet( i ).Marker( j ).PointCloudSolved =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 2 ) );
                        %t.MarkerSet( i ).Marker( j ).AssetSolved =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 3 ) );
                        %t.MarkerSet( i ).Marker( j ).AssetMarker =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 4 ) );
                        %t.MarkerSet( i ).Marker( j ).UnlabeledMarker =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 5 ) );
                        %t.MarkerSet( i ).Marker( j ).ActiveMarker =
logical( bitget( data.MarkerSets( i ).Markers( j ).parameters , 6 ) );
                        %t.MarkerSet( i ).Marker( j ).Residual =
data.MarkerSets( i ).Markers( j ).residual;
                    end
                end
                frameOfMarkerSetData = t;
            else
                obj.report( 'connection not established' )
            end
        end % getFrameMarkerSet


        function enable( obj , eListenerIndex )
            if ~isnumeric( eListenerIndex ) || eListenerIndex < 0 ||
eListenerIndex > obj.MaxListeners
                obj.report( 'invalid index' )
                return
            end

            if obj.IsConnected == 0
                return
            end

            if eListenerIndex == 0
                for k = 1:obj.MaxListeners
                    if isa( obj.Listener{ k ,  1} , 'event.listener' ) &&
obj.Listener{ k ,  1}.Enabled == false
```

```
                        obj.Listener{ k , 1  }.Enabled = true;
                        obj.report( [ 'listener enabled in slot: ' ,
num2str( k ) ] )
                    end
                end
            else
                if( isa( obj.Listener{ eListenerIndex , 1 } ,
'event.listener' ) ) && obj.Listener{ eListenerIndex , 1 }.Enabled == false
                    obj.Listener{ eListenerIndex , 1 }.Enabled = true;
                    obj.report( [ 'listener enabled in slot: ' ,
num2str( eListenerIndex ) ] )
                end
            end
        end % enable


        function disable( obj , dListenerIndex );
            if ~isnumeric( dListenerIndex ) || dListenerIndex < 0 ||
dListenerIndex > obj.MaxListeners
                obj.report( 'invalid index' );
                return
            end

            if obj.IsConnected == 0;
                return
            end

            if dListenerIndex == 0
                for k = 1:obj.MaxListeners
                    if isa( obj.Listener{ k , 1 } , 'event.listener'  )&&
obj.Listener{ k ,  1}.Enabled == true;
                        obj.Listener{ k }.Enabled = false;
                        obj.report( [ 'listener disabled in slot: ' ,
num2str( k ) ] )
                    end
                end
            else
                if( isa( obj.Listener{ dListenerIndex , 1 } ,
'event.listener' ) ) && obj.Listener{ dListenerIndex , 1 }.Enabled == true;
                    obj.Listener{ dListenerIndex , 1 }.Enabled = false;
                    obj.report( [ 'listener disabled in slot: ' ,
num2str( dListenerIndex ) ] )
                end
            end
        end % disable


        function startRecord( obj )
            obj.sendMessageAndWait( 'StartRecording' );
        end % startRecord


        function stopRecord( obj )
            obj.sendMessageAndWait( 'StopRecording' );
```

```
        end % stopRecord


        function cycleRecord( obj , iterations , duration , delay )
            if ( isnumeric( iterations ) && isnumeric( duration ) &&
isnumeric( delay ) );
                for i = 1:iterations
                    pause( delay );
                    obj.startRecord;
                    pause( duration );
                    obj.stopRecord;
                end
            end
        end % cycleRecord


        function liveMode( obj )
            obj.sendMessageAndWait( 'LiveMode' );
        end % liveMode


        function editMode( obj )
            obj.sendMessageAndWait( 'EditMode' );
        end % editMode


        function startPlayback( obj )
            obj.sendMessageAndWait( 'TimelinePlay' );
        end % startPlayback


        function stopPlayback( obj )
            obj.sendMessageAndWait( 'TimelineStop' );
        end % stopPlayback


        function setPlaybackStartFrame( obj , startFrame )
            obj.sendMessageAndWait( [ 'SetPlaybackStartFrame,' ,
num2str( startFrame ) ] );
        end % setPlaybackStartFrame


        function setPlaybackEndFrame( obj , endFrame )
            obj.sendMessageAndWait( [ 'SetPlaybackStopFrame,' ,
num2str( endFrame ) ] );
        end % setPlaybackEndFrame


        function setPlaybackLooping( obj , val )
            obj.sendMessageAndWait( [ 'SetPlaybackLooping,' ,
num2str( val ) ] );
        end % stopPlaybackLooping
```

