Smartphone & Cross-platform Communication Toolkit User Manual for iOS

Release 3.0.0

July 2013 Edition

Worldwide technical support and product information:

www.toolsforsmartminds.com

TOOLS for SMART MINDS Corporate headquarter

Via Padania, 16 Castel Mella 25030 Brescia (Italy)

CONTENTS

About this Manual

The Smartphone & Cross-platform Communication Toolkit User Manual describes the virtual instruments (VIs) used to communicate and pass data between LabVIEW and either a local or remote application. You should be familiar with the operation of LabVIEW, your computer and your computer operating system.

Conventions

The following conventions appear in this manual:

▸ The ▸ symbol leads you through nested menu items and dialog box options to a final action. The sequence **Tools** ▸ **Options** directs you to pull down the **Tools** menu, select **Options** item.

Bold Bold text denotes items that you must select or click on the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excepts.

`monospace italic`

Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

INTRODUCTION

The Smartphone & Cross-platform Communication Toolkit (SCCT) is a library which sets up a connection among different devices and allows data communication. SCCT can send several types of data: analog data, digital data, files, images, etc. and all of them in a very fast and easy way!

The toolkit offers a set of high level functions for sending data and advanced functions for customized tasks with which SCCT has succeeded in resolving all the common problems regarding data communication.

In this way you can see how the communication results simplified, reliable and quickened!

You can find more information about general features of SCCT and supported platforms in the "SCCT Overview" document, downloadable by

www.toolsforsmartminds.it/products/SCCT.php.

GENERAL WORKING

In this chapter are explained the basis of SCCT working. SCCT is constituted by two complementary libraries: **SCCT subscriber** and **SCCT publisher** that work together during the communication process.

SCCT PUBLISHER

SCCT publisher is a library developed in LabVIEW (to get more details about it see http://toolsforsmartminds.com/products/labview_communication_library.php): it has the role of producer; it can acquire from a remote system several types of data (analog and digital data, xml messages, files, images..) and sends them through the net to all clients who request it.

SCCT SUBSCRIBER

SCCT subscriber, instead of SCCT publisher, is a library that has client role: it receives data being sent by SCCT publisher and provides them neatly to the developer, just ready to be used. In order to receive data and use them, the developer has only to implement the interface provided by the library.

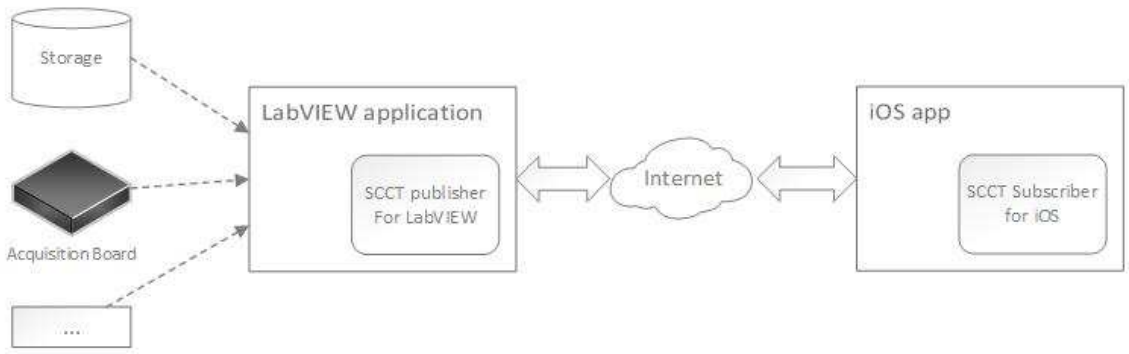


FIGURE 1 - COMMUNICATION SCHEME

REQUIREMENTS

In this chapter we make you a list of the main requirements of SCCT for iOS, each of them has been tested in the indicated versions:

- **XCode 4.x version;**
- **iOS 5 or later version;**
- **One of the next architectures, as:**
  - **armv7;**
  - **armv7s;**
  - **i386;**
- **Internet connection.**

INSTALLATION

First of all you need to have XCode installed on your computer (it is possible download it from Apple Store).

