RCF User Manual

August 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

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 excerpts.

`monospace italic`

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

GETTING STARTED WITH RCF

In this chapter you are introduced to the main working elements of RCF. In order to completely understand the RCF use and the programming library capability, it is highly recommended to check the basics of object-oriented programming methods (OOP) in LabVIEW.

The next diagram represents a generic LabVIEW application:

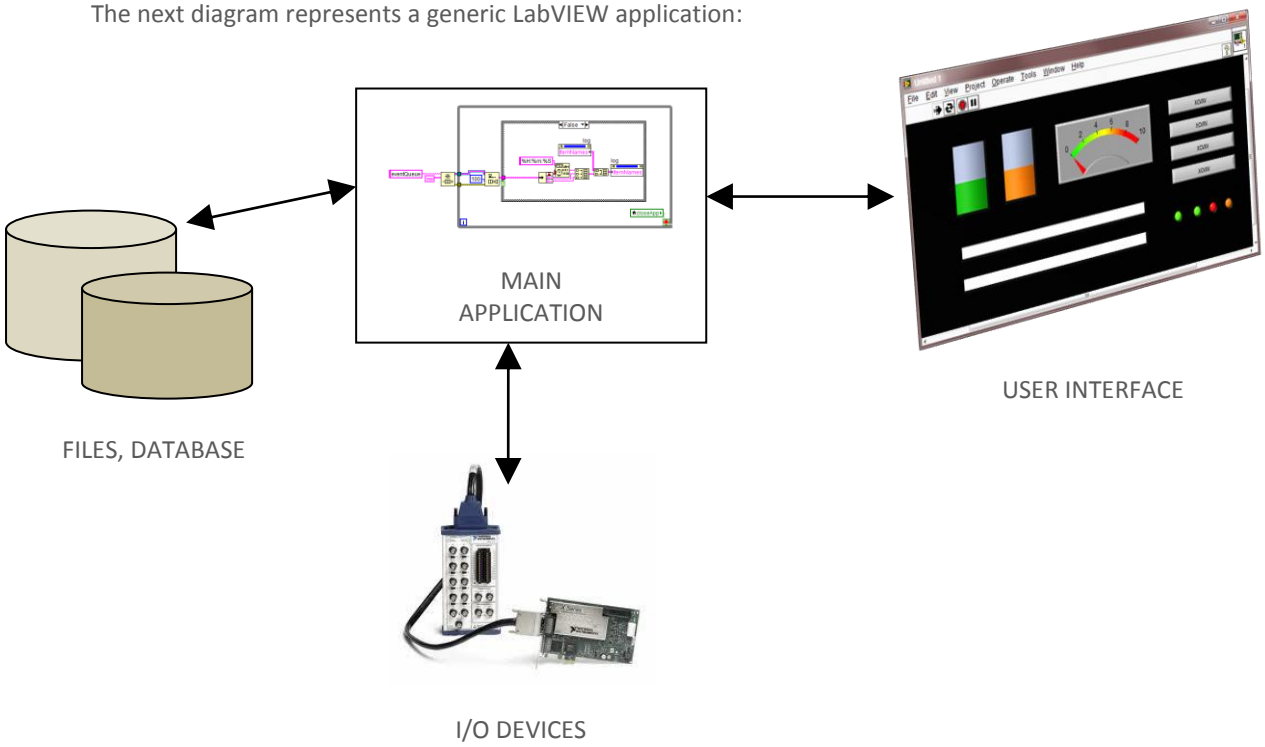


FIGURE 1- GENERIC LABVIEW APPLICATION

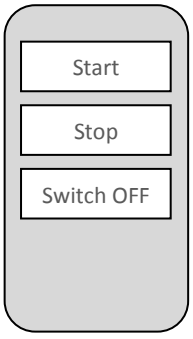
RCF extends your application with a new interface towards mobile devices such smartphones and tablets and towards web browsers. Your system (i.e. your application) maintains its original desktop front-end because RCF runs as a background service responsible to gather requests from connected clients, sends these requests to your application for processing, retrieves results and publishes them to clients. Beside, RCF is responsible to maintain connections, validate requests and notify changes to your system interface. One of the main advantages of RCF over traditional approach as web services is its capability to change system's interface at run-time. Your application is responsible to define its interface as a set of available functionalities called **Commands** in RCF terminology.

Example

A simple data acquisition system may require a simplified interface with three commands:

- Start acquisition
- Stop acquisition
- Switch system OFF

When a client connects to RCF, it receives a description of your system's interface and displays three buttons associated with the commands.



Every command can be parameterized or can be a simple button as the commands you have on a TV remote control.

RCF is integrated in the architecture represented in the previous figure in the following way:

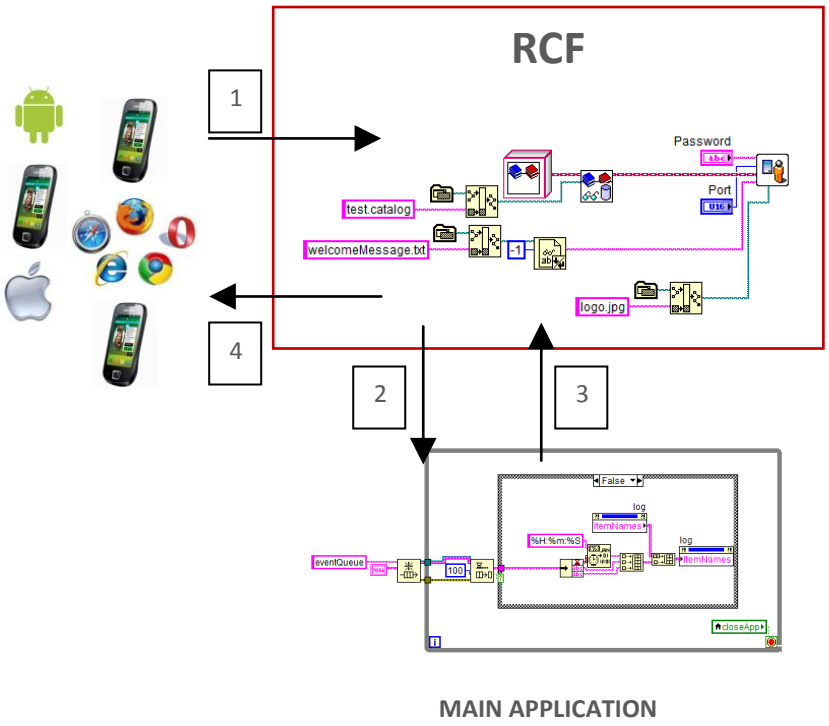


FIGURE 2- RCF INTERACTIONS WITH USERS AND A GENERIC LABVIEW APPLICATION

The above picture illustrates four steps involved in every user request. At the beginning, a user sends a request using one of available commands provided by RCF interface, as indicated by Arrow number 1. RCF validates this request, called **query** in RCF terminology, and then invokes the function associated to that command, as indicated by Arrow number 2. RCF interaction with your application is not limited to a specific protocol or pattern so you can apply the solution you think is best for your application. Arrow 3 indicates that, optionally, the application can produce some results you want to return to the user as response to his request. Arrow number 4 means that RCF results are formatted and returned to client into one of available defined formats, that client's application is capable to display, for example a chart or a table.

RCF manages all communication details in order to allow the programmer to focus on his own application.

REQUIREMENTS

RCF is available as VIP package for LabVIEW 2010 or higher. RCF requires SCCT (Smartphone & Cross-platform Communication Toolkit) available at

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

RCF can be downloaded at

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

REMOTE CONSOLE CLIENT APPS

Remote console clients are available for different platforms and devices. In this paragraph, the platforms and URL where you can find RCF Clients are indicated. At the time this manual has been redacted, third party vendors are making their version of RCF client app and will be available from their web site. Visit official page of RCF project to get more details.

Remote Control Client for windows - developer edition – is available at

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

INSTALLATION

To install VIP packages you need VIP Manager from JKI, you can download a free copy at

www.jki.net

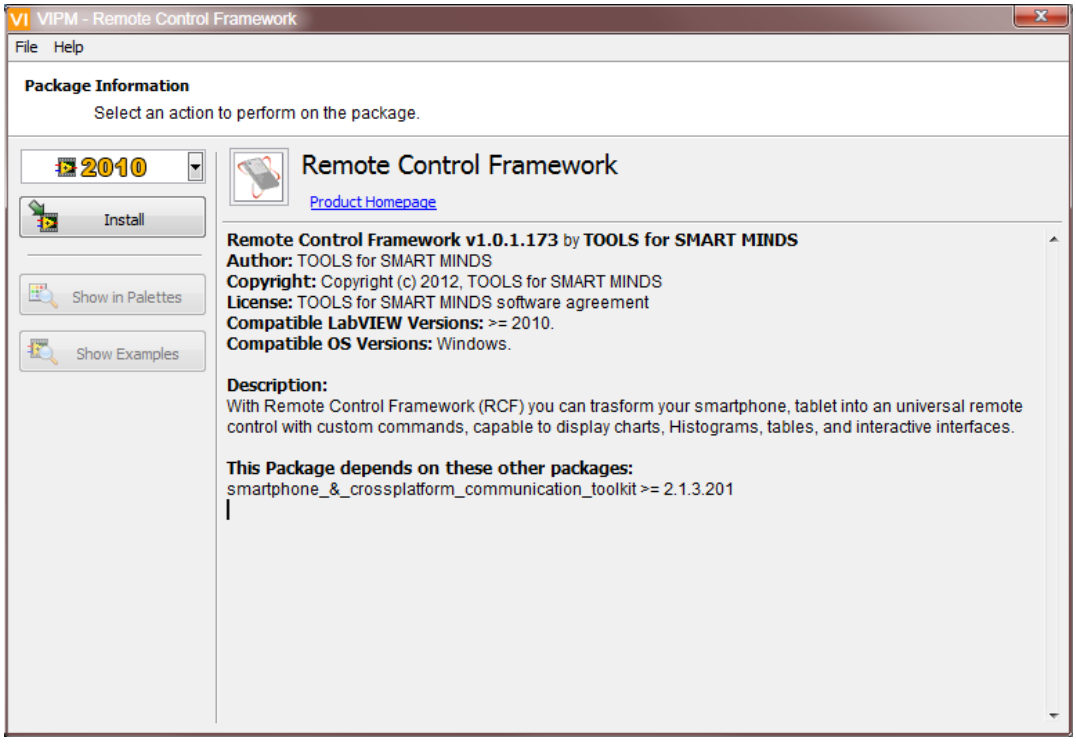


FIGURE 3- RCF PACKAGE INSTALLATION WINDOW.