```
        function setPlaybackCurrentFrame( obj , currentFrame )
            obj.sendMessageAndWait( [ 'SetPlaybackCurrentFrame,' ,
num2str( currentFrame ) ] );
        end % stopPlaybackLooping


        function setTakeName( obj , name )
            obj.sendMessageAndWait( strcat( 'SetRecordTakeName,' , name ) );
        end % setTakeName


        function setPlaybackTakeName( obj , name )
            obj.sendMessageAndWait( strcat( 'SetPlaybackTakeName,' ,
name ) );
        end % setPlaybackTakeName


        function setCurrentSession( obj , name )
            obj.sendMessageAndWait( strcat('SetCurrentSession,' , name) );
        end % setCurrentSession


        function getMode( obj )
            [ bytearray , rc ] = obj.sendMessageAndWait( 'CurrentMode' );
            if rc == 0
                state = bytearray(1);
                if ( state == 0 )
                    obj.Mode = 'Live';
                elseif ( state == 1 )
                    obj.Mode = 'Recording';
                elseif ( state == 2 )
                    obj.Mode = 'Edit';
                else
                    obj.Mode = ' ';
                end
                obj.report( [ 'mode: ' , lower( obj.Mode ) ] )
                return
            else
                obj.Mode = ' ';
            end
        end % getServerState


        function disconnect( obj )
            obj.disable( 0 );
            for k = 1 : obj.MaxListeners
                if isa( obj.Listener{ k , 1 } , 'event.listener' )
                    delete( obj.Listener{ k , 1 } )
                end
            end
            obj.Listener{ obj.MaxListeners , 1 } = [  ];
            if ( isempty( obj.Client ) == 0 )
                obj.Client.Uninitialize;
                obj.Client = [  ];
```

```
                if ( obj.IsConnected == 1 )
                    report( obj , 'disconnecting' );
                end
                obj.IsConnected = 0;
            end
        end % disconnect


        function delete( obj )
            obj.disconnect
        end % delete - destructor
    end % methods ( Access = public )


    methods ( Access = private )
        function getLastAssemblyPath( obj )
            pathtomfile = mfilename( 'fullpath' );
            mfile = mfilename(  );
            mfilenamelength = length( mfile );
            foldertomfile = pathtomfile( 1 : end-mfilenamelength );
            obj.LastAssemblyPathFile = strcat( foldertomfile ,
'assemblypath.txt' );
            if ( exist( obj.LastAssemblyPathFile , 'file' ) == 2 )
                assemblypath = textread( obj.LastAssemblyPathFile , '%s' );
                assemblypath = strjoin( assemblypath );
                if( exist( assemblypath , 'file' ) == 2)
                    obj.AssemblyPath = assemblypath;
                end
            end
        end % getLastAssemblyPath


        function getAssemblies( obj )
            obj.IsAssemblyLoaded = 0;
            obj.Assembly = [  ];
            obj.AssemblyVersion = [  ];
            domain = System.AppDomain.CurrentDomain;
            assemblies = domain.GetAssemblies;
            assembly{ assemblies.Length , 1 } = [  ];
            for i = 1:assemblies.Length
                assembly{ i } = assemblies.Get( i-1 );
                obj.Assembly{ i } = char( assembly{ i }.FullName );
                if ( strncmp ( obj.Assembly{ i } , 'NatNetML' , 8 ) )
                    aver = regexp( obj.Assembly{ i } , '\d+' , 'match' );
                    obj.AssemblyVersion = strcat( aver{ 1 } , '.' ,
aver{ 2 } , '.' , aver{ 3 } , '.' , aver{ 4 } );
                    obj.IsAssemblyLoaded = 1;
                end
            end
        end % getAssemblies


        function addAssembly( obj )
            if ( exist( obj.AssemblyPath , 'file' ) == 2 )
```