To install SCCT for iOS you have to follows next steps:

1. Open XCode.
2. Start your project.
3. Add the headers and `libSCCT.a` files to your project using the **"Add Files to …"** from **File** menu or drag and drop it in your project.
4. Add to **Other Linker Flags** in your target "Build Settings": `-ObjC -all_load`

You can download all versions of SCCT subscriber and publisher libraries from

www.toolsforsmartminds.it/products/SCCT.php .

In the next chapters it is described how to configure and use this library.

DOCSET INSTALLATION

In the SCCT library package you find also the .docset file. Copy it into the **~/Library/Developer/Shared /Documentation/DocSets/** folder to see the SCCT documentation in the Documentation browser of XCode.

How SCCT subscriber for iOS works

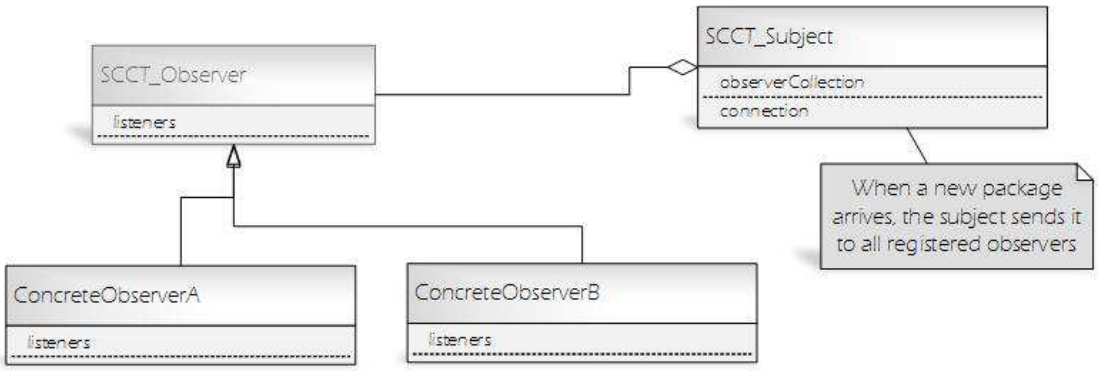
As said in the “General Working” chapter, SCCT is composed of two parts: a publisher and a subscriber. SCCT for iOS works as subscriber during the data exchanging, and has to manage the connection with the publisher.

In order to do this, SCCT for iOS implements the Observer pattern. This is a pattern intuitively used as an architectural base of a lot of event management systems.

This pattern is substantially based on one or more objects, called observers or listeners, that are registered to manage an event that may be generated by the "observed" object, called subject.

In particular, this library provides you with the following elements:

- **`SCCT_Subject`** is a class that has the task of managing in a direct way the communication with the data producer (SCCT publisher), of providing methods that allow the connection with SCCT_Observer and enable the data sending.
- **`SCCT_Observer`** is a protocol that any class can implement. It provides optional methods that are needed to receive all sent data (NB. It starts to receive data only after having been registered to the Subject). You can implement one or more of these methods according to your needs. You can define observers with different roles.
- **`SCCT_Package`** is an abstract class from which other classes inherit. These classes holds different kinds of data, and each type of package is sent in different listeners of `SCCT_Observer` protocol by `SCCT_Subject` (for more details see “Packages” chapter).



HOW DATA PACKAGES ARE MANAGED

`SCCT_Subject` performs the methods of more `SCCT_Observer objects` in the main thread loop. This means that if the invoked Observers methods aren't fast enough, they could stand in a queue in the main thread loop and slow down the main thread performance and therefore the GUI.

If you need to perform long tasks with received data, you are suggested to execute them through a background thread.

