SCCT

Smartphone & Cross-platform
Communication Toolkit

Contents

Introduction

"Give me a smartphone and I will change the world": this seems to be the third millennium saying: laptops, smartphones, tablets and so on have revolutionized the way in which all of us keep in touch with the rest of the world.

Information..

Services..

Contacts..

Work..

Focused in few inches of display!

Impossible not to notice the great potential of such technology in the everyday life!

Due to the high availability of broadband technologies, the possibility of reaching everything you need, wherever it could be, is more and more around us, at home or at work.

Actually, as a matter of fact, mobile technologies are powerfully breaking in the industrial sector because of their **convenience**, **cheapness** and **practicality.**

Unfortunately, conciliating a similar kind of technological new entry with the being-in-use industrial system is not always easy: technology updates faster and faster and technicians have often the challenging task of conciliating the oldest facilities with the most up-to-date solutions, investing time and energies in code adaptation and with no guarantee that the just elaborated solution will work if applied to different devices.

Generally the problem of communication between different machines and devices has always been a great matter for industries just because of a **long series of problems** technicians have to deal with, concerning the complexity of industrial systems, the great difficulty in handling big machines, and several security problems they involve.

This problem has been finally solved: the purpose of this document is show and explain why SCCT is the solution you've been looking for and how it actually works.

Common Usual Problems

In this chapter we describe the most common problems that developers usually have to face when approaching to data communication.

Incompatibility:

It happens often that a great loss of time is caused by the incompatibility of **different devices**. This is a great, very widespread, problem because frequently many off-the-shelf products adopt **proprietary protocols** or communication techniques not compatible with third party systems.

- **Expensiveness:**

Sometimes connecting systems based on old technologies is very expensive. Often the lack of network cards in many old systems requires purchase of **PLCs with serial port** or proprietary interfaces, involving **remarkable costs and time waste**. Expensiveness, in addition, is not only a problem occurring when additional hardware is required, but also **when connecting systems using limited communication models** like the MASTER-SLAVE model. A similar out-of-date technology, in facts, significantly **limits system potential** and **involves adjunctive costs in terms of productivity**.

- **Slowness:**

One of the solutions having been used up to now for devices connection is the Web Server service. Unfortunately, the problem you can frequently find using this technology is that its performances are low, due to the fact that they have many **intermediate phases** and nonessential **overheads slowing down the process** and wasting broadband. In addition, developing a good application with Web Servers is not such an easy work and it takes time and care.

- **Security**:

Handling and sending data implies a **special care and attention for security**, but reaching this goal is not always as simple as it might appear.

Developing an application that controls logins is possible by means of Web Services, but you have to develop your own and spend a long time for it, besides risking eventual bugs and malfunctioning.

SCCT features and benefits

In response to all these problems, T4SM has worked out the most innovative solution for developers who are tired of wasting time and money on avoidable obstacles. SCCT is the newest library developed so that programmers can focus on **WHICH** data are being sent, and **NOT HOW** data can being sent! SCCT meets finally the **best performances**, **reliability** and **simplicity** and takes care of all communication details so you don't have to.

Comparative table: features & benefits.

The following table clarifies that SCCT includes all benefits of different communication technologies. As a remarkable feature, SCCT provides built-in high level data packages, password protected connections and user profiling, so developers can easily define the way data are distributed.

	SCCT	**Web services**	**TCP/IP**	**NSV***
Data optimization to reduce consumed band	**Best**	Medium	Best	Good
Data type	**High level**	Not defined	String	High level
Multiple connections	**Yes**	Yes	No	Yes
Debugging	**Easy**	Complex	Medium	Easy
Integration level	**Complete**	Partial	Complete	Complete
Performance	**Best**	Low	Best	Medium
Complexity	**Easy**	Complex	Medium	Easy
Allow data streaming	**built-in**	No	Built-in	Not built-in
Cross-platform	**Yes**	Yes	Yes	No
White and black list management	**Yes**	No	No	No
User profiling	**Yes**	Limited	No	No
Secure connections	**Yes**	Yes	No	No

*NSV refers to LabVIEW Network Shared Variables

Advantages and improvements

In this chapter we illustrate all the answers T4SM has found out to the more common problems of data communication: SCCT is a solution made by developers for developers, and it is on everyday problems we particularly focused on designing SCCT.

- **Compatibility:**

SCCT is **easily integrable** in any work system, because it doesn't need particular hardware and requires only a computer.
The **great versatility of LabVIEW**, in which SCCT is developed, allows you to connect all those devices that haven't a net interface but use, instead, serial ports or custom boards.

SCCT offers the same interface for every platform you need and is available on the following ones:

- SCCT for LabVIEW
- SCCT for Android
- SCCT for Java
- SCCT for iOS
- SCCT for HTML5
- SCCT for Linux (Ansi-C)

You can find and download SCCT libraries by checking the following link:
http://www.toolsforsmartminds.com/products/SCCT.php

SCCT for LabVIEW supports LabVIEW 2010 or higher and LabVIEW real-time systems like Compact RIO and Single board RIO devices. SCCT for LabVIEW is distributed as ADD-ON and includes both client and server functionalities.

SCCT for HTML5 library is a free library, compliant to RFC 6454.

SCCT for Android, Java and ANSI C libraries are available in two different editions:

- Lite Edition, which includes most common functionalities
- PRO Edition which provides highest performance and advanced features

But why a different type of interface for every different platform?

SCCT for HTML5 works on every platform because it allows you to work through browsers, but it could be less productive than a native application. That's why T4SM has developed for you several types of interfaces for each of your needs, in order to guarantee you the maximum of performances in every moment.

- **Cost-effectiveness:**

SCCT is quite remarkable for **price accessibility of libraries** and for **complete absence of supplementary costs**. In facts SCCT has **no royalties** and, due to its high compatibility with any device, it **doesn't require special hardware**. Every developer knows that *Time is money* and wasting time in modifying code and adapting eventual third party products, is one of the most annoying inconveniences (unfortunately even too much recurring) of developers job: SCCT's easiness and completeness **smooth the way to efficient job removing useless obstacles** and in this way highly **increases productivity** .

- **Fastness and Productivity:**

SCCT is fast in terms both of performances and productivity. Actually in facts is easy and fast to be developed as well it has the best performances in terms of broadband use and fastness in sending data. SCCT **doesn't use http protocols** and doesn't produce http overhead; it uses instead the **PUSH methodology** for data sending, with which it simplify a lot the data passage. Programming a good application will be no longer tedious and slow as before!

