SCCT

subscriber library for

User Guide

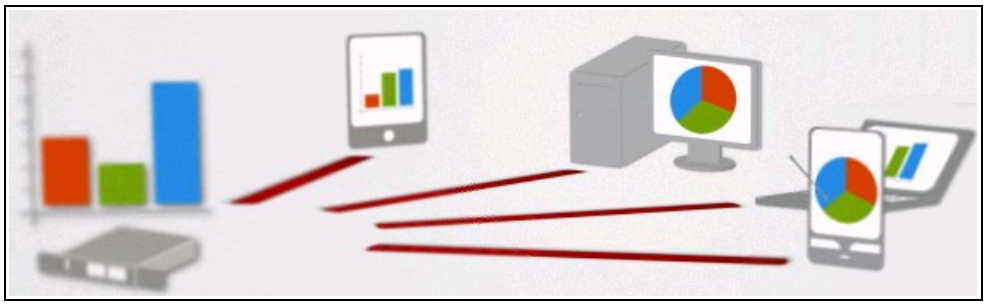
Release 2.1 – July 2012 Edition

Worldwide technical support and product information:
http://www.geminilab.org
http://www.toolsforsmartminds.com
TOOLS for SMART MINDS Corporate headquarter
Via Padania, 16 Castel Mella 25030 Brescia (Italy)
Copyright © 2011 Tools for Smart Minds. All rights reserved.

1. Introduction

1.1 Overview

The Smartphone & Cross-platform Communication Toolkit is an add-on package for sending and receiving data through applications running on multiple platforms at the same time. The toolkit contains a set of high level functions for sending and receiving your application data and advanced functions for customized tasks.



The following list describes the main features of the SCCT:

- Works over any TCP/IP connection
- Implements the publisher – subscriber pattern (also known as Observer pattern)
- Authenticates subscribers through a API-KEY.
- Controls in background the state of every connection to identify loss of communication.
- Publishes GPS coordinates to manage mobile systems.
- Works with platform independent data format and communicate with multiple platforms at the same time: third party vendors have implemented toolkit to develop on Android platform, Java, .NET and VB, Unix/Linux and iOs. Because of the wide range of devices the Smartphone & Cross-platform Communication Toolkit works with, some portability issues remain. Consider the following issues when choosing your way to publish data:

  - *Some smart phones and tablets uses CPU with low computing power so are not able to receive and process large streams of data.*

  - *Smartphone & Cross-platform Communication Toolkit uses a platform independent data format and subscribers require a some computing power to decode data streams into their specific binary format.*

  - *Smartphone & Cross-platform Communication Toolkit handles communication with subscribers as a set of peer to peer connections and every data you publish is transmitted individually to each subscriber. So you have to identify the right size of your data streams to avoid band saturation over your communication channel.*

  - *Some data types are not supported on all platforms.*

1.2 Top reasons to use SCCT

Adopting this toolkit you have the following advantages:

Don't re-invent the wheel: don't care about communication details over a TCP communication channel, SCCT does it for you.

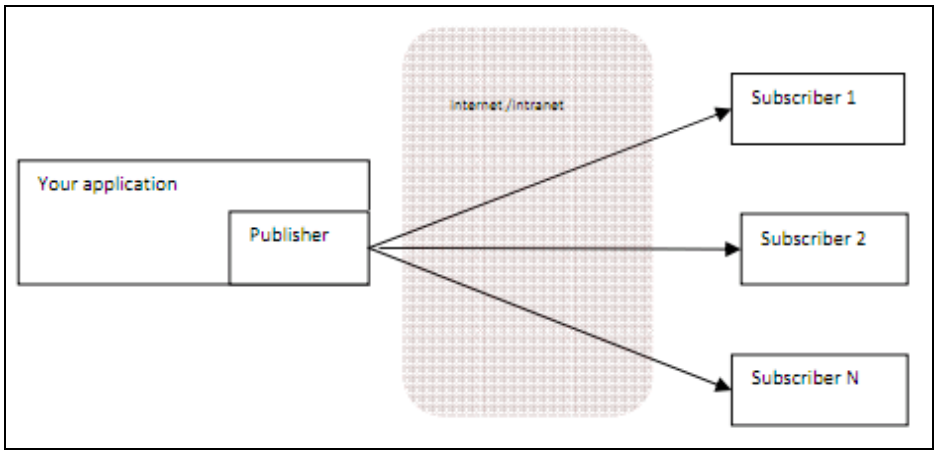
Multiple platforms are supported: exchange your data with a protocol supported on a wide range of platforms and programming languages.

It's reliable: many applications have been created with this toolkit around the world.