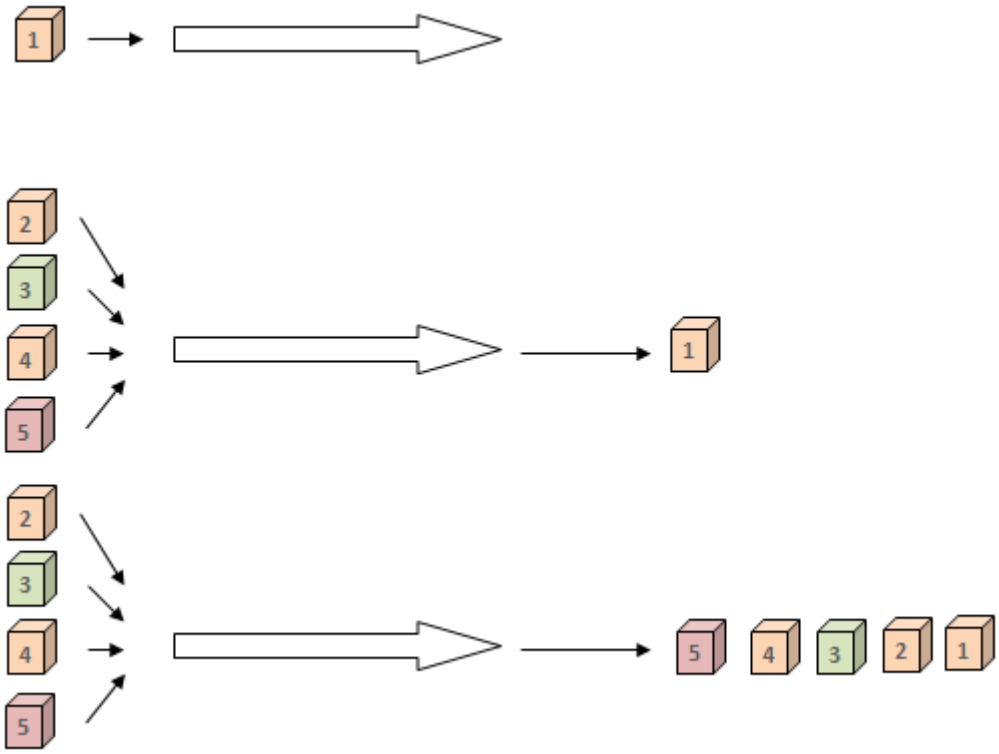


FIGURE 2 - DIAGRAM OF A POSSIBLE PACKAGES QUEUE

The packages are received by `SCCT_Observer` in the same order in which the publisher sent it, so that the developer doesn't need to worry about to sort them. The types of packages and the listeners of observer necessary to receive them are described in the following chapters.

PACKAGES

SCCT manages many types of packages containing data of different kinds. These packages are used to send data from SCCT Publisher to SCCT Subscriber and vice versa in a faster way than sending a few data little by little. These packages are received by `SCCT_Subject` object that sorts them to all registered `SCCT_Observer` objects. `SCCT_Subject` also deals with sending these packages from subscriber to publisher.

In the following chapters are described in detail every type of package with its content and its purpose.

CONFIGURATION PACKAGE

Configurations are contained in the `SCCT_ConfigurationPackage` object that holds information about the used device, like name and type of device, and channels and lines configurations. In particular, a configuration refers to a specific data source and holds information about `SCCT_AnalogDataPackage` and `SCCT_DigitalDataPackage` (see "Analog data package" and "Digital data package" chapters).

Usually this package is sent at the beginning of the transmission to inform the client about how many lines and channels are used and which configuration they have.

CHANNEL CONFIGURATION

`SCCT_ChannelConfiguration` objects are held in the *channels* NSArray property of `SCCT_ConfigurationPackage` object.

`SCCT_ChannelConfiguration` has the following properties:

- **Description**: the name or the description of the channel;
- **Direction**: indicates if it is an input or output channel; this field can have "Input" or "Output" value.
- **Unit**: the measurement unit of the channel;
- **SamplingRate**: the sampling rate of the channel;
- **MinValue**: the minimum value that the channel can assume;
- **MaxValue**: the maximum value that the channel can assume;
- **Index**: the index of the channel.

LINE CONFIGURATION

`SCCT_DigitalLineConfiguration` objects are held in the *digitalLines* NSArray property of `SCCT_ConfigurationPackage` object.

`SCCT_DigitalLineConfiguration` has the following properties:

- **Description**: the name or the description of the line;
- **Direction**: indicates if it is an input or output channel; this field can have "Input" or "Output" value.
- **Index**: the index of the line.