- **Security:**

Unlike traditional products, SCCT includes solutions regarding system security and safeness. Functions for **access and traffic volume control**, lists for valid and invalid addresses (**white and black lists**) have been developed in order to simplify the realization of applications that need access monitoring, operation logging and so on. Moreover, SCCT handle **both** the **broadcasting** and **point to point communication**, so that you can choose who is allowed or not to receive your data. Surely it is possible to do it with Web Servers, but you need to develop it by yourself: let SCCT do it for you!

- **Versatility:**

SCCT allows you to manage your data everywhere you are and through every device you're using, thanks to its versatility. T4SM development team has worked to make SCCT able to run on most popular platforms, so that developers don't have to face to all those details of communications among heterogeneous platforms: data formats, data encoding, etc. Thanks to SCCT users community worldwide, SCCT's functionalities are continuously improved, APIs are optimized to get better performance and to support emerging technologies and mobile devices.

Programming details

The next chapters, SCCT's basic features are illustrated, in order to look more into the problem and show how SCCT makes data communication easier and accessible. Step by step, these sections explain main details about functioning, differences between usual methods and SCCT solutions and equip them all with useful consideration and code examples. For precision and completeness sake, each chapter has been dedicated to the explanation of a single type of SCCT library.

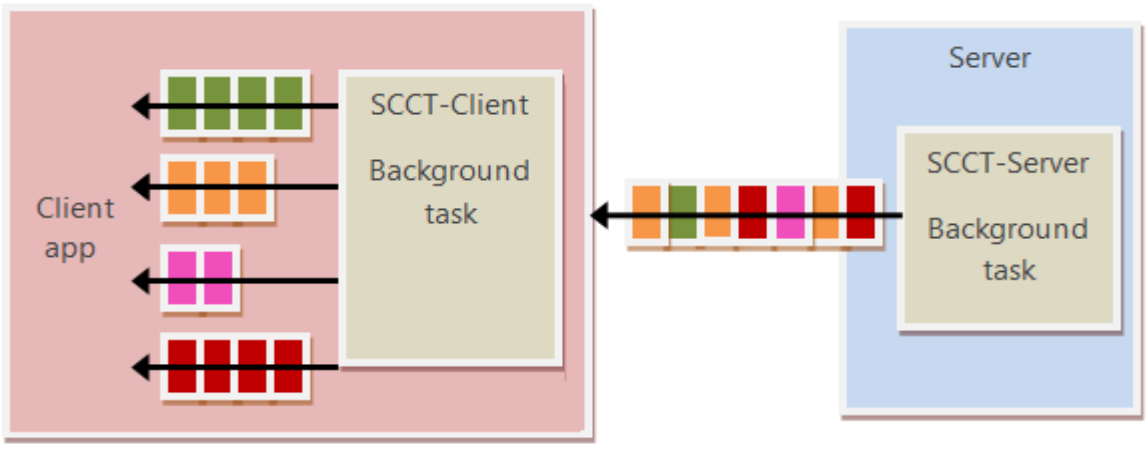
SCCT for LabVIEW

In this paragraph SCCT for LabVIEW developers is briefly illustrated.

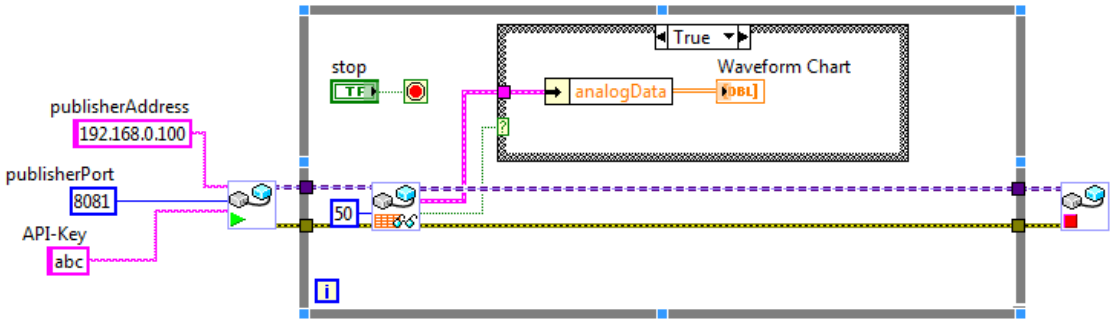
SCCT was created for LabVIEW to help developers and system integrators to exchange data with third party machines, because of LabVIEW open architecture and its capability to manage multiple communication interfaces: serial port, TCP, UDP, etc. Moreover, LabVIEW includes native functionalities to exchange data with third party industrial systems through OPC interface, it supports can exchange data with databases and because of its pervasive diffusion in many industrial fields: automotive, aerospace mechanic and in physics and medical research labs. In the last years, SCCT is grown up to handle communication with emerging technologies related to mobile devices, embedded systems and distributed computing. SCCT has been adopted by as many research centers and Universities to create distributed computing networks for real-time signal signals.

SCCT Client for LabVIEW developers

From a LabVIEW developer point of view, a client connection made by SCCT offers a set of queues created by a background task named SCCT-Client. That background task is in charge to send and receive data packages and manage all connection details so programmer doesn't have to. Data coming from server are pushed in the related queue and client app just has to consume the available data. The figure below illustrates the client task and the SCCT-Client (background) task. Queues are used to transfer data between the two tasks. This architecture allows programmers to create clear code.