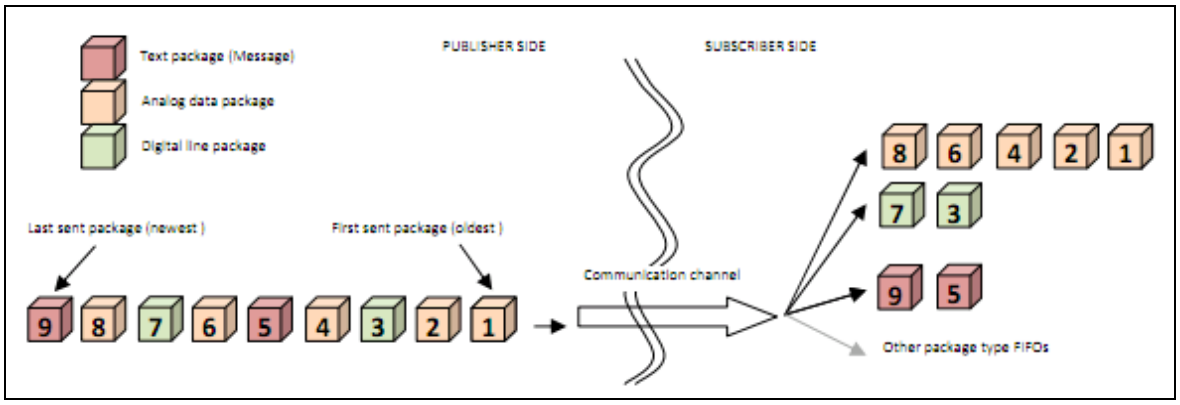
Speed up your development activity: this toolkit simplifies the creation of distributed application and let you save a lot of your time.

1.3 Communication concepts

Smartphone & Cross-platform Communication Toolkit implements the publisher-subscriber pattern. This well known pattern is also called Observer pattern. In this pattern you have one application (publisher) that receives the data you want to publish, and one to many applications (subscribers) which subscribe the service. To subscribe the service and receive fresh data from the publisher, they must use an API-KEY to be authenticated. The following figure represents the pattern:



When an application wants to receive data, asks the publisher to be inscribed among the active subscribers. Publisher will accept all incoming requests with the valid API-KEY. Received data are organised into separated FIFOs so that subscriber application can process data packages according to their types. In the following figure it's shown the case where some different type packages are published according to server logic and subscriber side task organizes packages (nine in the example) into different FIFO structures.



1.4 Publisher and subscriber libraries

Smartphone & Cross-platform Communication Toolkit implements the publisher-subscriber pattern. This well known pattern is also called Observer pattern. In this pattern you have one application (publisher) that receives the data you want to publish, and one to many applications (subscribers) which subscribe the service. To subscribe the service and receive fresh data from the publisher, they must use an API-KEY to be authenticated.

The complete SCCT is so composed by two main components:

- publisher library
- subscriber library

The first (publisher library) let you create a full-featured publisher, which authenticates incoming subscribers, check connection status, sends data to all active publishers and passes their request to your application. This library is available as a set of Vis for LabVIEW 2010 or later. To get more details or download an evaluation copy of this library please visit:

http://www.toolsforsmartminds.com/products/SCCT.php

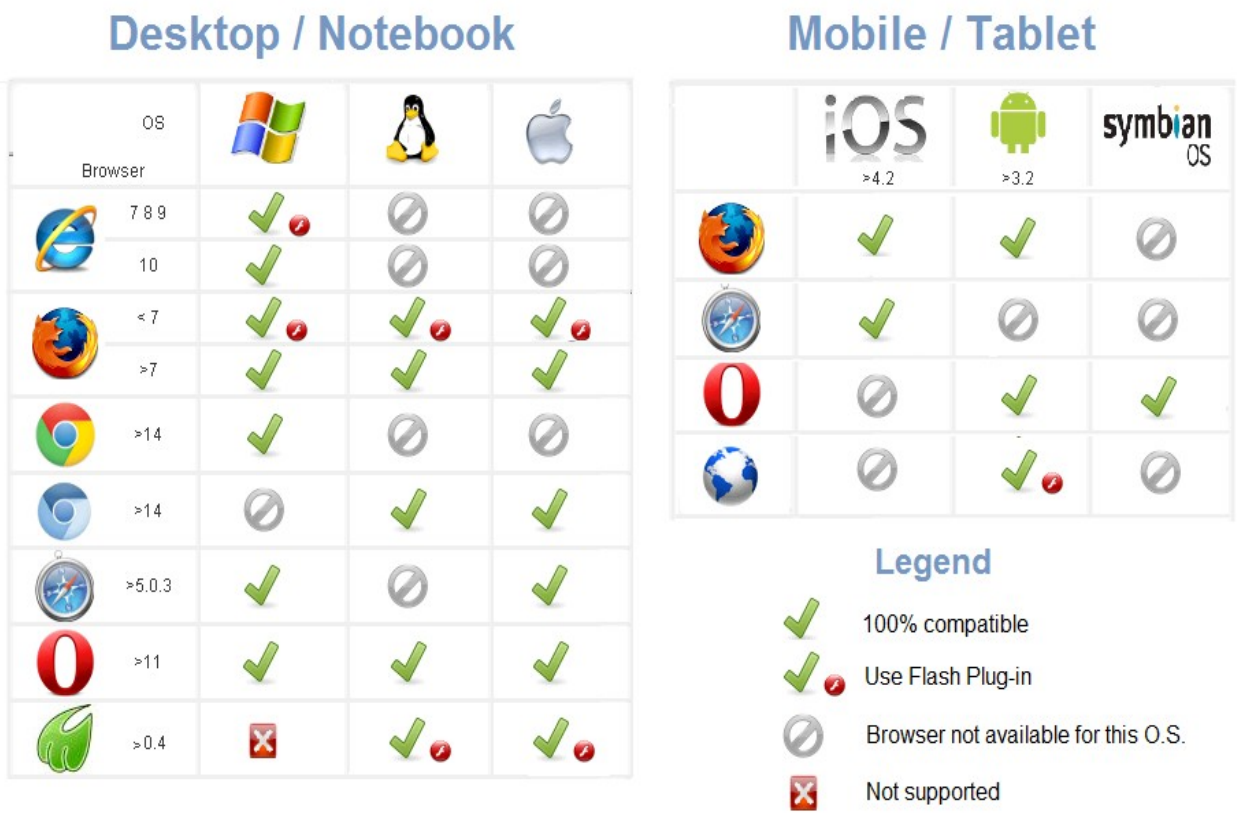
This second (subscriber library) let you create a subscriber which handles all communication details with a publisher so you don't have to. It receives data packages and present them to your application according to their data types. This library is provided for a wide variety of platforms (a complete list is available at http://www.toolsforsmartminds.com/products/SCCT.php). In this document we will focus only on the subscriber library for HTML5.

2. Getting started

2.1 Supported platforms and requirements

SCCT for HTML5 is a pure JavaScript library created to be used inside a browser. This means that is has been widely tested on browser-based JavaScript engine, and not for standalone or custom JavaScript engines.
SCCT for HTML5 has been built above the *LibGeminiSocket* library. This library permits to create the TCP/IP bidirectional channel needed for communicate with the SCCT publisher.
The SCCT server side library implements GeminiLab technology (http://www.geminilab.org) that permits browser to reach the final TCP/IP server.

As a browser-based JavaScript library, SCCT does not need special hardware or operating system requirements. Its compatibility table (the same of the LibGeminiSocket library) is show below.



2.2 Installation

The SCCT HTML5 library is provided by a classic zip file. If you have a graphical front-end such desktop manager, you can use your specific front-end to open the archive.

After the unzip of the file, it should be visible only one directory called “libssct”. You can place this directory anywhere inside your web server file system (or file system in general if you are going to use the library offline without a web server).

2.2.1 Installation for webapps inside pages retrieved by a webserver

Simply put libscct directory on your webserver. The library is ready to be used from Web Applications executed via http (this means that you have to test them using a web server and **not opening it directly from file system**). See section 4 for library usage.

The installation of a local or remote webserver (usually Apache or IIS) goes beyond the purpose of this document.

2.2.2 Installation for webapps executed directly from local file system

The use case in which the page is directly present on a file system as a normal file is not widespread. In fact, normally JavaScript is used inside web pages stored on a web server and accessed via the http protocol. Anyway, it is still possible to build applications inside html files that are stored on a local directory. In this case, the user should be care about **some security considerations**.

In the case of native WebSockets compatibilities, GeminiSocket technology does not need special adjustments in the case on offline execution. The case is different in Flash Bridge cases (normally inside Internet Explorer browsers).