RCF functions are available in the LabVIEW palette as indicated in the next image:

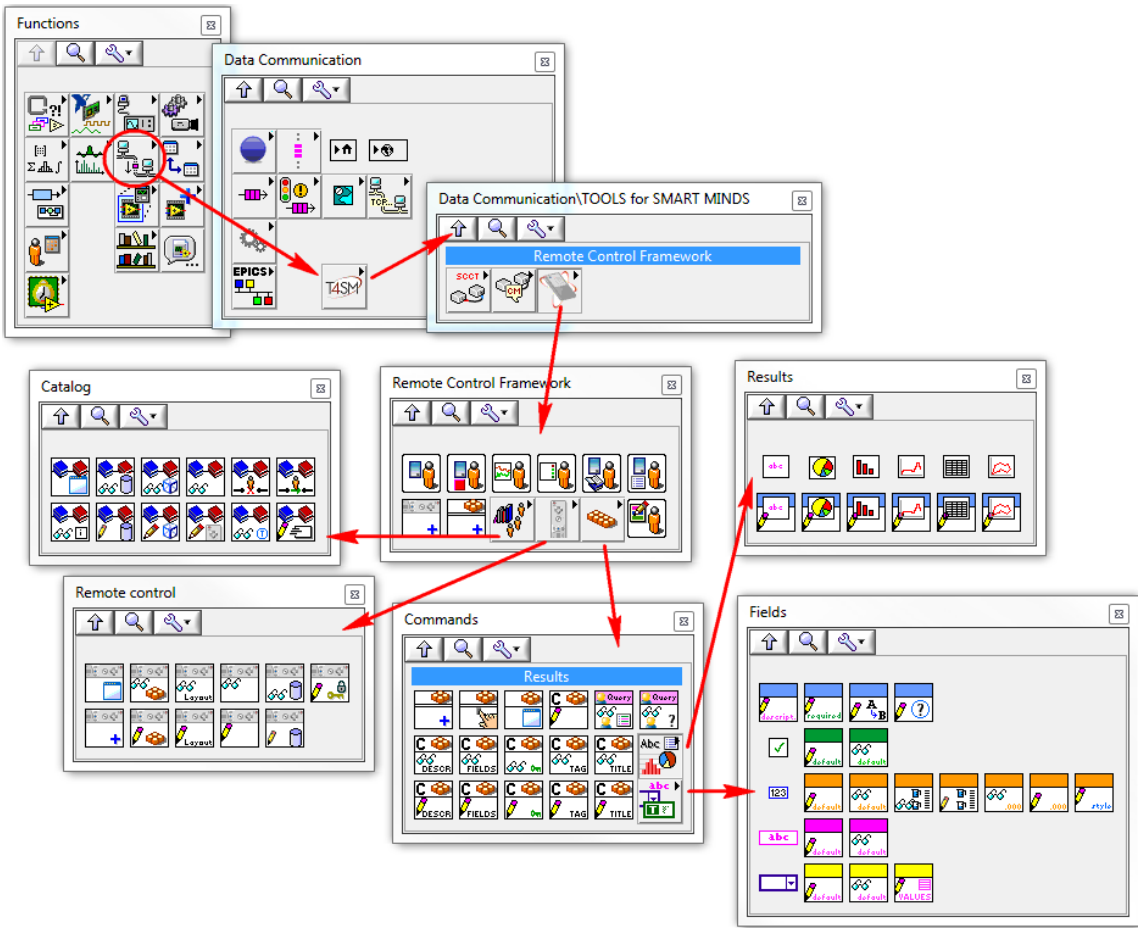


FIGURE 4- RCF PALETTES INTO LABVIEW IDE

Thanks to the control of your application, RCF allows you to build an interface for a system in a simple and safe way.

SYSTEM INTERFACE

The interface is built as a set of "remote controls": in this way you can organize system interface separating groups of commands in coherent sets of functionalities.

For example: if you have a machine producing a certain product, its interface could be constituted by three remote controls:

- **Remote control A:** Commands to view statistics (of production), for management.
- **Remote control B:** Commands to view/edit machine parameters, for technical staff.
- **Remote control C:** Reserved commands, for programmers.

INITIALIZATION

In order to initialize RCF you have to define the following parameters:

- **TCP port:** it is the listening port for clients connections.
- **Password:** it is the password that clients have to use to log in during the connection.
- **Catalog:** it is the system interface that you want to provide. The catalog is constituted by one or more remote controls. In the same way, each catalog is constituted by one or more commands.
- **Welcome message:** it is a message that RCF sends to clients when the connection is opened.
- **Logo path (optional):** it is the absolute path of the logo which has to be displayed on the clients next to the welcome message.

A catalog can be either built programmatically in the applications or loaded from a file. In the following example is showed how to use RCF with a catalog loaded from a file:

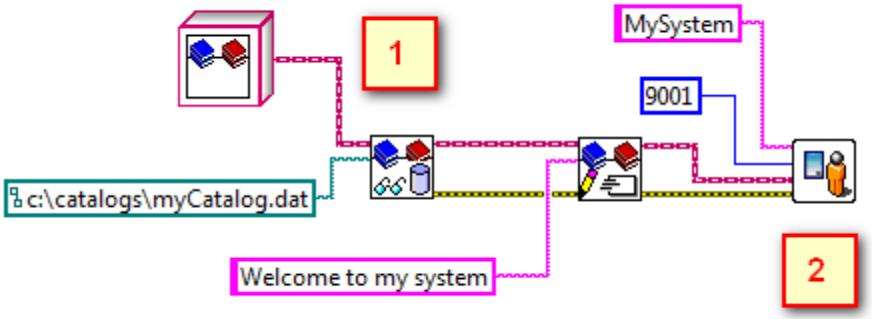


FIGURE 5- LOADING A CATALOG FORM FILE [1] AND LAUNCHING RCF [2]

When clients connect to a system, RCF sends a welcome message:

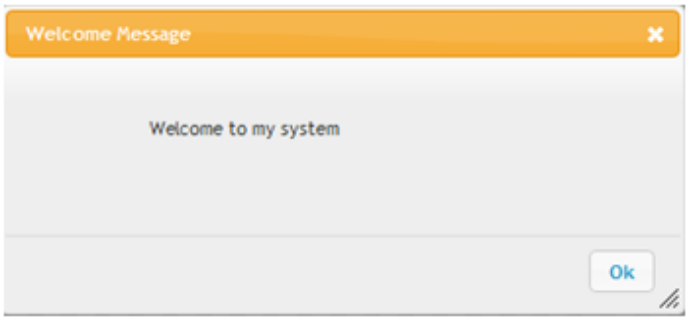


FIGURE 6- WELCOME MESSAGE DISPLAYED ON A RCF CLIENT FOR WEB BROWSER.

RCF allows you to use only one catalog at a time. During the execution of your application you can:

- Modify the catalog by adding or removing remote controls.
- Modify the remote controls by adding or removing single commands.

CREATING A CATALOG

In order to create a catalog into a LabVIEW program, you have to define an array composed of one or more remote controls. RCF does not manage empty catalogues, i.e. catalogues which don't include remote controls.

In the following figure is shown a catalog constituted by a remote control only:

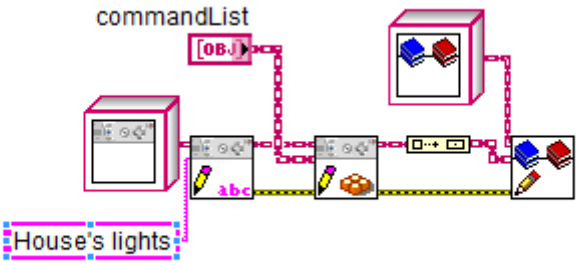


FIGURE 7- A CATALOG, COMPOSED BY A SINGLE REMOTE CONTROL NAMED "HOUSE'S LIGHTS".

In this figure the catalog is saved in a file through the `SaveCatalog.vi` function: you can also use an already existing catalog by uploading it from a file through the `LoadCatalog.vi` function. You can create a catalog using the `EditCatalog.vi` that you find in the LabVIEW palette pressing **RCF ▸ Utilities.** With this utility you can define all of the catalog functional details **at run-time.**

RUNNING RCF

RCF has to be executed in your main application VI. RCF creates additional threads to manage communication with clients. Only one instance of **RCF-Manager.vi** can be executed into your application.

STOPPING RCF

RCF runs in a separate loop from main application so you need to notify to RCF when application is terminated so that RCF itself stops execution. To achieve this result, use **stopManager.vi** as shown in the figure below.

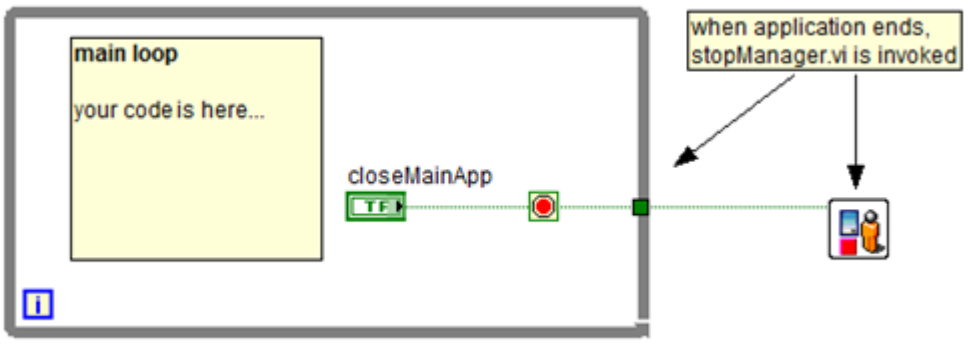


FIGURE 8- STOPMANAGR.VI IS REQUIRED TO END RCF TASK.

CREATING A REMOTE CONTROL

A remote control is composed of the following elements:

- **Description:** It is the name of the remote control displayed by client's terminal.
- **ID:** It is a univocal number used by the Catalog displayed by client's terminal.
- **Commands:** It is an array of objects created for executing functions associated to each remote control button.

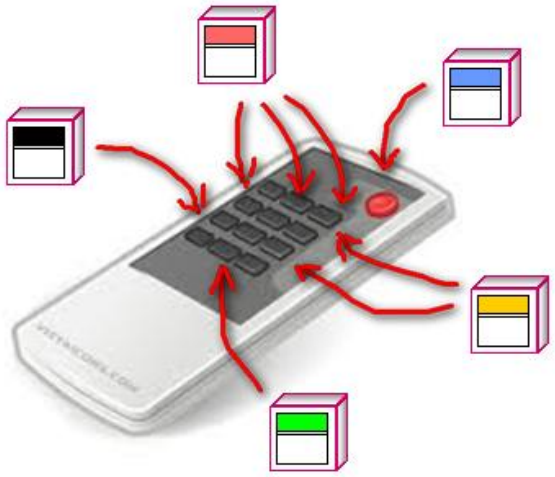


FIGURE 9- COMMANDS ARE ASSOCIATED TO REMOTE CONTROL BUTTONS.