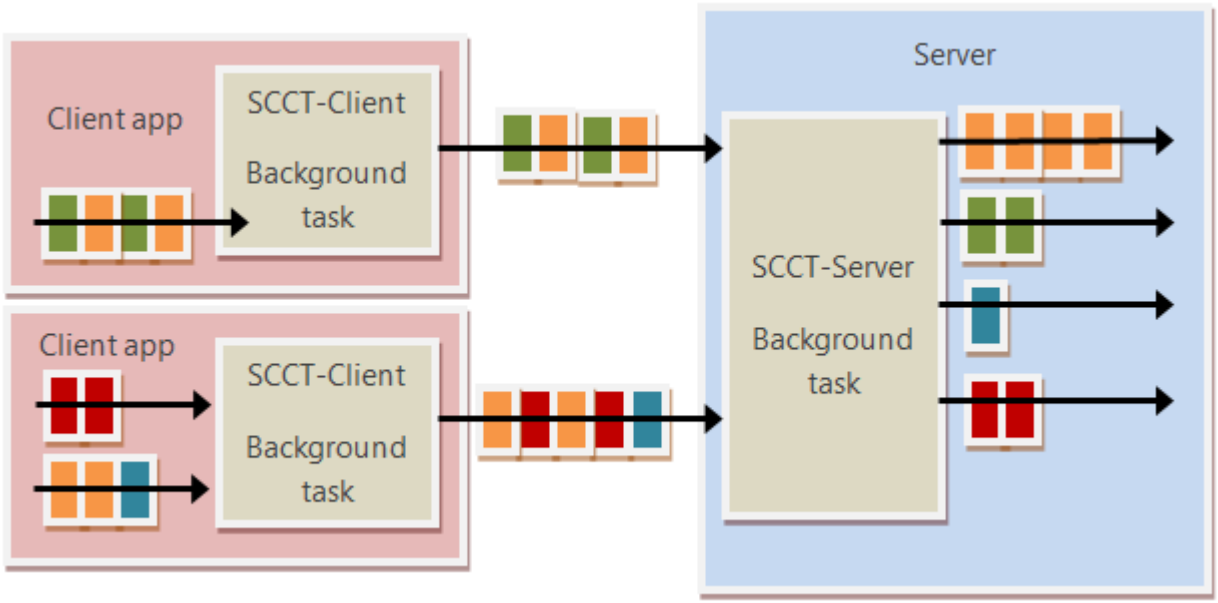


To transmit data to server, SCCT includes polymorphic Vis that greatly simplifies programmer job and reduce development time. The Following example shows the code required to a LabVIEW client that displays analog data streams on a chart.



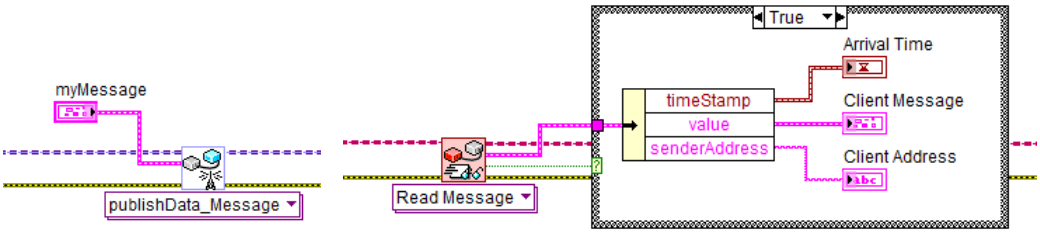
With Three Vis, a client connects to a server, providing the required API-Key, displays received analog data packages, and close connection when task is over. The above example illustrates that SCCT programming style is perfectly integrated in LabVIEW, easy to debug and maintain.

With SCCT, clients can transmit data to server application. From server point of view, data transmitted from clients are gathered into a pipe and then distributed into a set of queues according to send data types, as indicated below:



Data sent by clients are gathered and ordered by SCCT background task at server side. Data packages are organized into dedicated queues with additional information about the clients who sent the package.

This greatly helps LabVIEW developers to log client's requests and, if necessary, to create programmatically a peer to peer communication with one of the active clients. The following figure illustrates the code required to a client to send a message package to a server.



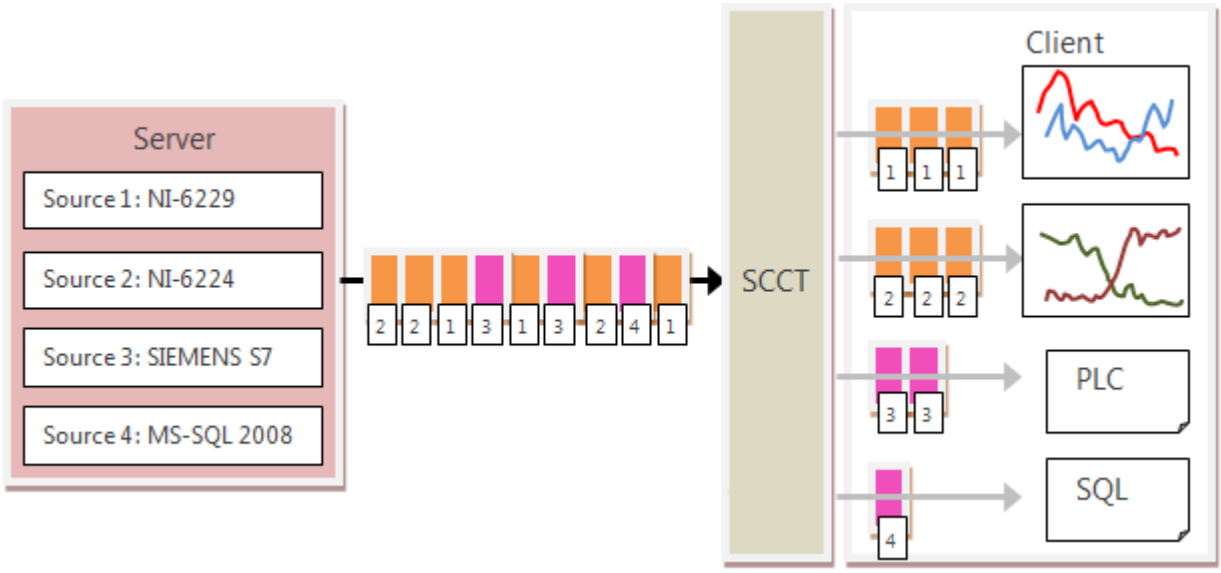
The communication, in the above figure, is the same for all types of packages, so developers can extend their applications with new data types in a straightforward manner.

SCCT features that simplify the software architecture

SCCT APIs include high level features created to reduce development time, to help programmers to create high quality code and to debug their apps in the easiest way. In the following paragraph, some features are briefly described. SCCT development team continuously works to improve SCCT performance, add new features and help developers to save time and create robust high quality code. Refers to SCCT User manual to get more details.

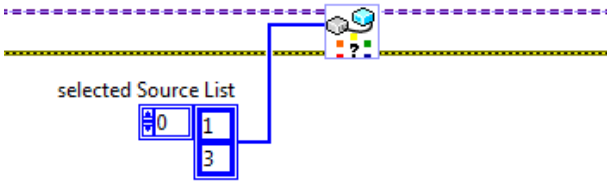
Source selection

Let's consider a server that publishes fresh data continuously to all connected clients. In real life systems data comes from different sources managed by the server. For example consider the case of a server connected to some acquisition devices, database and PLCs. It's important to mark data coming from these sources so that clients can distinguish among packages. SCCT allows data tagging according to their source so clients can properly organize received packages as illustrates in the following figure.



In many situations, clients don't need data from all available sources: to avoid wasting communication band, SCCT provides a powerful feature that filters data at server side.