Using to the “supported platforms table” as a graphic reference (section 2), **this security considerations must be considered only in the cells that contain flash icon.**

Browsers that need these considerations are reported in the following list**.**

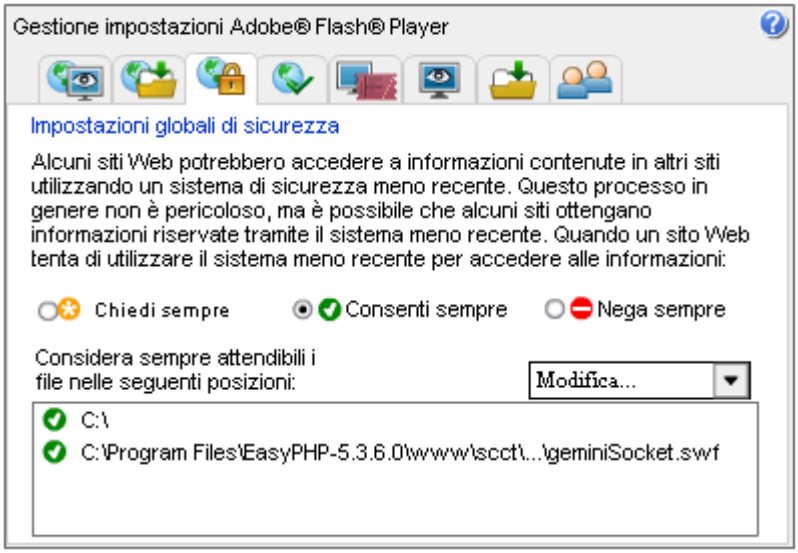
- Firefox versions before or equal 7 with flash player installed
- Internet Explorer 7,8,9 (previous versions are not supported) with flash player plugin insalled
- Midori on Linux or Mac OsX
- Android browser with flash player installed

In this case it is necessary to explicit allow the web page that cointains GeminiSocket code to execute the small piece of ActionScript code with no restrictions. To do this, you have to choose a path inside your local pc or notebook, and then set it as “trusted” inside the browser Flash plugin “**Settings Manager**”, especially on the “**Global Security Settings Panel**”.

Usually this panel is available online, at

http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html

and it appears as shown below (screenshot is in Italian).



3. Interface Description

Library interfaces are exported in the libscct.js file.

3.1 Scct object API

This section contains the definition of data types used and returned by the library. In the rest of the paragraph there is a short description of each one.

AnalogData: represents a set of analog channels data received from the publisher

- **Constructors:**
  function AnalogData(common_data,num_channels,channels)
  You should not directly used this constructor. This object is returned from the getAnalogData API of SCCTChannel Object
- **Attributes**:
  common_data: attribute that contains timestamp and source information
  num_channels: number of analog channels
  channels : array of Channel objects
- **Methods**:
  No methods

DigitalData: object that represents a set of digital lines data received from the publisher

- **Constructor:**
  function DigitalData(common_data,num_lines,lines)
  You should not used directly this constructor. This object is returned from the getDigitalData API of SCCTChannel Object
- **Attributes**:
  common_data: attribute that contains timestamp and source information
  num_lines : number of digital lines
  lines : array of Line objetcs
- **Methods**:
  No methods

CustomXMLData: object that represent a custom xml message received from the server

- **Constructor:**
  function CustomXMLData(common_data,xml_data)
  You should not directly used this constructor. This object is returned from the getCustomXMLData or create into the sendCustomXMLData API of SCCTChannel Object
- **Attributes**:
  common_data: attribute that contains timestamp and source information
  xml_data : string that contains XML text received from the server
- **Methods**:
  No methods

MessageData: object that represents a general message received from/sent to the publisher

- **Constructor:**
  function MessageData(common_data,code,message)
  You should not directly used this constructor. This object is returned from the getMessageData or create into the sendMessageData API of SCCTChannel Object
- **Attributes**:
  common_data: attribute that contains timestamp and source information
  code : integer code associated with the text message
  message : string containing text message
- **Methods**:
  No methods

CommonData: object that is a embedded inside each of previous data types. It contains information of server time (timestamp), its id and its description

- **Constructor:**
  function CommonData(timestamp, source_id, source_description)
  You should not directly used this constructor. This object is part of many library objects and should be used only as attribute
- **Attributes**:
  timestamp : integer containing timestamp of the server
  source_id : integer associated to the data source
  source_description: string containing a description of the data source
- **Methods**:
  No methods

Channel: object that represents a single channel set of samples. An array of Channel instances is contained inside an AnalogData type instance.

- **Constructor:**
  function Channel(num_channel,samples,samples_value)
  You should not directly used this constructor. This object is part of AnalogData Object and should be used only as attribute
- **Attributes**:
  num_channel : integer containing channel index
  samples : integer containing the channel number of samples
  samples_value: array of floating point values
- **Methods**:
  No methods

Line: object that represents the digital value of a single Line. An array Line instances is contained inside a DigitalData object.