RECEIVING CONFIGURATION PACKAGE

In order to receive the configuration you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)configurationListener:(SCCT_ConfigurationPackage*)configurationPackage
```

ERROR PACKAGE

The errors are generated when a problem occurs and are managed with a `SCCT_ErrorPackage`. Each error is associated with a code (in the `SCCT_ErrorPackage` class there is an enumeration of error codes).

The **error** is a particular type of package because unlike other packages it doesn't hold data but advises that an unexpected event happened. In addictions, this package can be also generated by SCCT for iOS itself if there is any problem in the connection or if some of these parameters are wrong.

Moreover, **whenever an error occurs the connection breaks down,** so this package should be handled with particular care**.**

This error can be generated because of one of these reasons:

Code	Name	Description
1	Wrong API-Key	This error is thrown if API-Key is incorrect.
2	Wrong Timeout	This error is thrown if the timeout is less than or equal to 1.
3	Wrong timestamp	This error is thrown if the client's timestamp is too much different from the producer's one.
4	Expired timeout	This error is thrown if timeout is expired.
5	Lost connection	This error is thrown if connection breaks down before timeout expiring.
6	Transmission error	This error is thrown if it's impossible to send a message.
7	Destination unreachable	This error is thrown if it's impossible to open the connection with the producer.
8	Empty address	This error is thrown if the address string is empty.
9	Wrong port	This error is thrown if the port is less than or equal to 0.

Some of them are generated by the library and have a different timestamp from other packages (that are created by the producer).

RECEIVING ERROR PACKAGE

In order to receive error packages you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)errorListener:(SCCT_ErrorPackage*)errorPackage
```

ANALOG DATA PACKAGE

Analog data are the sampled values of analog channels and are contained in the `SCCT_AnalogDataPackage` object.

These data are stored in a bidimensional matrix which is built by means of a `NSArray` containing in turn other `NSArray` objects of `NSNumber` objects: the first index selects the *channel* and the array of sampled values associated to it; the second index selects the single sampled value and returns a `NSNumber` object.

Sample values are stored in **double** format.

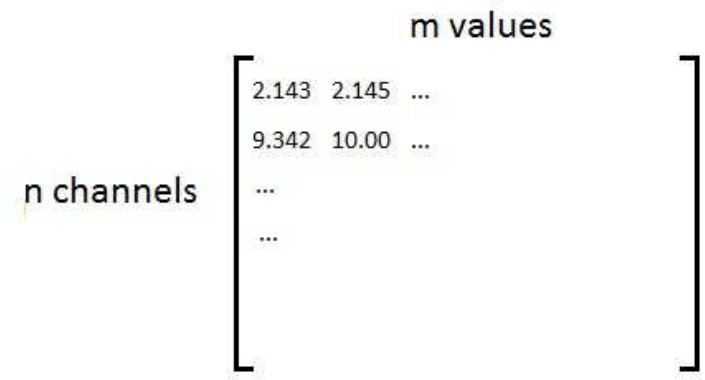


FIGURE 3 - ANALOG DATA MATRIX

You can use these data to show them in a label, draw them in a graph or store them in a database or in a file.

If, for example, you want to get the i-th value of the j-th channel, you have to do as follow:

```
[[[analogDataPackage.values objectAtIndex: j] objectAtIndex: i] doubleValue];
```

RECEIVING ANALOG DATA PACKAGE

In order to receive the analog data packages you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)analogDataListener:(SCCT_AnalogDataPackage*)analogDataPackage
```

DIGITAL DATA PACKAGE

Digital data are values of digital lines and are contained in the `SCCT_DigitalDataPackage` object.

These data are stored in a `NSArray` of `NSNumber` objects. The lines status is represented in **BOOL** format.

If, for example, you want to get the staus of the i-th line, you have to do as follow:

```
[[digitalDataPackage.values objectAtIndex: i] boolValue];
```

RECEIVING DIGITAL DATA PACKAGE

In order to receive digital data packages you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)digitalDataListener:(SCCT_DigitalDataPackage*)digitalDataPackage
```

MESSAGE PACKAGE

A SCCT_MessagePackage object contains a short message with an associate code number. This kind of package is useful for exchange messages or commands with the publisher (e.g. to request some data or to tell it to start a task).

You can instantiate it with the initWithMessage:code: instance method, or with the packageWithMessage:withCode: class method and use the instanced object to send it with the sendMessage: method of SCCT_Subject.