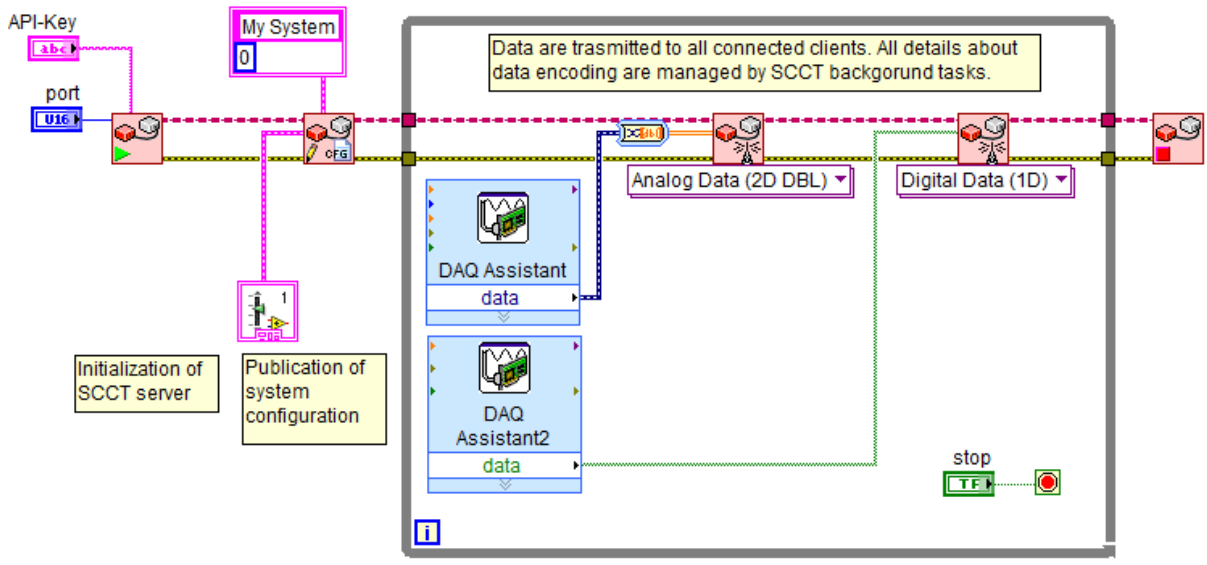
Clients simply inform server that they want data only from a specific subset of sources. SCCT manages the source selection between background tasks. Data filtering is a dynamic procedure that allows client to change their source selection at run-time. The figure below illustrates how much easy is, for a client, the selection of sources.



At server side, programmers don't need to be aware of selections applied by clients: SCCT background tasks manage and filter data packages according to client requests. For this reason "Source Selection" greatly simplifies application development and, at the same time, improves performance of both clients and server and reduces consumed band.

Welcome Kit

Automation systems have a "machine status" which reflects the state of each component of the system. Often there is also a "configuration description" which helps clients to properly use and display received data. For example, a system with an acquisition device has to describe the published analog data (Channel name, Unit of measure, signal range) to allow clients a properly visualization (charts and tables). With SCCT, all data packages, regardless of their types, can be marked as part of Welcome Kit: a set of packages that clients receive whenever they succeed to connect to the server. This feature greatly simplifies the code, as illustrated in the following figure.

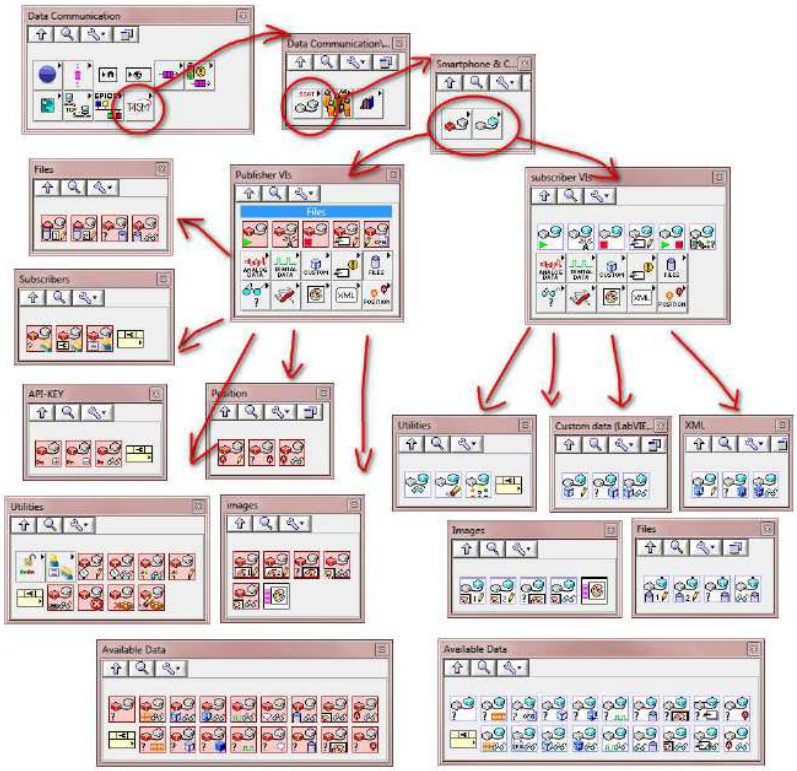


In the above example, server configuration is published when there aren't clients connected! It works because configuration package is marked to be part of Welcome Kit, so when a client will connect, SCCT background task will send the configuration package at the very

beginning of their communication. Welcome Kit can be composed with multiple packages. **With Five SCCT Vis only, a complete application, capable to communicate with a wide heterogeneous systems, is ready to run!**

Fully integration into LabVIEW IDE

SCCT ADD-ON for LabVIEW is completely integrated into LabVIEW IDE, SCCT Vis are accessible from LabVIEW's palette.



Many functionalities are grouped into polymorphic VIs to accelerate their placement into LabVIEW code. Debugging is easy and takes advantage of all debugging features in LabVIEW, like probes and breakpoints. Every SCCT VI is completed with inline detailed help. SCCT has been successfully adopted to create machine to machine (m2m) business solutions, due to the high efficiency of its data encoding and its capability to transfer any type of LabVIEW data.

SCCT fully supports real-times systems like National Instruments CompactRIO and single board RIO products. For this reason SCCT is the best choice for applications where an embedded system has to be monitored and controlled with web-based apps and mobile devices.

Many research centers and business companies have added SCCT for LabVIEW to their software solutions to easily exchange data with other LabVIEW applications and create distributed computing networks.