- **Constructor:**
  function Line(num_line,line_value)
  You should not directly used this constructor. This object is part of DigitalData Object and should be used only as attribute
- **Attributes**:
  num_line : integer containing the line index
  line_value: boolean value containing the line status
- **Methods**:
  No methods

ConfigurationData: object that represents the general configuration of the publisher

- **Constructor:**
  function ConfigurationData (common_data, release, device, product_type, location, num_channels, num_lines, channels_configuration, lines_configuration)
  You should not directly used this constructor. This object is returned from the getAnalogData API of SCCTChannel Object
- **Attributes**:
  - common_data : attribute that contains timestamp and source information
  - release : string containing server release
  - device : string containing device description
  - product_type : string containing product type
  - location : object containing geographical coordinates of the server
  - num_channels : integer containing the number of analog channels of the server
  - num_lines : integer containing the number of lines of the server
  - channels_configuration: array of ChannelConfiguration objects
  - lines_configuration : array of LineConfiguration objects
- **Methods**:
  No methods

ChannelConfiguration: object that represents the configuration of a single analog channel. An array of ChannelConfiguration istances is cointained in a ConfigurationData object instance.

- **Constructor:**
  function ChannelConfiguration(number,description,sampling_rate,units,min_value,max_value,direction)
  You should not directly used this constructor. This object is part of ConfigurationData and should be used only as attribute
- **Attributes**:
  - number : integer containing the channel index
  - description : string containing the channel description
  - sampling_rate: integer containing the sampling rate value
  - units : string representing units of the channel
  - min_value : minimum value of the channel
  - max_value : maximum value of the channel
  - direction : string containing the channel direction (“Input” o “Output”)
- **Methods**:
  No methods

LineConfiguration: object that represents the configuration of a single digital line. An array of DigitalConfiguration istances is contained in a ConfigurationData type instance.

- **Constructor:**
  function LineConfiguration(number,description,direction)
  You should not directly used this constructor. This object is part of ConfigurationData and should be used only as attribute
- **Attributes**:
  - number : integer containing the line index
  - description : string containing line description
  - direction : string containing the line direction (“Input” o “Output”)
- **Methods**:
  No methods

Location: object that contains geo localization info.

- **Constructor:**
  function Location(common_data,description,latitude,longitude,elev)
  You should not directly used this constructor. This object is returned from server into ConfigurationData object
- **Attributes**:
  common_data : attribute that contains timestamp and source information
  description : string containing the location description
  latitude : floating point latitude value
  longitude : floating point longitude value
  elev : floating point elevation value
- **Methods**:
  No methods

ImageData: object that contains an image file and other info.

- **Constructor:**
  function ImageData(common_data, img_format, img_desc, img_len, byte_img, attributes_vect)
  You should not directly used this constructor. This object is returned from getImageData API
- **Attributes**:
  common_data : attribute that contains timestamp and source information
  img_format : number that specify the format of image:
  0: JPEG
  1: BMP
  2: PNG
  3: TIFF
  img_desc : description of the image
  img_len : image bytes length
  byte_img : array of byte that represents image file
  attributes_vect : array of strings that represents additional attributes
- **Methods**:
  No methods

FileData: object that contains a file and other info.

- **Constructor:**
  function FileData(common_data, nome_file, byte_file, str_md5, attributes_vect)
  You should not directly used this constructor. This object is returned from getFileData API
- **Attributes**:
  common_data : attribute that contains timestamp and source information
  nome_file : string containing full file name (with extension)
  byte_file : array of byte that represents file
  str_md5 : optional md5 hash string calculate on byte_file
  attributes_vect : array of strings that represents additional attributes
- **Methods**:
  No methods

SCCTChannel: object that contains all events and methods to send and receive data from the SCCT server

- **Constructor:**
  function SCCTChannel()

- **Attributes**:
  bConnectedWithFinalServer: boolean value representing the state of connection

- **Methods**:

  function connectToPublisher(ip,port,apikey,timeout)
  This method opens a connection with the SCCT server.
  Return value: no value.
  Events fired: **connectionOpenedHandler**
  connectionClosedHandler
  connectionRefusedHandler
  configurationDataArrivedHandler

  function getConfigurationData()
  Pops the first available ConfigurationData. This method should be used inside the **configurationDataArrivedHandler** event handler, otherwise the queue could be empty and a null value is returned.
  Return value: ConfigurationData object or null

  function getAnalogData()
  Pops the first available AnalogData. This method should be used inside the **analogDataArrivedHandler** event handler, otherwise queue could be empty and a null value is returned.
  Return value: an AnalogData object or null

  function getDigitalData()
  Pops the first available DigitalData. This method should be used inside the **digitalDataArrivedHandler** event handler, otherwise the queue could be empty and a null value is returned.
  Return value: a DigitalData object or null