RECEIVING MESSAGE PACKAGE

In order to receive message packages you have to implement the following method of SCCT_Observer protocol:

```
-(void)messageListener:(SCCT_MessagePackage*)messagePackage
```

SENDING MESSAGE PACKAGE

In order to send SCCT_MessagePackage object you have to use the following method of SCCT_Subject object:

```
-(NSUInger)sendMessage:(SCCT_MessagePackage*)package
```

XML PACKAGE

A SCCT_XmlPackage object contains an xml document. This class doesn't validate the xml, so it can be also contained a malformed xml: is a developer responsibility to validate the document if necessary. The document is stored in the *xml* property as a NSString object. This kind of package is similar to message package, but it has a different semantic purpose and then is handled differently from SCCT.

You can instantiate it with the initWithXml: instance method, or with the packageWithXml: class method and use the instanced object to send it with the sendXml: method of SCCT_Subject.

RECEIVING XML PACKAGE

In order to receive xml packages you have to implement the following method of SCCT_Observer protocol:

```
-(void)xmlListener:(SCCT_XmlPackage*)xmlPackage
```

SENDING XML PACKAGE

In order to send SCCT_XmlPackage object you have to use the following method of SCCT_Subject object:

```
-(NSUInteger)sendXml:(SCCT_XmlPackage*)package
```

FILE PACKAGE

SCCT allows you to exchange easily any kind of file between publisher and subscriber through the SCCT_FilePackage class.

This class holds the following data:

- **File name:** The name of the file.
- **File content:** This is a NSData object containing the content of the file received or sent.

- **Md5:** the md5 code of file. This field is optional, so if the publisher sent the file without calculating md5 code, this string is empty. When you send a file you can decide whether calculate it with a flag in the constructor.
- **Attributes:** an NSArray of NSString objects containing a list of file attributes (e.g. the file author name, the creation date etc.). This field is optional, so if the publisher sends the file without it, the array will be empty.

NB: This package is available only in PRO version.

Receiving File Package

In order to receive file packages you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)fileListener:(SCCT_FilePackage*)filePackage
```

Sending File Package

In order to send `SCCT_XmlPackage` object you have to use the following method of SCCT_Subject object:

```
-(NSUInteger)sendFile:(SCCT_FilePackage*)package
```

Image Package

SCCT allows you to exchange easily images between publisher and subscriber through the `SCCT_ImagePackage` class.

This class holds the following data:

- **Description:** The description or the name of the image.
- **Image:** The image received or sent. You can get the image in NSData object representation using the format field to decode it or you can use the UIImage object representation already decoded.
- **Attributes:** an NSArray of NSString objects containing a list of image attributes (e.g. the file author name, the creation date etc.). This field is optional, so if the publisher sends the file without it, the array will be empty.
- **Format:** It's the format of the image. The format can be one of the following values:
  - **kSCCTImageFormatPng;**
  - **kSCCTImageFormatJPeg;**
  - **kSCCTImageFormatBmp;**
  - **kSCCTImageFormatTiff.**

NB: This package is available only in PRO version.

Receiving Image Package

In order to receive image packages you have to implement the following method of `SCCT_Observer` protocol:

```
-(void)imageListener:(SCCT_ImagePackage*)imagePackage
```

Sending Image Package

In order to send `SCCT_ImagePackage` object you have to use the following method of SCCT_Subject object:

```
-(NSUInteger)sendImage:(SCCT_ImagePackage*)package
```

ARRAY PACKAGES

The array packages, that inherit from the SCCT_2DArrayPackage class, are a powerful way to exchange two-dimensional arrays of different primitive types. The supported types are:

Type	**Package**
double	SCCT_2DDoubleArray
float	SCCT_2DFloatArray
integer	SCCT_2DIntegerArray
short	SCCT_2DShortArray
long	SCCT_2DLongArray
boolean	SCCT_2DBoolArray
String	SCCT_2DStringArray

Data are stored in an `NSArray` object accessible by means of `array2D` property. array2S contains other `NSArray` objects containing the single elements of the array. Each element is stored as `NSString` in SCCT_2DStringArray and as `NSNumber` for all other kinds of array.

To understand this concept in detail, the following example shows how to receive a double array and get the element at the 0,0 index as primitive type:

```
-(void)doubleArrayListener:(SCCT_2DDoubleArray*)doubleArray{
     NSNumber * numb = [doubleArray.array2D objectAtIndex:0] objectAtIndex:0];
     double value = [numb doubleValue];
}
```

You can get the dimension of the array with the `rowCount` and `columnCount` properties.

Each array package has also an associated filter id (filterId property) that you can use to see which filter is applied on these data. For more information about filters see **Filters** chapter.

RECEIVING ARRAY PACKAGES

To receive different kinds of array, you have to implements the appropriate listeners (one for each type of array) provided by the `SCCT_Observer` protocol. The following table show you the provided listeners:

Type	**Listner**
double	-(void) doubleArrayListener:(SCCT_2DDoubleArray*)doubleArray
float	-(void) floatArrayListener:(SCCT_2DFloatArray*)floatArray
integer	-(void) integerArrayListener:(SCCT_2DIntegerArray*)integerArray
short	-(void) shortArrayListener:(SCCT_2DShortArray*)shortArray
long	-(void) longArrayListener:(SCCT_2DLongArray*)longArray
boolean	-(void) boolArrayListener:(SCCT_2DBoolArray*)boolArray

String	-(void) stringArrayListener:(SCCT_2DStringArray*)stringArray

Otherwise, you can use the generic listener `arrayListener:` that receive all kinds of array. In this case you have to use the `type` property to discriminate the type of array and use it in the appropriate way.

```
-(void)arrayListener:(SCCT_2DArrayPackage*)arrayPackage{
    switch(arrayPackage.type){

        case kSCCTString:
            //Do something with string array
            break;
        case kSCCTDouble:
            //Do something with double array
            break;
        …
    }
}
```

SENDING ARRAY PACKAGES

In order to send a subclass of `SCCT_2DArrayPackage` object, you have to use the following method of SCCT_Subject object:

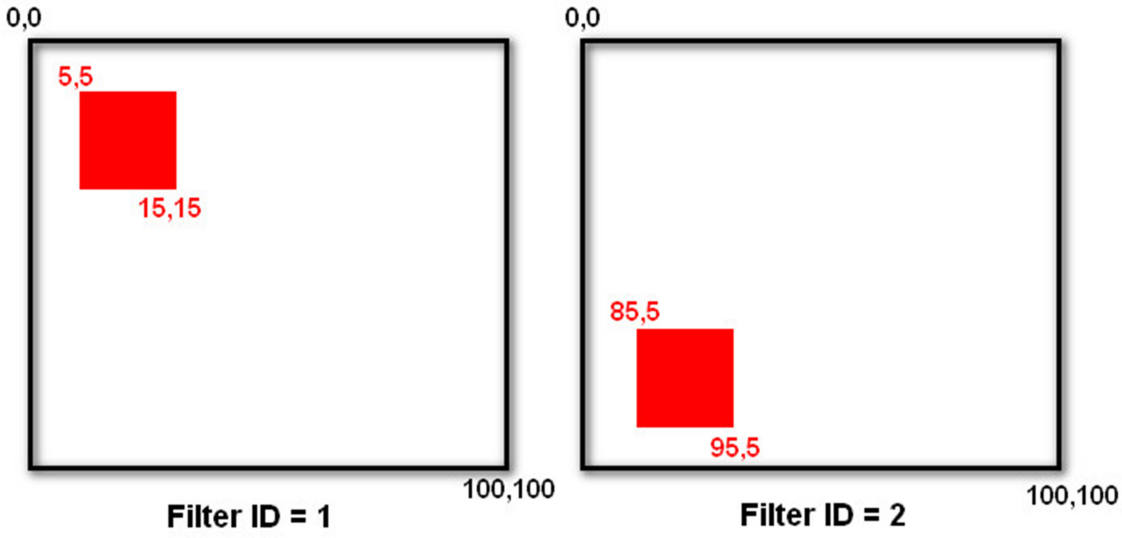
```
-(NSUInteger)sendArray:(SCCT_2DArrayPackage*)package
```

FILTERS

To reduce the amount of bandwidth used to communicate and the cpu and memory utilization on client device, SCCT 3.0 has introduced filter packages. Filters allow subscribers to request a subset of data transmitted by the publisher.

Filters is available for analog data and for 2D array of boolean, String, int, long, short, double, and float. For analog data, filtering allows to select specific channels. Otherwise, for 2D arrays, filtering permits you to select a subset of the available published array data.

Each instantiated filter is identified with a progress id called `filterId`. The `filterId` permits the client to relate every received data package to the relative filter request. `filterId` is fundamental to process received data in the right way. For example, if server manages a 100x100 array and a client needs a 11x11 subarray (from index 5,5 to 15,15), the client sends a filter request. As it is the first filter request sent to the server, the filter request must generate a `filterId = 1`. When the filter request is performed, server sends the required subarray and marks transmitted packages with `filterId = 1`. Later, if the client needs a different 11x11 subarray, it sends a another filter request. As it is the second filter request, the filter request generates a `filterId = 2`. The server performs this new filter request, sends the new subarray and marks transmitted packages with `filterId = 2`. The following figure schematizes this example. The use of `filterId` permits the client to link every received package with the first or the second filter request.



ARRAY FILTER

Filters of array are represented by SCCT_`ArrayFilter` class. In the constructor you have to indicate the array type on which apply the filter.

To select columns and rows you can use `selectColoumnsFrom:to` or `selectRowsFrom:to` methods to select rows and columns. *From* parameter indicates the start value and *to* the final value to select. Otherwise you can directly set up the `query` property using the following syntax: "r1-r2;c1-c2" where r1 is the start row, r2 is the end row, c1 is the start column and c2 is the end column.

The following example explains how to select a subarray of double from index 5,5 to 15,15 .