The previous figure illustrates the case some commands, with red and yellow box, are associated to multiple buttons on your remote control. This is useful when a single command can execute different actions according to its input parameters, so you don't need to create a separate command for every single action. For instance, if you want to manage a light switch, instead of creating two commands called respectively SwitchLightOn and SwitchLightOff, you can create a single command LightSwitch with a boolean input parameter, called State, indicating if light must be On or Off. If you have multiple lights you can add a second parameter indicating which light has to be switched. then you associate the command to different buttons with different titles: (Garden light ON, Garden Light Off, etc.).

In the image below is illustrated the necessary code for creating a remote control denominated "House lights".

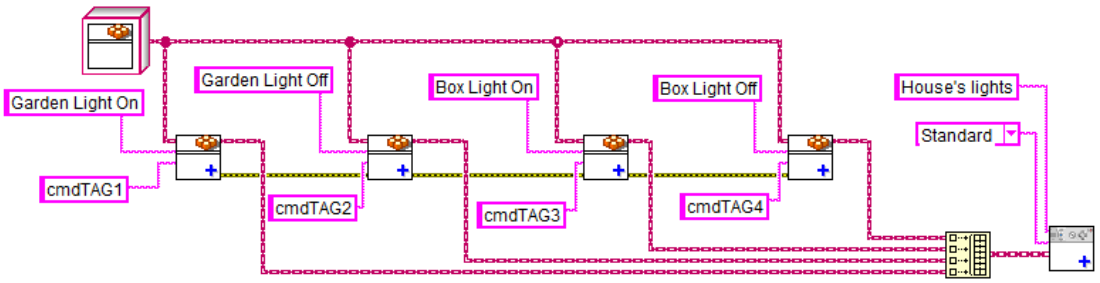


FIGURE 10- A SIMPLE REMOTE CONTROL WITH FOUR COMMANDS.

In this example the remote control is made up of four commands titled *Garden Light On*, *Garden Light Off, Box Light On, Box Light Off*. These four simple commands have no parameters. You will learn in the following chapters that each command can implement a simple function or manage complex activities according to the user's input and to the current status of the application.

You can select the display layout of the remote control on the client device, through the **Layout** parameter. Two types of layout are available:

- **Standard layout.** This layout shows buttons and results in the same window. An example of this layout is showed in the following figure:

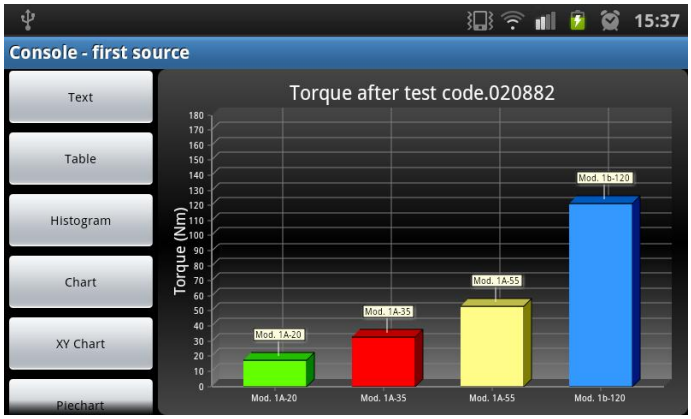


FIGURE 11- CLIENT WITH "STANDARD" INTERFACE, FOR ANDROID OS

- **Remote control layout:** This remote control displays commands as a button list. The following figure shows Remote Control Layout on RCF client for Android.

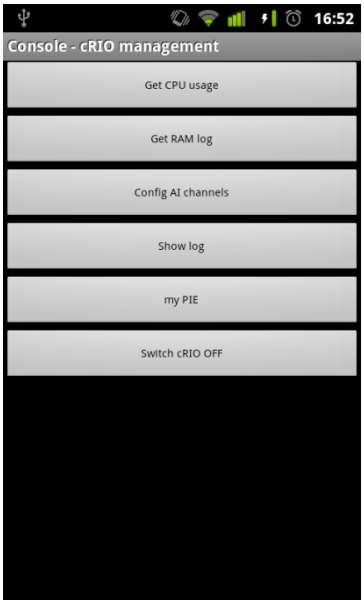


FIGURE 12- CLIENT WITH "REMOTE CONTROL" INTERFACE, FOR ANDROID OS

According to the different supported platforms there may be variations in the layout definition.

CREATING A COMMAND

Commands have to be created through the extension of the `command` class predefined in the RCF library. The `command` class can't be modified in order to guarantee the code portability.

In order to create a new command, follow the following steps:

Create a new class from File >> New as shown below:

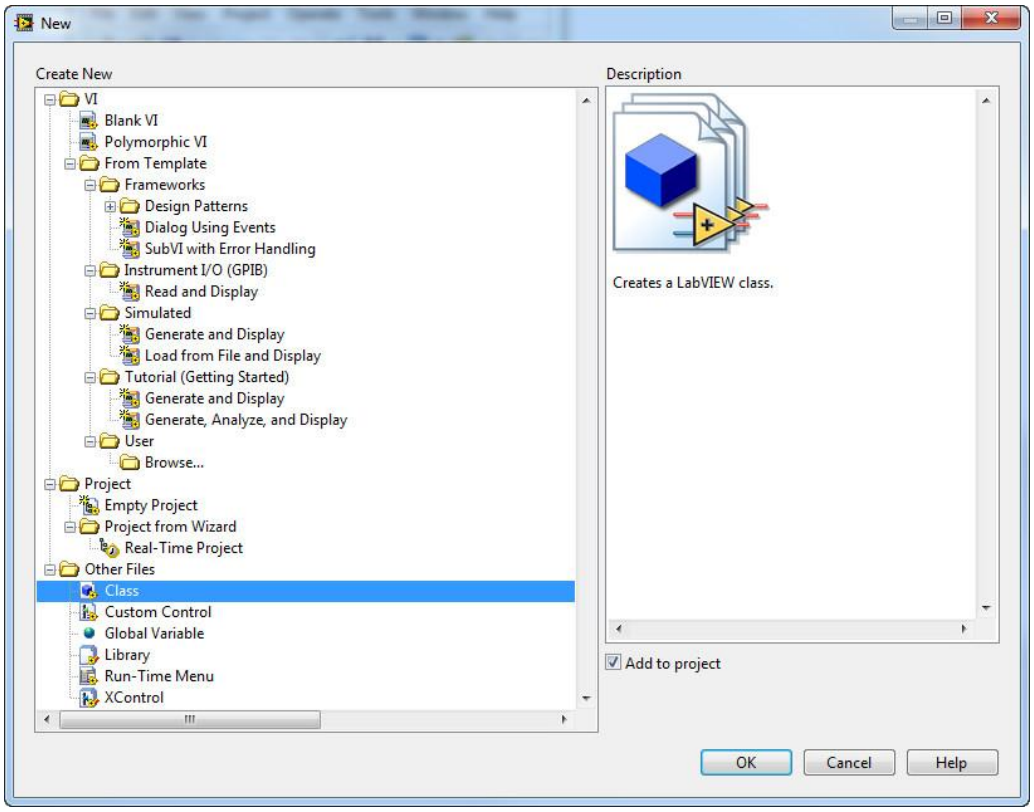


FIGURE 13- CREATING A NEW CLASS IN LABVIEW.

Assign a name to the new class, as shown below:

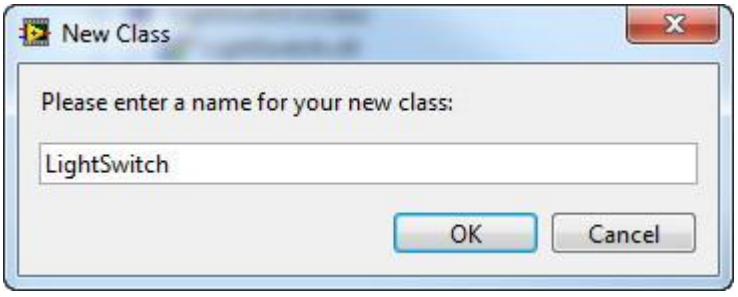


FIGURE 14- SPECIFY THE NAME OF THE NEW CLASS.

Open class property window and modify class inheritance so that it inherits from the `Command` class, as indicated in the figure below.

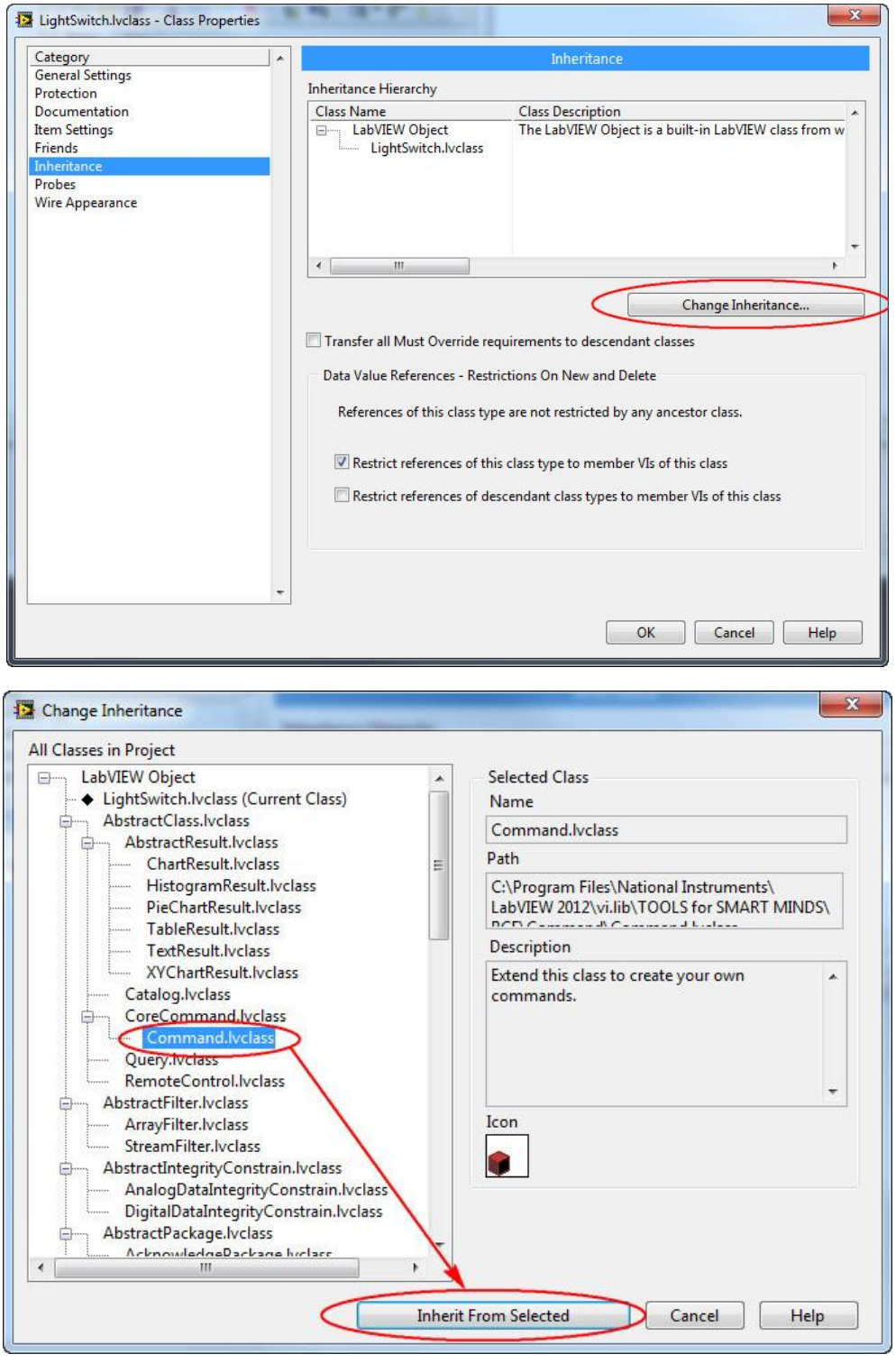


FIGURE 15- INHERIT YOUR COMMANDS FROM COMMAND CLASS.