SCCT for Java and Android

Thanks to the great portability of Java and the wide-spread diffusion of Android in the mobile area, SCCT for Java and SCCT for Android cover a wide range of devices:

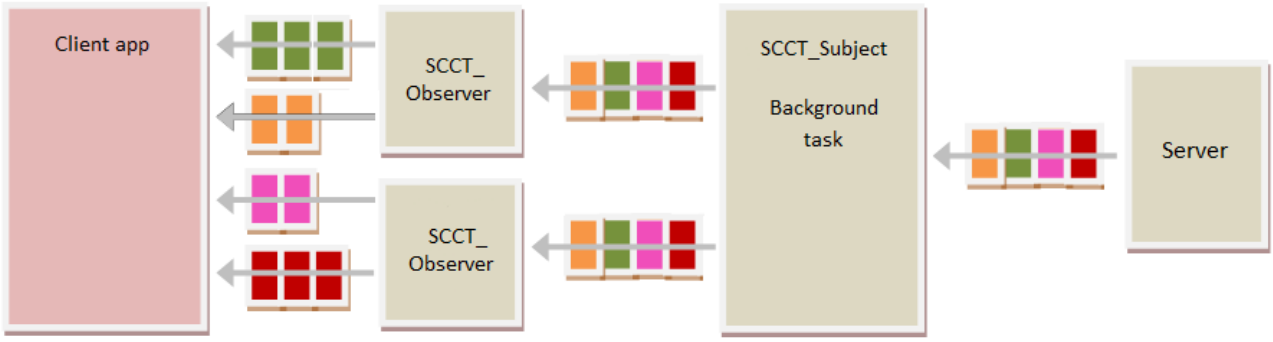
- SCCT for Java is available for all of the main operative systems (as Windows, OS X, Linux, etc.) supporting JVM 5 and higher.
- SCCT for Android is available for any device (smartphones, tablets, etc.) working on Android 2.1 platform and higher.

SCCT Client for Java and Android developers

In this chapter we treat both SCCT for Java and SCCT for Android as a single topic because from developer's point of view they are very similar, although they differ one from the other and they aren't interchangeable (Refer to SCCT User manuals for Java and Android to get more details). SCCT for Java and Android are based on the Event-driven Programming Paradigm (EDP) that provides developers with an easy way to handle data and events generated by server through the overriding of certain listeners. In order to do this, SCCT implements the Observer design pattern and then provides two main classes:

- **SCCT_Subject:** it manages the connection in a background task, sends data to SCCT server and sends the received packages registered SCCT_Observer objects;
- **SCCT_Observer:** it provides abstract listeners that developers have to implement in order to receive data.

Data coming from server are packaged by subject and queued in registered observers. Every observer has its own queue and can implement different listeners in different ways. The figure below illustrates the basic structure and how incoming data are managed:



The following example shows the code required for a Java and Android client to display analog data streams on a chart.

```
public static void main(String[] args) {
        SCCT_Subject subject = new SCCT_Subject(); // Istantiate a SCCT_Subject object
        ConcreteObserver observer = new ConcreteObserver();
        s.registerObserver(observer);
        try {
            subject.openCommunication(address, port, apikey, clientDescription);
            subject.startCommunication();
        } catch (IOException ex) {  /* Manage Error */  }
```

```
        …
}
public class ConcreteObserver extends SCCT_Observer {

    @Override
    public void analogDataListener(AnalogDataPackage package) {
        double[][] data = package.getData();
       /*Use data to display point on a chart*/
      …
    }

    @Override
   public void errorListener(ErrorPackage package){
        /*handle error*/
    }
    … other listener …
}
```

You can also transmit data to server in simple and safe way using some useful methods provided by SCCT_Subject object.

For example, if you want to transmit a message to server you can do as follow:

```
/*The subject object has been already instantiated and the connection has been
opened.*/
int code = 1;
String message = “Hello World”;
subject.sendMessage(message, code);
```

If the message can’t be sent, the sendMessage method returns a false value and an ErrorPackage object is automatically generated and sent to all registered observers in order to be handled.

As illustrated above, SCCT for Java and Android is very easy to use, you don’t have to worry about low level problems or about thread handling: SCCT looks after it for you. This allows you to save lots of time, keep code cleaner and maintainable and get an already tested tool safe to use. For these reasons, SCCT is considered the most powerful and complete communication library today available on the market.

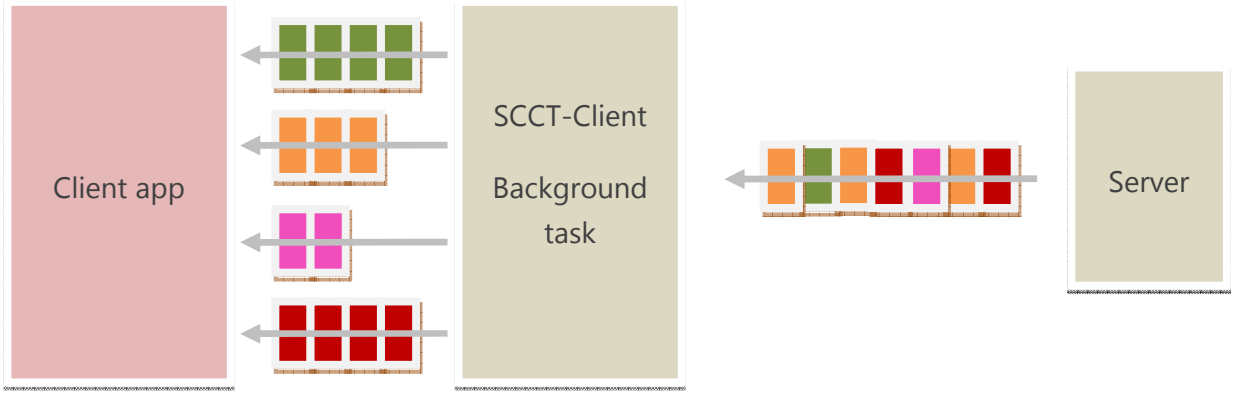
SCCT for HTML5

Thanks to the great diffusion of HTML5 compatible browsers in the mobile area, SCCT for HTML5 can reach any kind of device. Based on the RFC 6455 websockets. You can use SCCT with one of the following browsers:

- Chrome, Chrome for Android, Chromium
- Firefox and Firefox Mobile
- Opera and Opera Mobile
- Safari (version 5 or above) and Safari Mobile
- Internet Explorer (version 9 or above) with the flash plugin

SCCT Client for HTML5 developers

From an HTML5 developer point of view, a client connection made by SCCT offers a set of queues created by a background task named SCCT-Client. That background task is in charge to sends and receives data packages and manage all connection details so programmer doesn't have to. Data coming from server are pushed in the related queue and client app just has to consume the available data using an event driven approach. The figure below illustrates the client task and the SCCT-Client (background) task. Queues are used to transfer data between the two tasks. This architecture allows programmers to create clear code.