```
SCCT_ArrayFilter* filter = [[SCCT_ArrayFilter alloc]initWithType:kSCCTDouble];
filter.query = @"5-15;5-15";
/*
```

```
Either way, you can use the following code:
[filter selectRowsFrom: 5  to: 15];
[filter selectColumnsFrom: 5  to: 15];
*/
[subject sendFilter:filter];
```

Analog Data Filter

Analog data filter is represented by `SCCT_AnalogDataFilter` class. In analog data filter you can select the channels (also scattered) and the sources to receive. To select channels and columns you can use the `following methods:`

`-(void)selectChannelsFrom:(NSUInteger)from to:( NSUInteger)from`	Select the channels from *from to to*.
`-(void)selectChannels:(NSArray*)channels`	Select the indexes of channels in the array.
`-(void)selectSources:(NSArray*)sources`	Select the ids of sources in the array.
`-(void)setQuery:(NSString*) query`	The query syntax is: *"c1-c2//s1,s2"* or alternatively *"c3,c4//s1,s2"*. In the first case are selected the channels from c1 to c2. In the second case, instead, the channels c3 and c4 only are selected. In both cases the sources s1 and s2 are selected.

The following example explains how to select the channels from 1 to 3 of the sources 0 and 2.

```
SCCT_AnalogDataFilter * filter = [[SCCT_AnalogDataFilter alloc] init];
filter.query = @"1-3//0,2";
[subject sendPackage:filter];
```

Sending Filter package

In order to send a subclass of `SCCT_Filter` object, you have to use the following method of SCCT_Subject object:

```
-(NSUInteger)sendFilter:(SCCT_Filter*)filter
```

REGISTER AND UNREGISTER OBSERVERS

The first thing to do in order to allow Observer to receive data is registering it to the Subject (multiple registrations are possible if you are handling more than one connection at the same time), so that it makes a request for receiving all the packages that are going to arrive (for which Observer has implemented the method) from then on. Register behaviour is very similar to that of Delegate pattern, often used in Cocoa Touch framework, with the only difference that, with SCCT, it's possible to connect more Observers to the same subject at the same time. It's a good practice to register at least one Observer before opening a new connection, otherwise you may lost some error or important package transmitted at the beginning (i.e. a configuration package). Then, in any case it is possible adding or removing an observer in any moment, also while connection is open. In order to register an Observer you have to call the following method of `SCCT_Subject` object:

```
-(void)registerObserver:(id<SCCT_Observer>)observer
```

In any moment you can decide if an Observer is no more necessary and it doesn't need data anymore. In order to unregister it you have to use the following method of `SCCT_Subject` object:

```
-(void)unregisterObserver: :(id<SCCT_Observer>)observer
```

OPENING AND CLOSING COMMUNICATION

`SCCT_Subject` is the class that deals with the communication: each data passes through it. Each object handles a single connection with a publisher, if you want communicate at the same time with more publishers you have to instantiate more objects.

In order to open the communication with the publisher you have to invoke the `SCCT_Subject` method that follows:

```
-(NSUInteger)openCommunication:(NSString*) address port: (UInt32)port
apikey:(NSString *) apikey description: (NSString*) description
```

This method has the following parameters:

- **Address:** The IP address or the host name of the publisher;
- **Port:** The port number of the publisher;
- **ApiKey:** The password for authenticating the client: if it's wrong, you receive an ErrorPackage;
- **Description:** The name or the description of the client, useful for the publisher to identify which client is connected;
- **Timeout:** The timeout value in seconds. If the connection is lost and the timeout expired the library closes the communication and sends an expired timeout error to observers. This value must be bigger than 1. By default this value is 10;

To close the communication, instead, you have to invoke `closeCommunication` method.

START AND STOP TRANSMISSION

When the communication is open a connection between publisher and subscriber is established, but the publisher waits a client command to begin sending data. The publisher can however send some packages if it considers them particularly important (e.g. a configuration or a message error).

To start data transmission you have to call `start` method, whereas you have to call `stop` method to stop it.

SOURCES

Sometimes data can come from different sources (e.g. the publisher can handle two different devices or take data from different databases). SCCT gives each source an ID and a description.

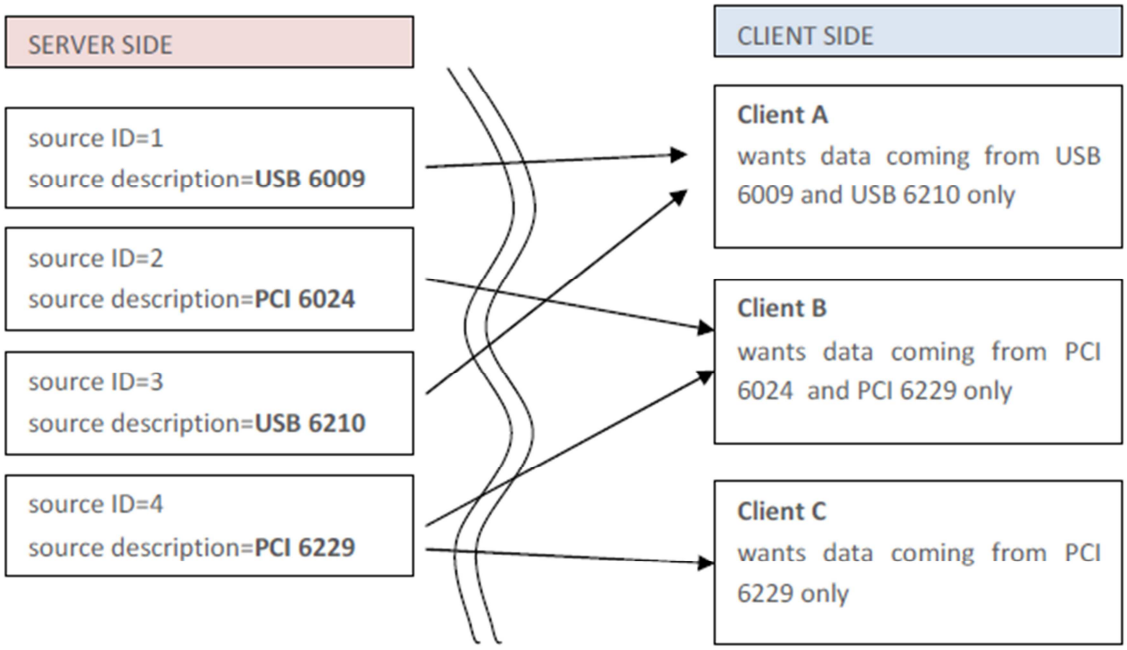


FIGURE 4 - SCENARIO WITH A MULTIPLE DATA SOURCE SERVER AND CLIENTS THAT NEED A SUBSET OF PUBLISHED DATA.

SCCT allows you to operate with different sources in an easy way[1]: if the `sourceFilterEnabled` flag in the `openCommunication` method (see *Opening and closing communication* chapter) is enabled the publisher will send only the packages of selected source list. By default this list is empty and you can modify it with `selectSourcesList:` or `selectSource:` methods. The list contains the ID of selected sources. If you call again this method, the source list selected before is overridden and therefore it is not cumulative with the new one. If, however, `sourceFilterEnabled` flag is false the publisher will send the packages of all sources.

To see which source the package arrives from, you can use `sourceId` and `sourceDescription` properties that are available in every kind of packages.

[1] Function available only in PRO version.

FIGURE INDEX

INDEX