In the created class you can personalize class attributes and create methods necessary for handling these attributes.

RCF requires only one method of the extended class which has to be implemented for each new command: the **execute** method. Override this method and save it with name execute.vi as shown below.

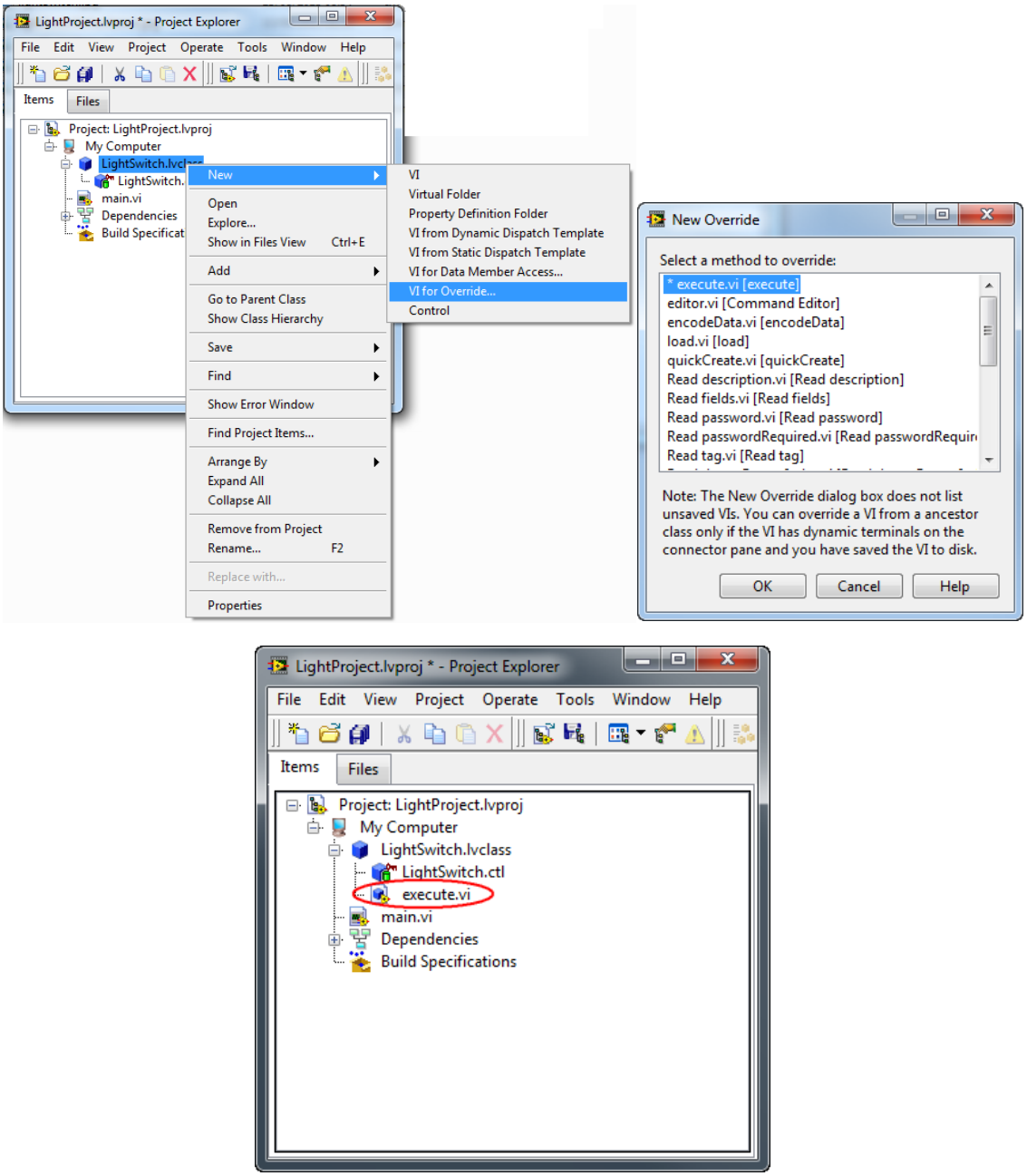


FIGURE 16- COMMAND CLASS MUST OVERRIDE EXECUTE.VI METHOD.

IMPORTANT: Every command can be used if and only if it implements the **execute** method.

The following figure illustrates the default block diagram of execute.vi method. By default, LabVIEW adds a call to ancestor class method.

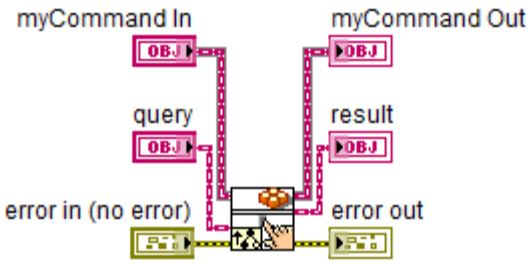


FIGURE 17- DEFAULT BLOCK DIAGRAM, CREATED BY LABVIEW WHEN YOU OVERRIDE EXECUTE.VI METHOD.

Remove call to ancestor method and create a structure as follow:

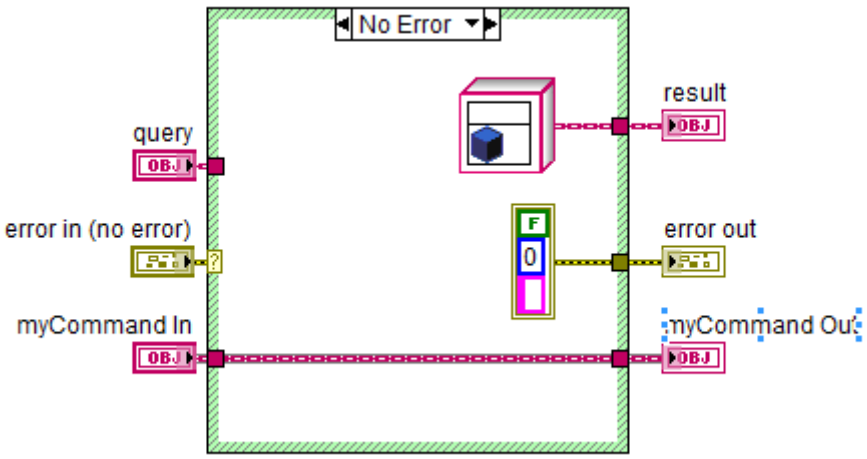


FIGURE 18- EDITED BLOCK DIAGRAM WITH CASE STRUCTURE TO HANDLE ERROR IN.

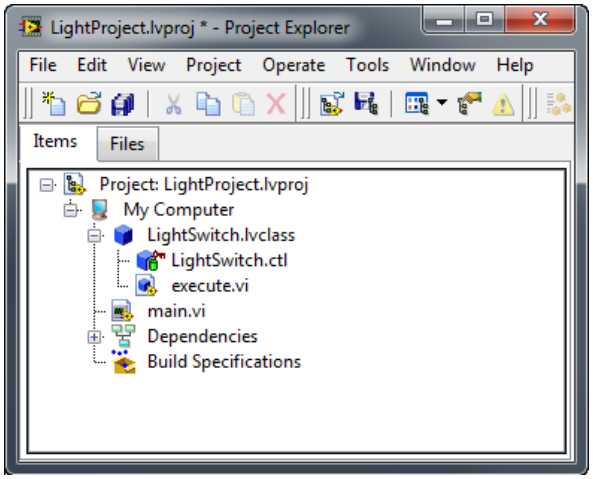
At run-time you can get information about:

- User who made the request;
- Additional field values typed by user to manage the request;
- Additional fields of the current command.

Add the code required to execute the command in the case indicated in the previous picture. Notice that the above command returns an **AbstractResult** constant value which means that the client doesn't receive any notification that command has been executed. With RCF, your commands can return one or more results to client who made the request. The following paragraph illustrates the different result types you can send to clients.

Placing a LabVIEW object on the block diagram

After you create a class, drag the class name or the associated .ctl item on the diagram and LabVIEW creates an instance of selected class.



Returning a result after command execution

Result types

Commands can return to user one or more of the results indicated in this paragraph. Keep in mind that not every client's interface can display multiple results at the same time. Available result types are:

- Text
- Chart
- Histogram
- Pie chart
- XY Chart
- Table

Results can be placed on block diagram from the result subpalette as indicated in the figure below:

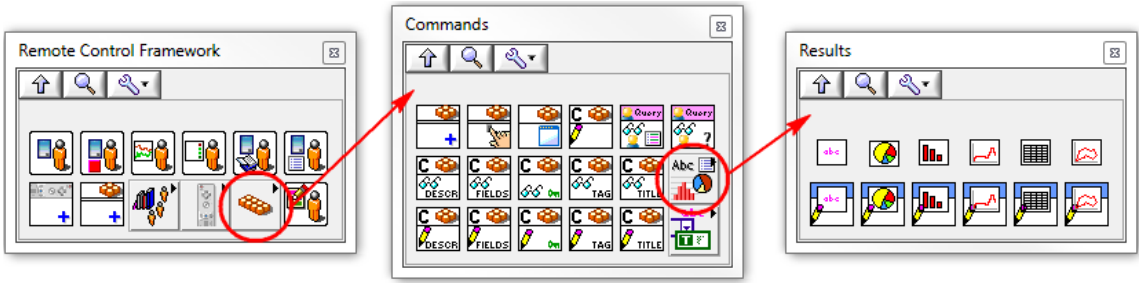


Figure 19- Result constants are in result Palette

Although client's interfaces are not exactly the same on different platforms, due to limits and particularities of their OS and capabilities, some common features are guaranteed:

- All results are displayed on black or very dark background
- All charts are shown to fit client display, regardless of its screen resolution.
- Table results can be browsed with scrollbars if needed
- Text results are displayed without length restrictions.