  function getImageData()
  Pops the first available ImageData. This method should be used inside the **imageDataArrivedHandler** event handler, otherwise the queue could be empty and a null value is returned.
  Return value: a ImageData object or null

  function getFileData()
  Pops the first available FileData. This method should be used inside the **fileDataArrivedHandler** event handler, otherwise the queue could be empty and a null value is returned.
  Return value: a FileData object or null

  function getNewLocationData()
  Pops the first available Location. This method should be used inside the **newLocationDataArrivedHandler** event handler, otherwise the queue could be empty and a null value is returned.
  Return value: a Location object or null

function getCustomXMLData()

Pops the first available CustomXMLData. This method should be used inside the **customXMLDataArrivedHandler** event handler, otherwise the queue could be empty.
Return value: a CustomXMLData object or null

function getMessageData()

Pops the first available MessageData. This method should be used inside the **configurationDataArrivedHandler** event handler, otherwise the queue could be empty.
Return value: a MessageData object or null

function start()

Sends a message to publisher and tells it to start to publish data
Event fired: **streamStartedHandler**

funciton stop()

Sends a message to publisher and tells it to stop to publish data
Event fired: **streamStoppedHandler**

function close()

Close the connection with the publisher

function getAvailableConfigurationDataCount()

Returns the numbers of available ConfigurationData objects

function getAvailableAnalogDataCount()

Returns the numbers of available AnalogData objects

function getAvailableDigitalDataCount()

Returns the numbers of available DigitalData objects

function getAvailableCustomXMLDataCount()

Returns the numbers of available CustomXMLData objects

function getAvailableMessageDataCount()

Returns the numbers of available MessageData objects

function getAvailableImageDataCount()

Returns the numbers of available ImageData objects

function getAvailableFileDataCount()

Returns the numbers of available FileData objects

function getAvailableNewLocationDataCount()

Returns the numbers of available Location objects

function sendMessageData(message,code, source_id, desc)

Send a message to the server

function sendCustomXMLData(xml_data, source_id, desc)

Send a custom xml to the server

function sendImageData(str_format, str_desc, byte_img, attribs, source_id, desc)

Send an image to the server with extra info
str_format: string represents format of image (“jpeg”,”bmp”,”png”,”tiff”)
byte_img : byte of image
attribs : array of strings for optional attributes

function sendFileData(namefile, str_md5, byte_array, attribs, source_id, desc)

Send a file to the server with extra info

namefile : file name
str_md5 : md5 hash string calculate on byte_array (optional)
byte_array : byte of file
attribs : array of strings for optional attributes

function sendNewLocationData(description, latitude, longitude, elev, source_id, source_desc)

Send a newLocation packet to the server

description : Description of new a geographic location
latitude : Floating point value of latitude
longitude : Floating point value of longitude
elev : Floating point value of elevation

- **Events**:

connectionOpenedHandler

This event is fired when a connection is opened

connectionClosedHandler

This event is fired when a connection is closed from the server

connectionRefusedHandler

This event is fired when a connection is refused from the server

streamStartedHandler

This event is fired when the client inform the server to start the stream of data with start method

streamStoppedHandler

This event is fired when the client inform the server to stop the stream of data with stop method

genericDataArrivedHandler

This event is fired when a generic data (AnalogData, DigitalData, CustomXMLData, MessageData, ConfigurationData, Location, ImageData, File Data) is arrived

configurationDataArrivedHandler

This event is fired when a new ConfigurationData is arrived

analogDataArrivedHandler

This event is fired when a new AnalogData is arrived

digitalDataArrivedHandler

This event is fired when a new DigitalData is arrived

customXMLDataArrivedHandler

This event is fired when a new CustomXMLData is arrived

messageDataArrivedHandler

This event is fired when a new MessageData is arrived

imageDataArrivedHandler

This event is fired when a new ImageData is arrived

fileDataArrivedHandler

This event is fired when a new FileData is arrived

newLocationDataArrivedHandler
This event is fired when a new Location is arrived

4. Library usage

The library is provided in a directory called “libscct”. This directory contains the following files:

- *libscct.js* : **it is the only one file that must be included inside the web page**
- *libgeministreammanager* subdirectory: it is an internal core library that permits the management of TCP/IP streaming using JavaScript.
- *libgeminisocket* subdirectory: it is the internal core library that permits the creation of TCP/IP channels using JavaScript.