The following example shows the code required to an HTML5 client that displays digital data streams on a simple web page and that sends commands to the server (the communication channel is bidirectional).

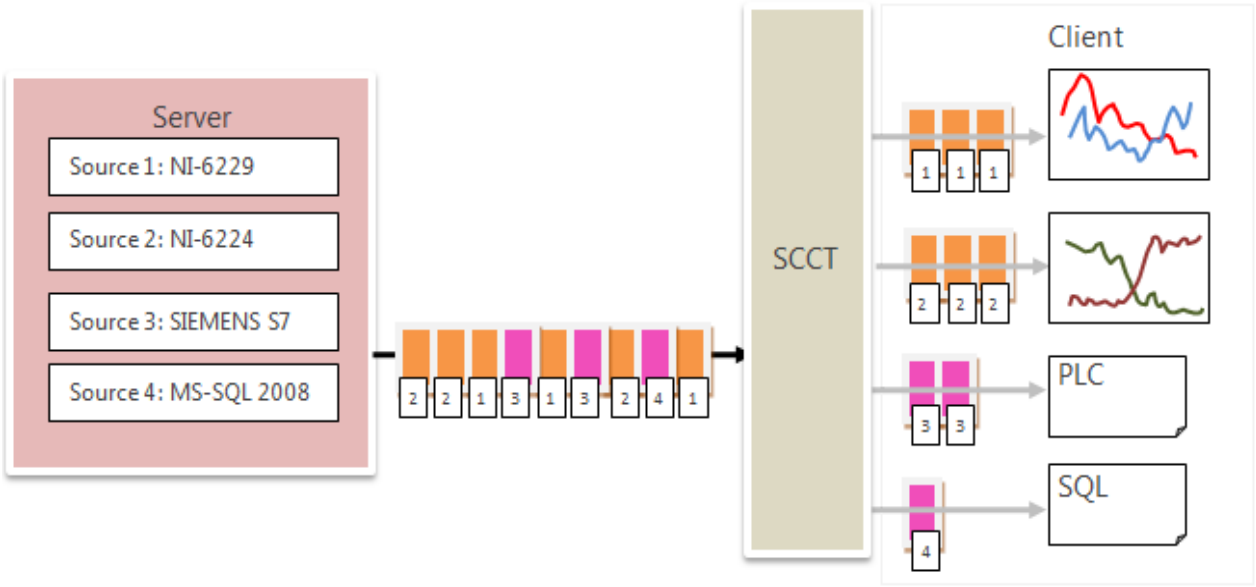
```
<!DOCTYPE HTML>
<head>
    <script language='javascript'>
           var libScctPath             =  "./libscct";
    </script>
    <script src="./libscct/libscct.js"></script>
    <script type='text/javascript'>
           var scctChannel = new SCCTChannel();

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

           /*******EVENT HANDLERS DEFINITION**********/
           function onOpened(){
```

```
                scctChannel.start();
            }
            function onClosed(){
                alert('Connection closed by server');
            }
            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 FUNCTIONS*******/
            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>
```

With the above code, a client connects to a server, providing the required API-Key, displays received digital data packages, and close connection when task is over. The above example illustrates that SCCT programming style is perfectly integrated in HTML5, easy to debug and maintain.

Source selection

Let's consider a server that publishes fresh data continuously to all connected clients. In real life systems data comes from different sources managed by the server. For example consider the case of a server connected to some acquisition devices, database and PLCs. It's important to mark data coming from these sources so that clients can distinguish among packages. SCCT allows data tagging according to their source so clients can properly organize received packages as illustrates in the following figure.



In many situations, clients don't need data from all available sources: to avoid wasting communication band, SCCT provides a powerful feature that filters data at server side. Clients simply inform server that they want data only from a specific subset of sources, while SCCT manages the source selection between background tasks. Data filtering is a dynamic procedure that allows client to change their source selection at run-time. The figure below illustrates how easy is, for a client, the selection of sources.

```
var sources = new Array();
sources[0] = 2; //select source 2
sources[1] = 7; //select source 7
scctChannel.selectSourceList(sources);
```

As illustrated above, SCCT for HTML5 is very easy to use and you don't have to worry about low level and browsers compatibility problems: SCCT takes care of all communication details, so you don't have to. This allows you to save lots of time, keep code cleaner and maintainable and get an already tested tool safe to use. For these reasons, SCCT for HTML5 is considered the most powerful and complete communication library today available on the market.

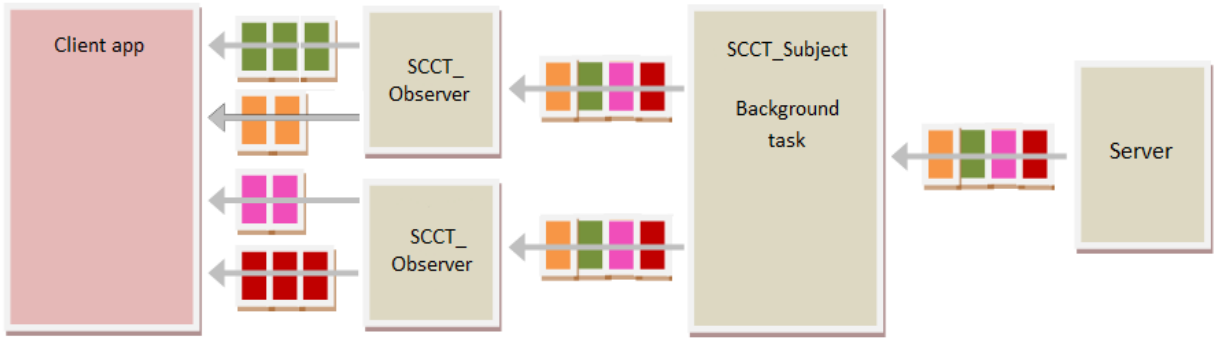
SCCT for iOS

SCCT for iOS is available for all of the most popular Apple mobile devices (iPhone, iPad, iPad mini, iPod touch, etc.) working with iOS 5 or later versions. It has been developed entirely in native code, which allows very high performances and a complete integration with the system. Moreover, it has been tested for guaranteeing a perfect compatibility with Xcode 4.x and higher.