RCF clients are available on different platforms: from powerful PCs to old generation smartphones. Some considerations have to be remarked about command's results:

- Avoid sending large set of data unless necessary because devices with small computational power cannot manage them (for example, avoid sending charts with 10000 points)
- Avoid sending multiple results with a single command if your clients have no capability to display them. Unnecessary results consume communication band and make difficult to understand how your system is working.

TEXT

Text result returns a string to a client who made the request. The following figure illustrates the block diagram of a command which executes some actions and returns the completion time to user.

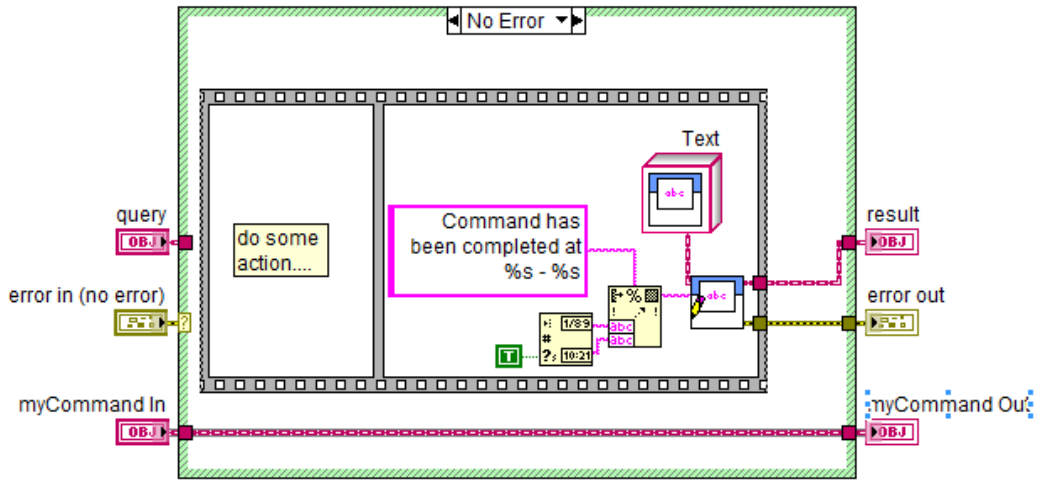


FIGURE 20- EXECUTE.VI METHOD WHICH RETURNS A TEXT RESULT.

CHART

Chart result returns a static chart composed of a single plot. A command can customize chart title, axis labels and plot color. The following example illustrates a command that generates a Chart result with a random data plot.

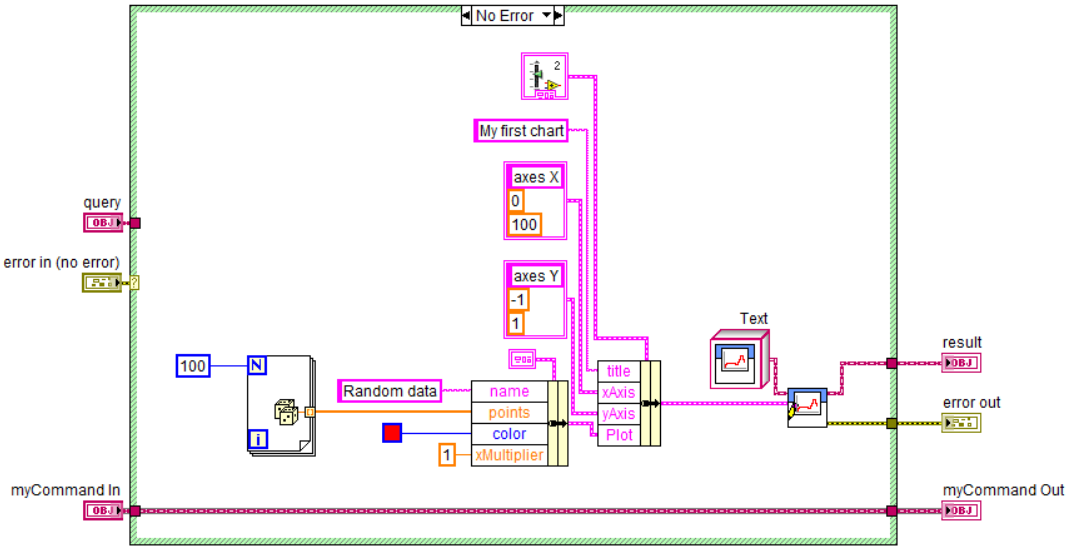


FIGURE 21- EXECUTE.VI METHOD WHICH RETURNS A CHART RESULT.

Histogram

Histogram result returns a static histogram; you can easily compose it, as shown below, with an array of clusters. Each element contains three values: bar label, bar color and its value. Notice that Y axis is not managed with auto scale so you have to set its minimum and maximum values.

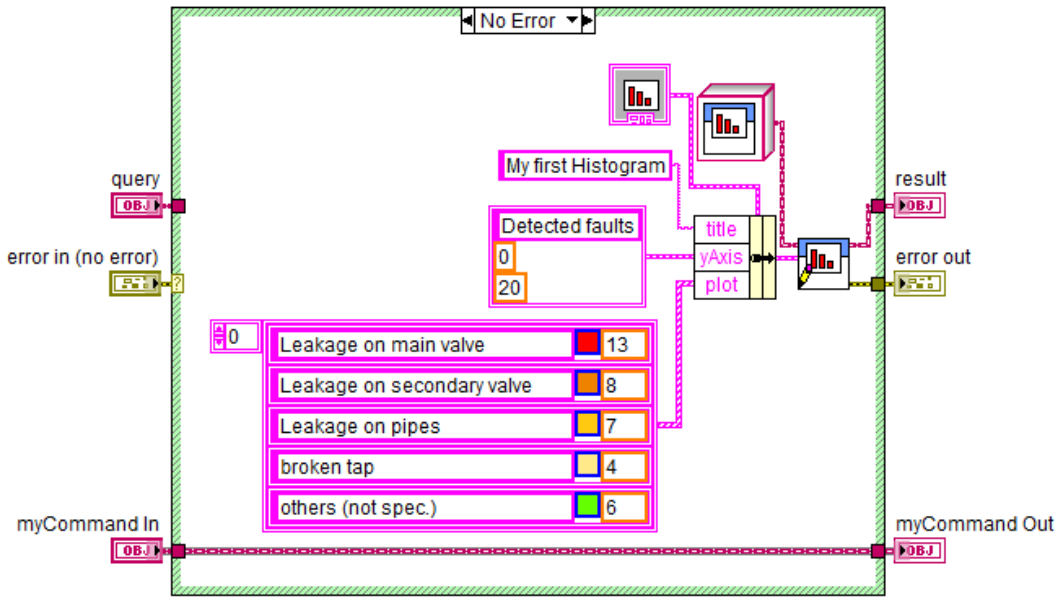


Figure 22- execute.vi method which returns an Histogram result.

Piechart

Piechart result returns a static Pie chart, where each slice is described by its label and value, included into the range 1..360 as illustrated in the following figure.

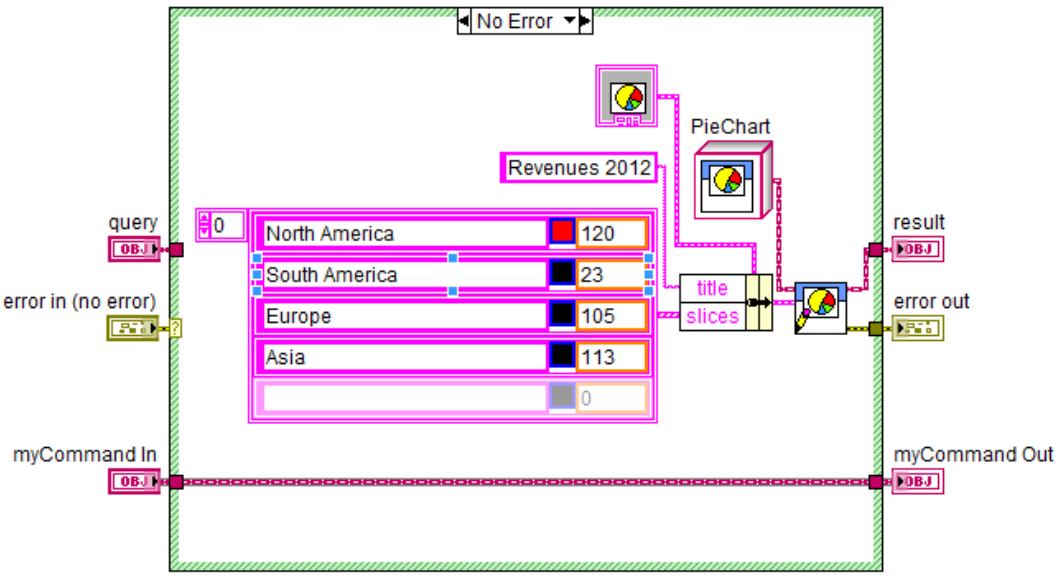


Figure 23- execute.vi method which returns a Pie chart result.

Avoid adding empty slices to pie chart (i.e. slices with size equal to zero) because pie chart legend has a limited dimension on mobile devices.

XY Chart

XY Chart result returns a static X-Y chart. Following figure shows the case a XY chart is created from data stored into a notifier. This approach is used when a main application acquires and processes data in different tasks and your system makes last processed data available to users.

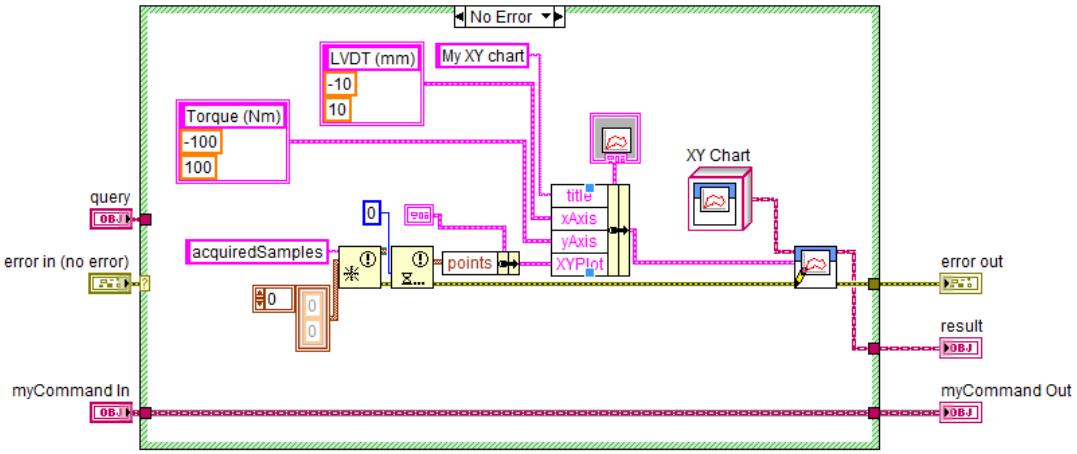


FIGURE 24- EXECUTE.VI METHOD WHICH RETURNS A XYCHART RESULT.