LibGeminiSocket is an indipendent library that can be used to create any kind of TCP protocols. In this case, it is included inside SCCT following an agreement between GeminiLab(http://www.geminilab.org) e Tools For Smart Minds (http://www.toolsforsmartminds.com). By default, LibGeminiSocket is included in a subdirectory of libscct, and this directory is called “libgeminisocket”. As shown below, it is still possible to change this default settings.

4.1 Import library inside HTML page

To use the library it is necessary to include only one file: *libscct.js*.

This file exports all necessary constants, data types and functions (APIs) needed by the programmer. These APIs are described in details in section 3 of this document.

To include libscct JavaScript library you should add the following script line inside the head tag of the html page:

```
<script type="text/javascript" src="libscct/libscct.js"></script>
```

where **libscct** is the default name of the library directory.

If the library directory containing *libscct.js* and support files is different from the default one, before the library inclusion you must set *libScctPath* global variable for specify the directory name. The path must be specified in the Unix style using “/” separators, even if you are using a Windows operating system.

The path **must not** end with a separator element, as shown in the following example.

```
<script type="text/javascript" >
        var libScctPath=”path/dir_name”;
</script>
<script type="text/javascript" src="path/dir_name/libscct.js"></script>
```

Note that the order is important!!! It is necessary to specify the custom libScctPath before and not after the inclusion of libscct.js file.

As based on *LibGeminiSocket*, libscct needs a server side part able to operate with GeminiSocket technology. To achieve this, GeminiSocket needs a “proxy program” that manages all the traffic between the two end-points. This proxy agent can be present in one of three different solutions:

- **GeminiLab GPlug**: a Windows and Linux plugin (based on .NET or mono APIs)
- **GeminiLab GCloud**: a proxy service available on geminilab.org

Without a GeminiLab proxy agent it shouldn't be possible to reach it via web technologies.

To simplify the installation and developmment**, SCCT publisher is already provided with a built-in GPlug specifically dedicated and optimized for this purpose, so no client actions are needed!!!**
In the remaining part of the document it is assumed that the default GPlug solution is used. If you need specific custom solutions or want to know more about server technologies see the “GeminiSocket User Guide” attached with this guide, or visit http://www.geminilab.org

Altough it is discouraged, it is possible to change the location of the “libgeminisocket” subdirectory. In that case you can move it everywhere in your filesystem, but it is necessary to set the “libGeminiSocketPath” variabile before library inclusion.

```
<script type="text/javascript" >
        var libScctPath=”path/dir_name”;
        var libGeminiSocketPath=”path/dir_name2”;
</script>
<script type="text/javascript" src="path/dir_name/libscct.js"></script>
```

4.2 Event handlers

LibScct is an event driven JavaScript library. This means that the entire program is based on a set of routines called “event handlers”. The programmer has to write his custom routines and bind them to their specific event handler exported by the SCCTChannel object.

As described in section 3, SCCTChannel exposed a set of event handlers. Each of them is fired when a specific event occurs on the communication channel.

Usually, the first action after the creation of a SCCTChannel instance is the definition of necessary event handlers. We can call this section as “Event Handlers Definition”.

```
var scctChannel = new SCCTChannel();

/* EVENT HANDLERS DEFINITION */
scctChannel.connectionOpenedHandler          = function(){...}
scctChannel.connectionRefusedHandler         = function(){...};
scctChannel.connectionClosedHandler          = function(){...}
scctChannel.streamStartedHandler             = function(){...}
scctChannel.streamStoppedHandler             = function(){...}
scctChannel.configurationDataArrivedHandler  = function(){...}
scctChannel.analogDataArrivedHandler         = function(){...}
scctChannel.digitalDataArrivedHandler        = function(){...}
scctChannel.customXMLDataArrivedHandler      = function(){...}
scctChannel.messageDataArrivedHandler        = function(){...}
scctChannel.genericDataArrivedHandler        = function(){...}
scctChannel.imageDataArrivedHandler          = function(){...}
scctChannel.fileDataArrivedHandler           = function(){...}
scctChannel.newLocationDataArrivedHandler    = function(){...}
```

As it shown in the previous code, event handler functions don't have parameters. The reason of this is the goal of simplicity that the library aims to reach. The pattern is based on one or more SCCTChannel declarations (and instantiations) inside global variables, so they can be used anywhere in the program (accessing their methods and attributes).
For example, a typical action that is taken inside a *connectionOpenedHandler* is the start command:

```
var scctChannel = new SCCTChannel();

scctChannel.connectionOpenedHandler   = onOpened;

function onOpened()
{
        scctChannel.start();
}
```

In addition to a typical action associated to a connectionOpenedHandler, the previous example shows that as event handlers it is possible to use also named functions (function with an identificator, in contrast with functions without name as in the classic "eventHandler=function(){}" approach). So it possible to have two distinct sections: "*event handlers declaration*" and "*event handlers definition*".