```
                [ ~ ] = NET.addAssembly( obj.AssemblyPath );
                obj.IsAssemblyLoaded = 1;
            else
                obj.IsAssemblyLoaded = 0;
                report( obj , 'failed to add NatNetML.dll assembly' );
            end
        end % addAssembly


        function setAssemblyPath( obj )
            [ name , path ] =    uigetfile( '*.dll' , 'Select the
NatNetML.dll assembly - NatNetLib.dll is a dependency' );
            if ( strcmpi( name , 'NatNetML.dll') == 1 )
                assemblyPath = strcat( path , name );
                textAssemblyPath = strrep( assemblyPath , '\' , '\\' );
                fileid = fopen( obj.LastAssemblyPathFile , 'wt');
                fprintf( fileid , '%s' , assemblyPath );
                fclose( fileid );
                if ( obj.IsAssemblyLoaded == 0 )
                    obj.AssemblyPath = assemblyPath;
                    obj.report( [ 'defined assembly path: ' ,
textAssemblyPath ] )
                else
                    obj.report( [ 'redefined assembly path: ' ,
textAssemblyPath ] )
                    obj.report( 'restart matlab to apply the changes' )
                end
            else
                obj.report( 'invalid assembly path in function
setAssemblyPath' )
            end
        end % setAssemlbyPath


        function report( obj , message )
            if ( obj.IsReporting == 1 )
                ctime = strsplit( num2str( clock ) );
                disp( sprintf ( [ ctime{ 1 } , '/' , ctime{ 2 } , '/' ,
ctime{ 3 } , '  ' , ctime{ 4 } , ':' , ctime{ 5 } , ':' , ctime{ 6 } , '
' , 'natnet - ' , message ] ) )
            end
        end % report


        function validIP = checkip( obj , value )
            if ( ischar( value ) && length( value ) < 16 && length( value )
> 6 )
                val = strsplit( value , '.' );
                if ( length( val ) == 4 )
                    if all( isstrprop( val{ 1 } , 'digit' ) ) &&
all( isstrprop( val{ 2 } , 'digit' ) )...
                            && all( isstrprop( val{ 3 } , 'digit' ) ) &&
all( isstrprop( val{ 4 } , 'digit' ) )...
                            && length( val{ 1 } ) < 4 && length( val{ 2 } )
```

```
< 4 &&  length( val{ 3 } ) < 4 && length( val{ 4 } ) < 4 ...
                            && ~isempty( val{ 1 } ) && ~isempty( val{ 2 } )
&&  ~isempty( val{ 3 } ) && ~isempty( val{ 4 } ) ...
                            && str2double( val{ 1 } ) < 256 &&
str2double( val{ 2 } ) < 256 && str2double( val{ 3 } ) < 256 &&
str2double( val{ 4 } ) < 256 ...
                            && str2double( val{ 1 } ) >= 0  &&
str2double( val{ 2 } ) >= 0 && str2double( val{ 3 } ) >= 0 &&
str2double( val{ 4 } ) >= 0
                            validIP = value;
                        else
                            report( obj , 'invalid string for ip address (e.x.
127.0.0.1)' )
                            validIP = [  ];
                        end
                    else
                        report( obj , 'invalid string for ip address (e.x.
127.0.0.1)' )
                        validIP = [  ];
                    end
                else
                    report( obj , 'invalid string for ip address (e.x.
127.0.0.1)' )
                    validIP = [  ];
                end
            end % validIP


            function [ bytearray , rc ] = sendMessageAndWait( obj , cmd )
                if ( obj.IsConnected == 1 );
                    [ bytearray , rc ] = obj.Client.SendMessageAndWait( cmd );
                    if ( rc == 0 )
                        return
                    else
                        obj.report( 'command failed due to an unknown error' )
                    end
                else
                    bytearray = '';
                    rc = '';
                    obj.report( 'connection not established' )
                end
            end % sendMessageAndWait
        end % methods ( Access = private )
    end % classdef natnet < handle


    ans =

      natnet with properties:

              ClientIP: '127.0.0.1'
        ConnectionType: 'Multicast'
                HostIP: '127.0.0.1'
           IsReporting: 0
```

```
    IsConnected: 0
           Mode: ' '
        Version: ' '
      FrameRate: 0
```

Published with MATLAB® R2023b