Table

Table result returns a table. You can customize rows and columns headers. RCF checks that a column header array has the same size of the columns of the table. If not so, RCF removes unnecessary column headers or table columns without headers. The following example shows the code necessary to create a Table result.

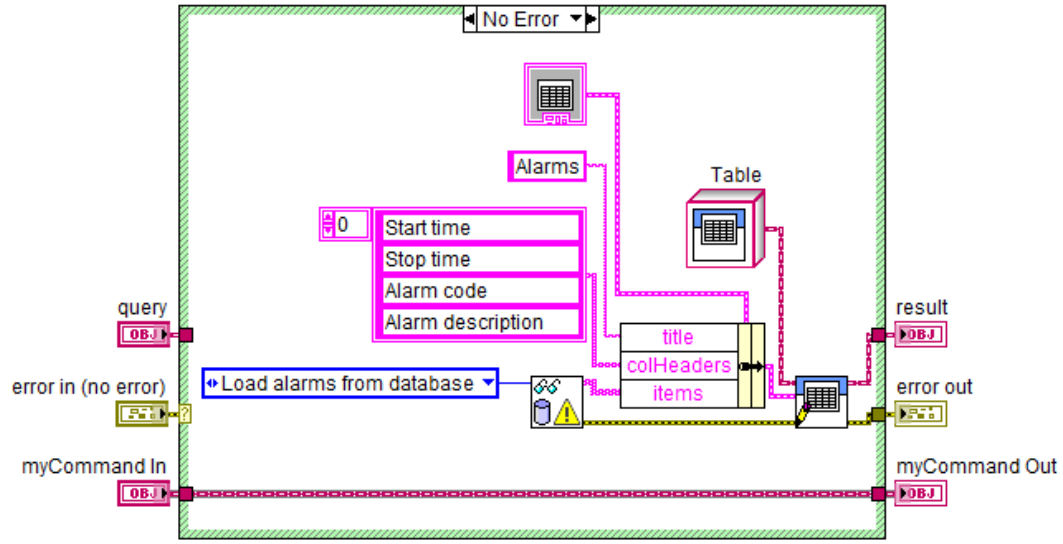


FIGURE 25- EXECUTE.VI METHOD WHICH RETURNS A TABLE RESULT.

The example above shows the case in which Table result includes column headers only. Remember that column headers are necessary to clients in order to visualize properly the data of the table. Row headers are not required: RCF transmits as many rows as those included in **items** value.

Protecting commands with a password

If a command has to be executed by authorized staff only, you have to associate a password (blank passwords are not allowed for security reasons) with that command. When you select a command protected by a password, a request is sent with the just typed password to the client. RCF validates the command execution request by verifying that the typed password is correct. If the password is correct, the **execute** method is invocated, otherwise RCF notifies to the client that the typed password is not correct.

Adding fields to a command

If your command requires some input from user, RCF allows you to define a form composed by a set of different fields. Forms can be composed with three types of fields, as explained below. Forms have no limitation to the number of fields.

Field types

Available field types are

- Text: represents a generic string value
- CheckboxField: represents a Boolean value
- ListboxField: represents a list of string values. User can select one string value only
- Numeric: represents a numeric value both integer and real

You can programmatically create and edit field using Fields palette indicated in the following figure:

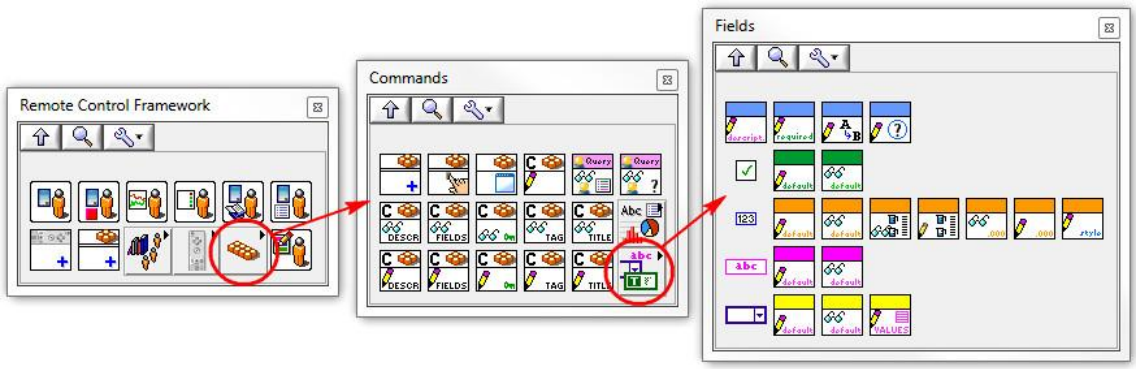


Figure 26- "Fields" palette.

The following paragraphs explain in detail several field types you can use in commands.

Text

The *Text* field is a text string with no length limits. It is possible define the following properties:

- Name: It is the name placed on the left of the text box in the form on your terminal.
- Default value: it is the value given to you in the form on your terminal.
- Required field: if *Required* property is True, this field cannot contain an empty string.

CheckboxField

Checkbox field is a Boolean value. You can specify **defaultValue** to this field. **Required** property doesn't apply to Checkbox field because it is always returned regardless its value.

ListboxField

Listbox field is a list of string values. You can specify **defaultValue** and **required** properties. Use **Write values.vi** to specify strings that populate the list. **DefaultValue** must be one of the values in the list.

NumericField

Numeric field is a numeric value. You can specify **defaultValue** and required properties. You can specify the number of decimal figures with **Write decimals.vi,** by default RCF uses 0 decimal figures**.** You can limit the numeric input range to clients using **Write min&maxValuea.vi,** if you don't specify any range, RCF sets min and max value to -∞ and +∞ respectively and clients don't limit user inputs for numeric fields. **Required** property does not apply to Numeric field because it is always returned regardless its value. RCF supports two different styles for numeric fields. The first one (Defaul style) is the classical style of numeric fields; it is a box in which users types numbers. The second style supported by is an horizontal bar. You can set the value of the numeric field simply by moving the bar. The following figure illustrates the case where a numeric field is created, with 3 digits after decimal point, a range between 10 and 40, and a default value of 20. Notice that default value has to be inside the specified range.

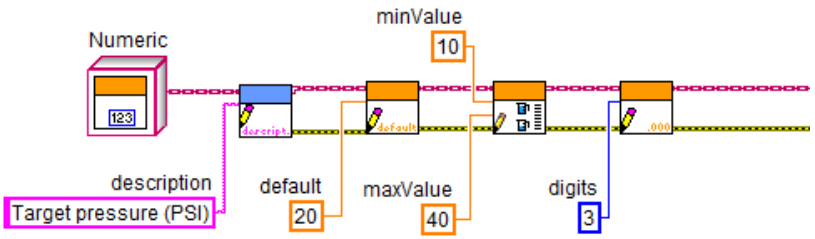


Figure 27- Setting a numeric field.

Creating a command with fields

The following block diagram shows a command with two fields. The next figure illustrates how the form is displayed on client console.

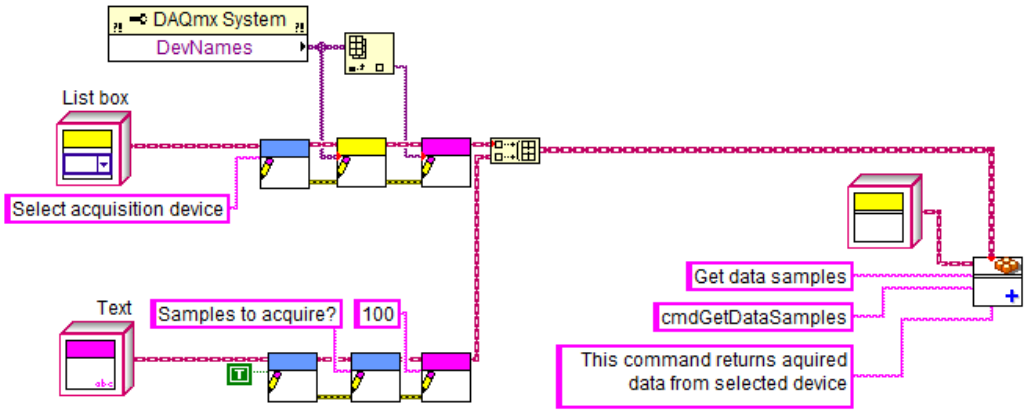


Figure 28- a command is created with two fields.

Command called "get data samples" has two fields called respectively "Select acquisition Device" and "Samples to acquire". The first is a listbox and values are populated at runtime with the names of active devices. The second parameter has a default value of 100 (samples). Remember that fields are displayed to user in the same order they are placed into the field's array.

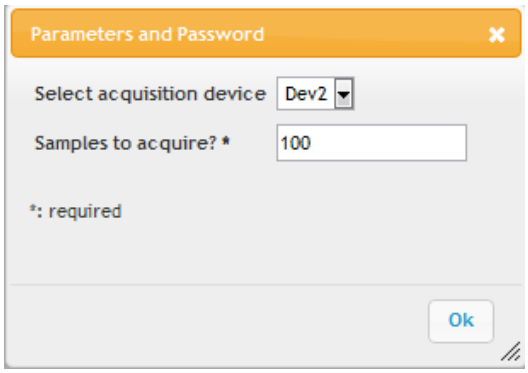


FIGURE 29- A FORM WITH TWO PARAMETERS GENERATED BY RCF.

RCF marks required fields with an asterisk. User has to fill the form and press OK to send a request to your application. This ensures that all required fields are received by **execute.vi** method when invoked. When a user sends its requests, RCF collects field values into a string array available from query objects. For example, if a field value is a numeric field, it must be converted in a numeric format.

The following block diagram shows what explained before:

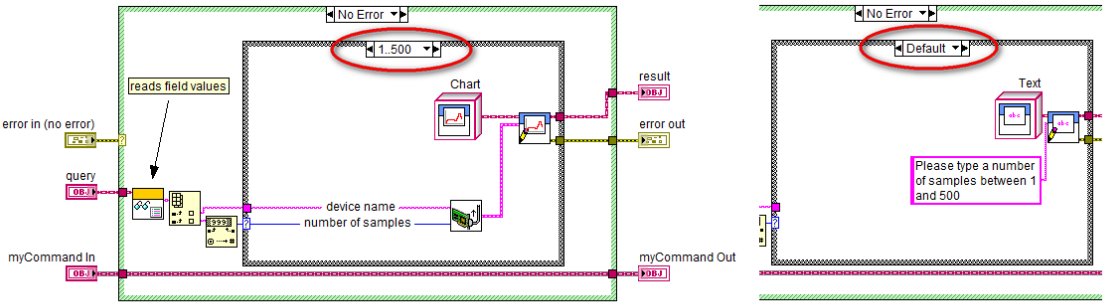


FIGURE 30- EXECUTE.VI PROCESS DATA ACCORDING TO FIELD VALUES.