The programmer usually has only to define the "data-arriving-events" event handlers.
The most general event is the *genericDataArrivedHandler*. This event is rised when any kind of data arrives from the server. This means that an event handler of this type needs to control wich queue is involved. This can be accomplished by using the **getAvailableTYPEDataCount** method or simply trying to pop an element from the queue. If the result is different from null the element can be processed.

```
var scctChannel = new SCCTChannel();
scctChannel. genericDataArrivedHandler   = onGenericDataArrived;

function onGenericDataArrived()
{
        if(scctChannel.getAvailableAnalogDataCount()>0){
                var analogData = scctChannel.getAnalogData();
                //processing
        }
        else if(scctChannel.getAvailableDigitalDataCount()>0){
                //...
        }
        else ; //....
}
```

Altough the previous example is correct, it isn't a pure event-driven approach. In general is better to choose a more specific event handler, dedicated to each data type available. In this way it is possible to write a simpler event handler, because it is necessary to pop only the correspondent queue.

```
var scctChannel = new SCCTChannel();

scctChannel. analogDataArrivedHandler   = onAnalogDataArrived;

function onAnalogDataArrived()
{
        var analogData = scctChannel.getAnalogData();
        if (analogData!=null)
                //elaboration
}
```

Note. Although SCCT for HTML5 is an event-driven oriented library, it is possible to use it without this paradigm. In fact, it is possible to create custom cycles (infinite loops) that continues to poll the queue to see if an available data is present. This approach is heavily inefficient, and it is in contrast with the traditional JavaScript approach. Anyway, the library can be used also in this way.

4.3 Example

```
<!DOCTYPE HTML>
<head>
	<script language='javascript'>
		var libGeminiSocketPath =  "../libgeminisocket";
		var libScctPath		=  "../libscct";
	</script>

	<script src="../libscct/libscct.js"></script>

	<script type='text/javascript'>
		var scctChannel = new SCCTChannel();



		/*************************************************************/
		/*		EVENT HANDLERS REGISTRATION			*/
		/*************************************************************/
		scctChannel.connectionOpenedHandler	= onOpened;
		scctChannel.connectionRefusedHandler	= onRefused;
		scctChannel.connectionClosedHandler	= onClosed;
		scctChannel.digitalDataArrivedHandler	= onDigitalDataArrived;



		/*************************************************************/
		/*		EVENT HANDLERS DEFINITION			*/
		/*************************************************************/

		function onOpened()
		{
			scctChannel.start();
		}

		function onClosed()
		{
			alert("Connection closed from remote host");
		}

		function onRefused()
		{
			alert(scctChannel.reasonOfConnectionFailure);
		}
```

```
function onDigitalDataArrived()
{
	var digitalData = scctChannel.getDigitalData();
	if (digitalData != null){
		if (digitalData.lines[0]==true){
		   document.getElementById("d1").style.background = "#00FF00";
		   document.getElementById("d1").innerHTML = "ON";
		}
		else{
		   document.getElementById("d1").style.background = "#FF4444";
		   document.getElementById("d1").innerHTML = "OFF";
		}
	}
}



/*****************************************************************/
/*                        APPLICATION CODE                       */
/*****************************************************************/

function switchOn()
{
	scctChannel.sendMessageData("switchOn",0);
}


function switchOff()
{
	scctChannel.sendMessageData("switchOff",0);
}


function connect(){
	scctChannel.connectToPublisher(
		document.getElementById('finalipaddress').value,
		document.getElementById('finalport').value,
		document.getElementById('apikey').value,
		 '20.00');
}
```

</script>

</head>

```
<body>
    Server
    <input id='finalipaddress' style='border-radius:10px;font-weight:bold;width:150px;' value=''>
    </input>

     Port
     <input id='finalport' style='border-radius:10px;font-weight:bold;width:30px;' value='8083'>
     </input>

     Api-Key
     <input id='apikey' style='border-radius:10px;font-weight:bold;width:100px;'value= 'OvenDemo'>
     </input>

     <button id="connectButton" style='border-radius:15px;' onclick="connect()">Start</button>
     <button id="onButton" style='border-radius:15px;' onclick="switchOn()">SetOn</button>
     <button id="offButton" style='border-radius:15px;' onclick="switchOff()">SetOff</button>

      <div id="d1" style='border-radius:10px;float:left;margin:0 auto; width:150px; height:20px;
                           background:lightgray;border:solid #777777 1px;text-align:center;
                           font-weight:bold;'>
                         OFF
      </div>
</body>
```