SCCT Client for iOS developers

SCCT for iOS is developed in Objective-C language and it is totally integrated in Cocoa Touch framework: this makes using and integrating it in iPhone and iPad applications easier and faster for developers. It implements the Observer design pattern, allowing to handle data related to events. The Observer pattern is very similar to Delegate pattern, that is often used in Cocoa Touch framework, but it allows to register at the same time more observers to the same data source. Then SCCT provides the *SCCT_Subject* class, that handles the connection in background and has the task of distributing data to registered observers, and the *SCCT_Observer* protocol, that provides optional methods that enable to receive data and events sent by the server.

The figure below illustrates the basic structure and how arriving data are managed:



The following example shows the code required for an iOS client to display analog data streams on a chart.

```
@implementation MainViewController
@synthesize … ;

- (void)viewDidLoad
{
   [super viewDidLoad];
   self.subject = [[SCCT_Subject alloc] init];
   [self.subject registerObserver: self];
   [self.subject openCommunication: self.address  port: self.port  apiKey: self.apiKey
description: self.description];
   [self.subject start];
}


/*SCCT_Observer listeners*
-(void) onConnected:(SCCT_Subject*)subject
{
   //this method is called if connection is established
}
```

```
-(void) onDisconnected:(SCCT_Subject*)subject
{
   //this method is called when connection is closed
}

-(void) analogDataListener:(SCCT_AnalogDataPackage *)analogDataPackage
{
   for(NSArray * channel in analogDataPackage.channels)
        for(NSNumber * value in channel)
        {
            // Use values to add it in your chart
        }
}

-(void) errorListener:(SCCT_ErrorPackage*)errorPackage
{
   //This method is called when an error occurs. You can use errorPackage.description
end errorPackage.code
   //to identify the type of error.
}
…
@end
```

You can also transmit data to server in simple and safe way using some useful methods provided by *SCCT_Subject* object. For example, if you want to transmit a message to server you can do as follow:

```
/*The subject object has been already instantiated and the connection has been
opened.*/
NSInteger code = 1;
NSString * message = @"Hello World";
[self.subject sendMessage:[SCCT_MessagePackage  packageWithMessage:message  code: code]];
```

If the message can't be sent, the *sendMessage* method returns a *False* value and an *SCCT_ErrorPackage* object is automatically generated and sent to all registered observers in order to be handled. SCCT for iOS perfectly integrates with Cocoa Touch and it results very simple for an iOS developer using and including it in his own programs. SCCT for iOS makes you save time while designing, developing and testing applications oriented to data communication.

SCCT for Linux ANSI C

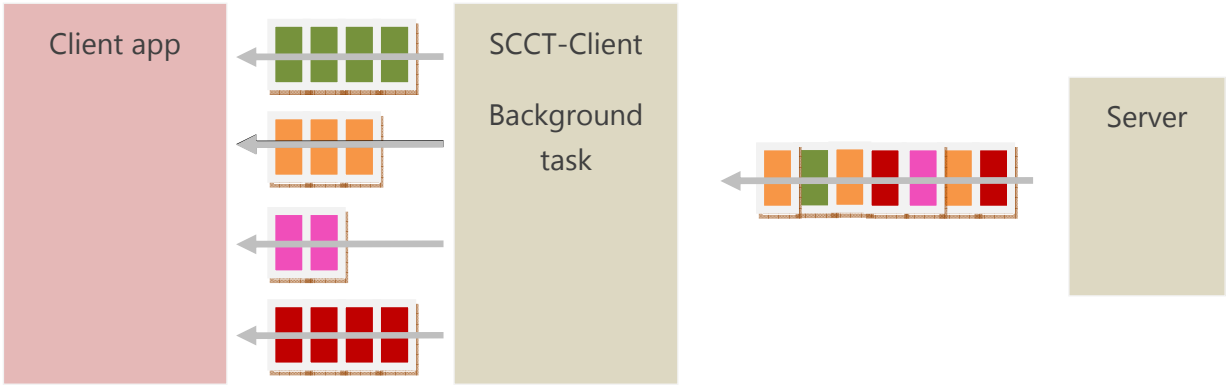
SCCT for Linux ANSI C is available for all of the most popular Linux distributions (Debian, Ubuntu, RedHat, Arch-Linux, Suse, Gentoo), and works naturally with gcc and *make* classic Linux utilities. It is provided in two different binary forms:

- SCCT for Linux ANSI-C for X86 CPU family
- SCCT for Linux ANSI-C for ARM CPU family

SCCT Client for ANSI C developers

From a Linux ANSI C developer point of view, a client connection made by SCCT offers a set of queues created by a background task named SCCT-Client. That background task is in charge to send and receive data packages and manage all connection details so programmer doesn't have to. Data coming from server are pushed in the related queue and client application just has to consume the available data simply calling the appropriate SCCT non-blocking pop function.

The figure below illustrates the client task and the SCCT-Client (background) task. Queues are used to transfer data between the two tasks. This architecture allows programmers to create clear code.



The following example shows the code of a Linux ANSI C client that receives configuration and analog data packets from server.

```
#include <stdio.h>
#include "libscct.h"

int main(int  argc,char **argv)
{
     char server_ip_or_name[512];
     char api_key[512];
     int port;
     int sec_timeout   = 10;

     ConnectResult        *cnn_res            =  NULL;
     ConfigurationData    *configurationData  =  NULL;
     AnalogData           *analogData         =  NULL;
     MessageInfo           msg_info;

     int i = 0;
     int j = 0;
```