Notice that field values are processed in the same order they are placed into field array when command has been created. Second field (Samples to acquire) is checked if its value is included in range 1..5, acquisition is executed, otherwise a text result is used to advice user that the value is out of valid range.

ADDING HELP

Every field can include a Help text displayed on client form. Use Write help.vi method to add help text to a field, as indicated in figure below.

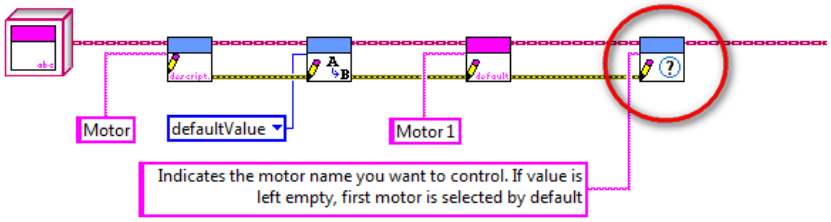


FIGURE 31- USE WRITE HELP.VI TO ADD A HELP TEXT.

FIELD UPDATE MODES

Fields included in commands show values when user activates their associated command. You can choose your favorite way to update the value displayed in a field. RCF allows the use of three different field update modes:

- Default value: RCF client displays always the default value defined in its command. Even if user inserts a value, default value will be shown at the next command execution.
- Last value: RCF client retains the last value edited from user. At the first command execution, RCF client uses default value indicated in command definition.
- Server value: RCF client updates the field value with values received from server during connection time. Updates are managed in background. When user activates a command, field values are the most recent values published by server.

A command can contain fields defined with different update modes. The following example illustrates the case of a command composed by three fields with different update modes.

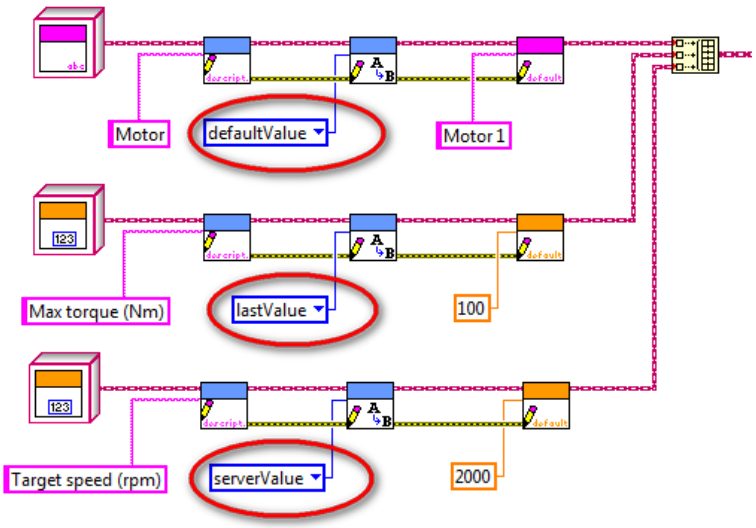


FIGURE 32- USE WRITE UPDATEMODE.VI TO SET THE PROPER VALUE UPDATE MODE FOR EVERY FIELD.

Use **Write updateMode.vi** to set the proper way RCF client manages the field values. By default, RCF uses ***Default value*** update mode in order to maintain compatibility with previous versions of RCF clients. RCF server can update field values at any time using polymorphic VI **refreshFieldValue.vi** in RCF palette. Notice that only fields defined with ***update mode = serverValue*** are updated.

The figure below illustrates the code required to refresh the field value of the third field (Target speed) indicated in previous image.

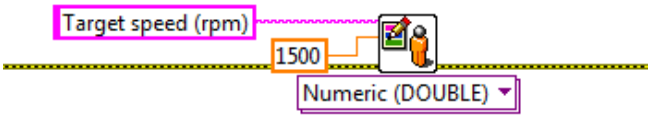


FIGURE 33- USE POLYMORPHIC REFRESHFIELDVALUE.VI TO UPDATE FIELD VALUES TO ALL CLIENTS.

Field values are updated using their description as search key. **refreshFieldValue.vi** in previous figure associates "Target speed (rpm)" to the new value 1500 instead of 2000 used at definition time. When a user execute the command that includes that field, field value is 1500 instead of 2000. If you specify an empty string as field description, **refreshFieldValue.vi** returns an error. If description string doesn't match any field description, non error occurs, but no updates are transmitted to clients.

PUBLISHING SYSTEM STATUS

RCF allows your server to publish a status table to all connected clients without explicit request. Sending a system status is useful when clients need to know system status changes whenever they happen.

IMPORTANT: while results are sent to users who made specific requests, system status is transmitted to all connected clients.

System status is transmitted as 2D table composed by two columns: the first is the property name and the second one is the property value. Suppose your system controls the temperature of a heat exchanger, your system status can be as follow:

Property name	***Property value***
Oil temperature	96 °C
Water temperature	24 °C
Current status	normal
Warning	None
Water flow	368 L/h
Heating system	Off

If you want to inform clients with system status when a warning is generated or status changes, you have simply to publish the system status with **refreshStatus.vi** as indicated below:

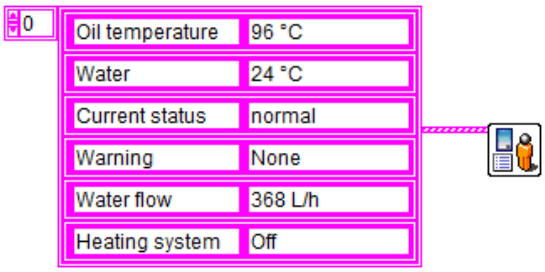


FIGURE 34- REFRESHSTATUS.VI TRANSMITS THE SYSTEM STATUS TO ALL CONNECTED CLIENTS.

The above example contains a constant status to explain how to format the 2D string array properly.

MANAGING EVENTS WHEN A CONNECTION STARTS/STOPS

A catalog can contain two commands associated to specific events:

- New connection established, executed every time a new client connects to RCF server.
- Connection terminated, executed every time a connection is terminated. Connection can be terminated for two reasons: remote user closes the client app or timeout error occurs when due to poor quality of communication channel.

Create your own commands extending Command class and override **execute.vi** methods. The following figure illustrates the necessary code to associate your custom command to "New Connection" event, , using **Write onNewConnection.vi** method.

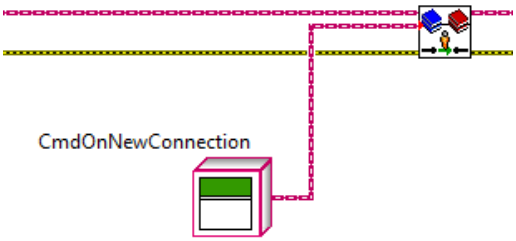


FIGURE 35- ASSOCIATE YOUR CUSTOM COMMAND OBJECT TO THE NEW CONNECTION EVENT.

Do not associate any field to those commands: RCF creates automatically three fields filled at runtime with client information:

- Name. It indicates the user name provided by client. This value is useful when server needs to associate user names to their IP addresses.
- Timestamp. It indicates the time connection has been established at server side.
- Platform. It indicates which platform is used by client (Android, iOs, LabVIEW, HTML5).

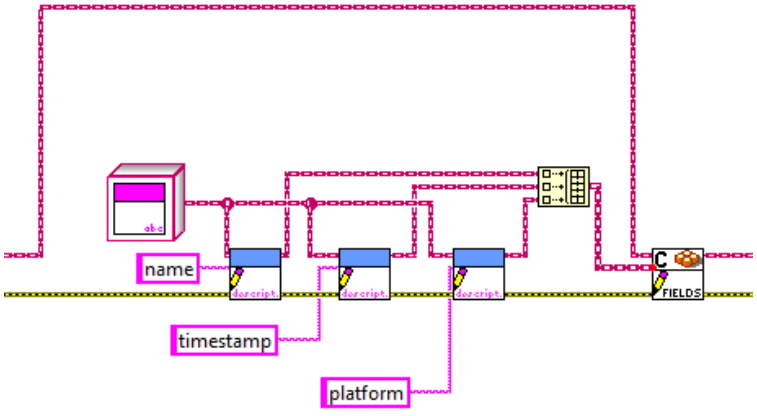


FIGURE 36- INTERNAL CODE GENERATED TO ASSOCIATE THREE PARAMETERS TO YOUR CUSTOM COMMAND FOR "NEW CONNECTION" EVENT.

The following figure illustrates the necessary code to associate your custom command to "Closed Connection" event, using **Write onClosedConnection.vi** method.

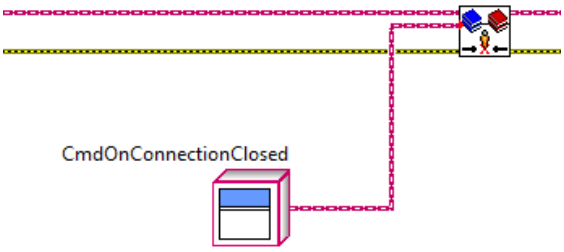


FIGURE 37- ASSOCIATE YOUR CUSTOM COMMAND OBJECT TO THE CLOSE CONNECTION EVENT.

Do not associate any field to those commands: RCF creates automatically two fields filled at runtime with client information:

- Reason. It describes the reason connection has been closed
- Timestamp. It indicates the time connection has been closed at server side

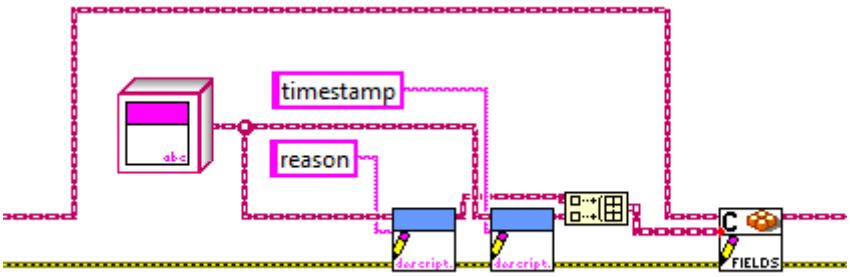


FIGURE 38- INTERNAL CODE GENERATED TO ASSOCIATE TWO PARAMETERS TO YOUR CUSTOM COMMAND FOR "CLOSED CONNECTION" EVENT.

MANAGING THEMES

RCF allows customizing client's aspect with themes. A theme is a collection of properties that affects the aspect of RC buttons, texts and background. Themes are defined at server side so each application can include a specific theme. When clients connect to server for the first time, download theme package from server. Theme data remain resident on smartphone. Theme is a zip file that contains images in JPG format and a configuration file in XML format. To add a theme to your catalog, use **load theme.vi** method as indicated in figure below.

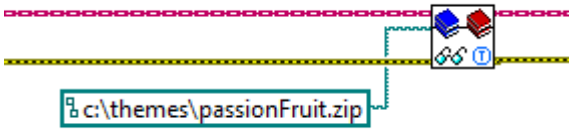


FIGURE 39- LOADING A THEME WITH LOAD THEME.VI METHOD.

If no theme is specified, Clients use their default theme which may vary for every platform (Android, iOs, HTML5). Notice that that RCF client for Windows does NOT support themes.

PROTECTING REMOTE CONTROL WITH A PASSWORD

If a remote control has to be viewed by authorized staff only, you have to associate a password (blank passwords are not allowed for security reasons) with that remote control. When you select a remote control protected by a password, a request is sent with the just typed password to the client. The figure below illustrates how to protect a RC with a password, using **Write password.vi** method

FIGURE 40- PROTECTING A REMOTE CONTROL WITH WRITE PASSWORD.VI METHOD.

When user enter correct password associated to a remote control, RCF client display RC commands, remembers that user knows the RC password and unlocks the RC for the current session. User can move to other RCs and return to the password protected RC later without re-entering the password.

ADVANCED FEATURES

In this paragraph are illustrated the RCF advanced functions. Catalogs can be stored and loaded into form files. You can create a catalog with custom commands, save it into a file and distribute the catalog to other applications which includes RCF and the same classes. Remember that a catalog is strictly tied to the LabVIEW classes created to extend Command class. For this reason you have to be sure that executables contain at least a constant element for every class you have created.

EDITING CATALOG AT RUN-TIME

Editing catalog at run-time is one of the advanced functions. RCF includes **editor.vi**, a general purpose editor that you can use to create a complex catalog composed by one or more remote controls, each with one or more commands. With **editor.vi** you can also customize every command with additional fields. Catalogs can change at run-time, RCF does not publish catalog changes automatically so you have to use **refreshCatalog.vi** as indicated in the following example.

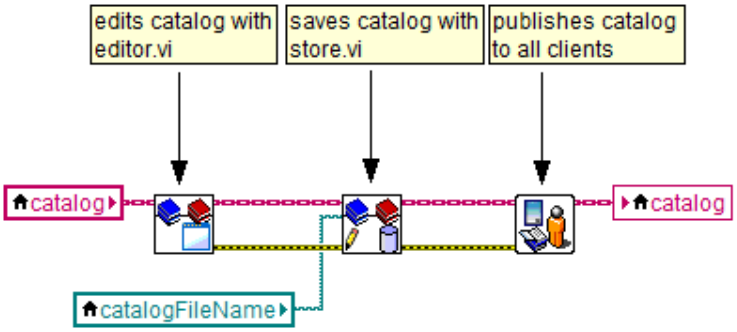


FIGURE 41- EDITING, SAVING AND PUBLISHING CATALOG WITH THREE METHODS.

ADDING CLASSES TO CATALOG

Using **editor.vi** to edit your catalog at run-time requires that custom command classes are added to catalog itself. Use **writeClassList.vi** method to include your custom commands to the catalog so that **editor.vi** can properly manage all commands.

PUBLISHING REAL-TIME DATA

RCF allows publishing real-time data to connected clients. it means that your LabVIEW program can produce/acquire analog and digital values and update client charts and tables in real-time. RCF is based on SCCT communication library and data are transferred through TCP permanent connections between server and clients to ensure that no data are lost.

Remember that displaying real-time analog data on dynamic charts can be a CPU intensive task for mobile devices, so it's your responsibility avoiding huge data packets to clients. Usually users need or know signal trends or digital lines status so updating data to clients at high rate (more than 10 times per seconds) or sending hundreds or thousands of samples can be useless. Notice also that RCF client do not perform log activities and all received data are displayed or discarded.

In the following paragraphs, the case of analog and digital data are illustrated. RCF allows publishing analog and digital data at the same time.

PUBLISHING REAL-TIME ANALOG DATA

To publish real-time analog data to clients, you have to proceed as follow:

1. Add a configuration cluster to **RCF-Manager.vi**, with a description of analog channels you plan to publish. This is necessary to client to understand that you server can publish analog data and prepare related data structures and tasks. Configuration-cluster is the classic system configuration cluster used by SCCT. Refer to SCCT User Manual for more details.
2. Publish analog data with **publishAnalogData.vi.**

The following example illustrates the code necessary to complete this activity.

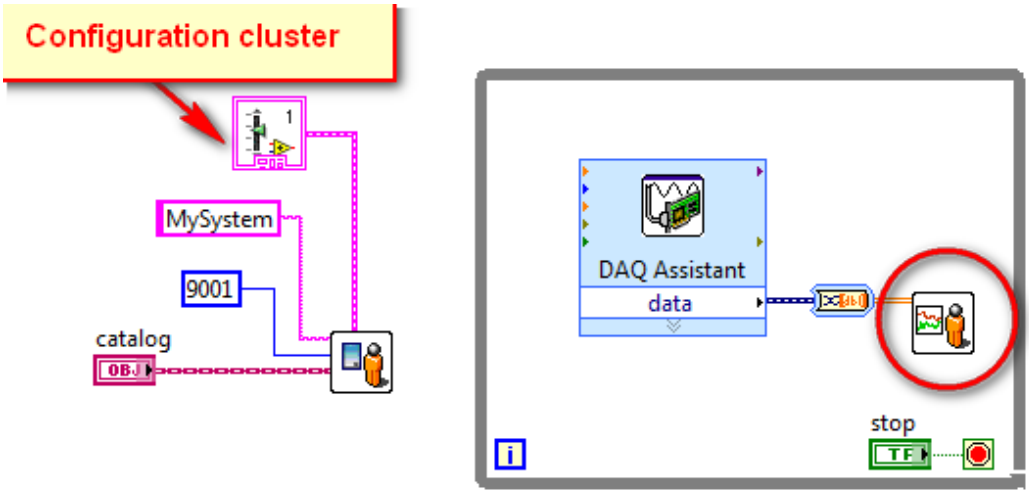


FIGURE 42- PUBLISHING REAL-TIME ANALOG DATA.

publishAnalogData.vi requires a 2D DBL array in input that contains your analog data organized in rows: every row contains samples of the same channel. Optionally you can connect a 1D array of colors: the number of color values has to be equal to the number of rows (i.e. channels) in 2D data.

PUBLISHING REAL-TIME DIGITAL DATA

To publish real-time digital data to clients, you have to proceed as follow:

1. Add a configuration cluster to **RCF-Manager.vi**, with a description of digital lines you plan to publish. This is necessary to client to understand that you server can publish digital data and prepare related data structures and tasks. Configuration-cluster is the classic system configuration cluster used by SCCT. Refer to SCCT User Manual for more details.
2. Publish digital line values with **publishDigitalData.vi.**

The following example illustrates the code necessary to complete this activity.

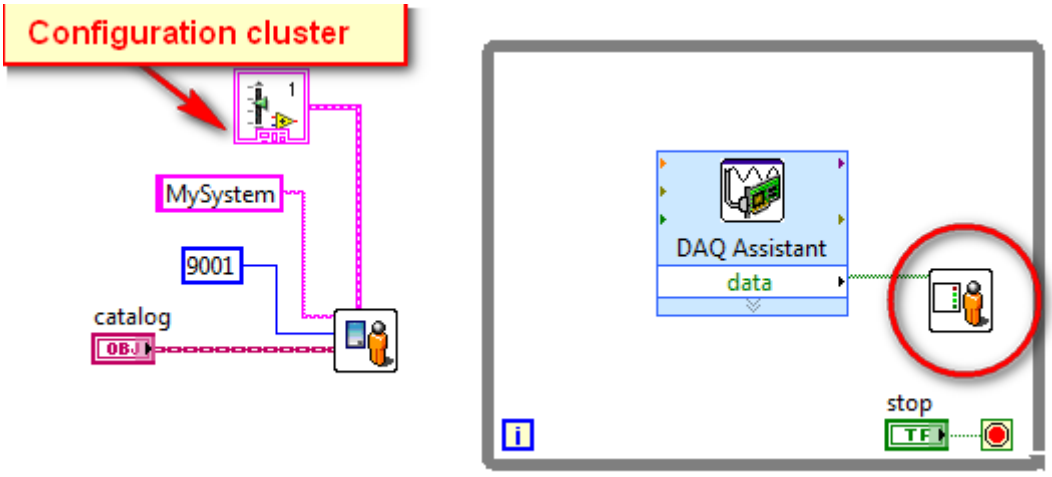


FIGURE 43- PUBLISHING REAL-TIME DIGITAL DATA.

publishDigitalData.vi requires a 1D Boolean array in input that contains your digital line values. Optionally, you can connect two 1D arrays respectively for TRUE and FALSE colors: the number of color values has to be equal to the number of digital lines in 1D data.

ADDING SYSTEM HELP PAGE

In many real life applications, it is important to create a web page that contains a description of your system, with detailed explanations of RC's commands and their parameters. You can specify the complete URL of that page to **RCF-Manager.vi** so that clients can display it to users. To add a system help page, simply connect the **helpURL** parameter as indicated in the following example.

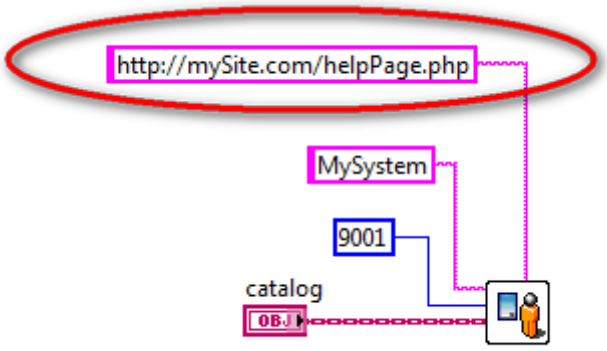


FIGURE 44- ADDING HELPURL TO RCF-MANAGER.

RCF CLIENTS

RCF clients are available for different platforms and devices. For a complete list of supported systems, visit:

www.toolsforsmartminds.com/products/remote_control_for_labview.php

The Remote Control client for Android OS is available at:

https://play.google.com/store/apps/details?id=SCCT.Console

The Remote control client for HTML5 is available at:

http://www.toolsforsmartminds.com/remote_control_html5/

Remote control for HTML5 requires one of the following browsers:

- Google chrome
- Mozilla Firefox
- Internet Explorer 9 or higher with Flash Plugin
- Safari

The remote control for Windows XP/7 is available at:

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

This client, created with LabVIEW includes installer and LabVIEW runtime engine.

FIGURE LIST

INDEX

U

V