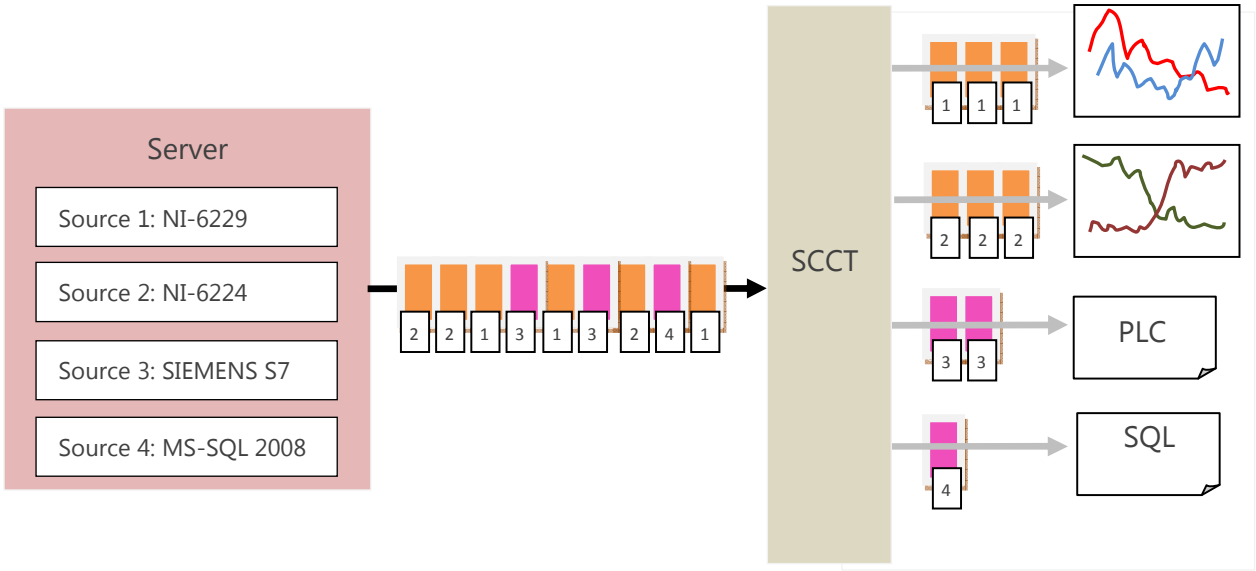
```
	if(argc<4){
		printf("\n\nUse: program.out  ip_address_server port api-key\n");
		return 0;
	}

	/* Prepare parameters for connect  function */
	port = atoi(argv[2]);
	sprintf(server_ip_or_name,"%s",argv[1]);
	sprintf(api_key,"%s",argv[3]);


	/* Connect */
	cnn_res = connectToPublisher(server_ip_or_name,port,api_key,"exampleClient",sec_timeout);

	/* wait for connecting  status */
	do{
	   usleep(5000);
	   msg_info = getStatusConnection(cnn_res->data_connection);
	}while(msg_info.code == CONNECTING);

	if (msg_info.code != CONNECTED){
		printf("\nConnection error");
		return 0;
	}

	if(msg_info.code==CONNECTED){
		printf("\nConnected...");
		msg_info = start(cnn_res->data_connection);
		printf("\nStart executed...”);
		fflush(stdout);

		do{
			usleep(5000);
			msg_info          = getStatusConnection(cnn_res->data_connection);
			configurationData = getConfigurationData(cnn_res->data_connection);
			analogData        = getAnalogData(cnn_res->data_connection);

			if(configurationData!=NULL){
				/*use configuration data*/
				free_configuration_data(configurationData);
				start(cnn_res->data_connection);
			}

			if (analogData != NULL){
				for(i=0;i<analogData->numChannels;i++)
					for(j=0;j<analogData->channels[i].num_channel;j++)
						/*use analog channel*/;
				free_analog_data(analogData);
			}


		}while(msg_info.code == CONNECTED);
	}

	freeDataConnection(cnn_res->data_connection);
	printf("\nProgram exited. %d %s", msg_info.code, msg_info.message);
}
```

With the above code, a client connects to a server providing the required API-Key, displays received configuration data, analog data, digital data, custom xml data, string message data, file and image packages and closes program when disconnected status occurs. The communication channel that SCCT library provides is bidirectional, for this reason it is possible to send some kind of packets to server using specific functions, for example a client sending an image packet to server with this simple piece of code :

```
ImageData *imageDataTmp = NULL;
// Create image packet from image file name
imageDataTmp = createImageDataFromFileName("image.jpg");
// Create two image attributes
appendAttributeOnImageData(imageDataTmp,"Attrib 1");
appendAttributeOnImageData(imageDataTmp,"Attrib 2");
// Send image packet to server
sendImageData(cnn_res->data_connection,imageDataTmp,0,NULL,0);
// free image packet
free_image_data(imageDataTmp);
```

Source selection

Consider the case of a server that continuously publishes fresh data to all connected clients. In real life systems data come from different sources managed by the server. For example consider the case of a server connected to some acquisition devices, database and PLCs. It's important to mark data coming from these sources so that clients can distinguish among packages. SCCT allows data tagging according to their source so clients can properly organize received packages as illustrated in the following figure.



In many situations, clients don't need data from all available sources: to avoid wasting communication band, SCCT provides a powerful feature that filters data at server side. Clients simply inform server that they want data only from a specific subset of sources. SCCT manages the source selection between background tasks. Data filtering is a dynamic procedure that allows client to change their source selection at run-time. The figure below illustrates how easy is, for a client, the selection of sources.

```
// Define an array of two integer
int source_ids[2] ;
// Settings value of id sources to select
source_ids[0] = 1;
source_ids[1] = 5;
// Send a request
selectedSourceList(cnn_res->data_connection,source_ids,sizeof(source_ids));
```

As illustrated above, SCCT for Linux ANSI C is very easy to use and you don't have to worry about low level problems or about thread handling: SCCT looks after them for you. In addition, you can use the same code for X86 and ARM CPU families. This allows you to save lots of time, keep code cleaner and maintainable and get an already tested tool safe to use. For these reasons, SCCT is considered the most powerful and complete communication library today available for Linux systems.

Conclusions

In this last chapter we sum up those SCCT most important features and characteristics you've already got an idea of in the previous parts. The purpose of this section is to provide you with an overview of the reasons why SCCT is the best solution to all the most common data communication problems for its completeness and efficiency:

- SCCT is the most complete solution on the market because it actually enables communication between a great variety of platforms.
- In comparison with other products, SCCT guarantees the best performances in terms of transfer rate, consumed band and CPU utilization. For these reason, SCCT is favorite solution to connect mobile devices to industrial systems.
- SCCT is easily integrable with every application, irrespective of different programming languages.
- SCCT manages all the communication details and allows programmers to focus only on the application they want to realize.
- SCCT for LabVIEW, as regards server side, provides a long series of benefits in terms of simplicity and rapidity and includes powerful functionalities which solve every common problem possible. Such useful tools are fundamental to avoid programmers wasting such a big amount of hours with avoidable details.
- SCCT for LabVIEW allows high performance data sending towards LabVIEW, Java and ANSI-C clients and therefore it is the most suitable tool for the elaboration of distributed computing solutions.

The points enlisted before fully explain the reasons why SCCT libraries are the best choice for handling communications between interconnected systems and for solutions of machine to machine (m2m) data elaboration. SCCT has been chosen and employed by research centers and Universities all around the world and has been chosen by private companies to create their own commercial products and extend their existing systems toward mobile devices.

White Paper Series

Release: 1.2

December 2012

Worldwide technical support and product information:

www.toolsforsmartminds.com

Email: info@toolsforsmartminds.com

TOOLS for SMART MINDS Corporate headquarter

Via Padania, 16 Castel Mella 25030 Brescia (Italy)