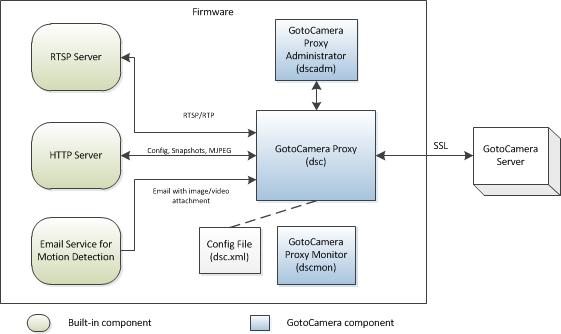
GotoCamera Proxy Integration Guide

# Overview

GotoCamera proxy enables an IP camera to act as a GotoCamera client. The proxy has been designed in such a way that it can be easily integrated into an IP camera with minimum changes to its firmware. The following diagram illustrates how it works.



The GotoCamera proxy consists of three programs:

* **Proxy**. The main program that handles communication between the camera and GotoCamera server.
* **Administrator**. It implements a console-based interface to Proxy, which can be easily integrated with most IP cameras web-based administration interface.
* **Monitor**. Monitors Proxy and starts or restarts it if necessary to ensure its healthy and continuous running status.

All three programs, once compiled, are simple command line utilities that have little external dependencies. We will first show how each program works and then look at how to integrate them into camera firmware.

# GotoCamera Proxy

The proxy program implements the following services:

* Bridging the communication between the camera’s built-in services and GotoCamera servers. This is its main functionality that turns an IP camera into a GotoCamera client.
* An administration service that listens on a preconfigured port (*adminport*) for incoming requests. A program may connect to the service and send requests to query its status, modify its settings or even trigger a Motion Detection notification. The Administrator utility is based on this mechanism.
* A simple SMTP service that enables cameras to integrate their existing SMTP-based motion detection alerts with GotoCamera.

The program can be run as a standard command line utility:

$ ./dsc -h

Usage: dsc [-h] [-v|-V] [-s] [-c cert\_file] [-m max\_iobuf] [-t block\_size]

[-o output\_file] [config\_file]

default config\_file: dsc.conf

-h print this menu

-v verbose debug logging

-V extra verbose debug logging

-s write status dump to standard output, default dsc.status

-c ssl certificate file. if missing, hc.pem in the current directory is used

-m maximum buffer size for I/O in bytes, default 64000000

-t test heap memory allocation using specified block size

-o write logging output to file

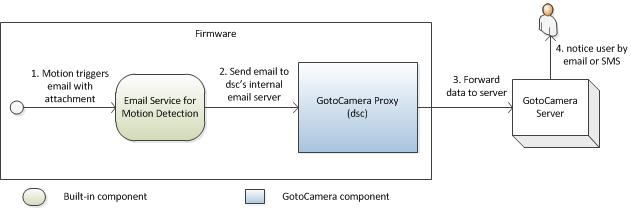
-b maximum size of log file (in kb) to maintain <default: 512 kb>

-n maximum number of log files to maintain <default: 2>

As shown, the program requires two files to run, a configuration file and a certificate. Command options can be used to load those files from anywhere on the file system. Otherwise, default files in the current directory will be used. Many options can help test the proxy program in a standard Linux environment before moving it onto camera firmware.

## Motion Detection Alert by Email

Many cameras allow users to configure a list of SMTP servers so that when motion detection is detected, an image or video will be captured and sent as an email attachment to those servers. This can be easily integrated with GotoCamera using the Proxy program’s SMTP service.



Assuming the Proxy program runs its SMTP service on the default port 2500, adding 127.0.0.1:2500 to the list of email servers will allow Proxy to forward received email data to GotoCamera servers as a motion detection alert. GotoCamera servers will, in turn, extract media data from the email and save it into the user’s media archive as a motion detection alert, and send an email or SMS to notify the user (as configured via the My Account page on the website).

## First-time Remote Configuration by Device Key

After the proxy program starts, it checks whether the <username> and <password> settings are both empty; if so, it then checks if <devkey> has been set (non-empty), and if yes, the proxy program connects to the GotoCamera servers as a “raw” device.

A user may login to the GotoCamera website and choose to add a raw camera. The system will prompt the user for the device key of the raw camera and if the user correctly provides the device key for some online “wild” camera, GotoCamera will set the username and password of the camera to match that of the user’s. This feature makes it possible to remotely setup a new camera for GotoCamera from the GotoCamera website instead of the camera’s local web interface. This helps de-clutter the camera’s interface and makes it easier and faster for the user to start using the service and the camera.

By default, a camera’s MAC address is used as its device key. The Proxy Administrator program can be used to query a camera’s device key. See the next section for details.

To prevent malicious users to get control of other people’s raw cameras by guess their device keys, remote configuration is disabled by default for raw devices. The Proxy Administrator program provides a “rconf” command, which can be used to notify GotoCamera Server to temporarily enable remote configuration by device key.

## Configuration

The proxy program keeps its settings in an XML file. The following sample configuration can be used as a template. Refer to its detailed explanations if you want to make changes to it.

<setting>

<enableupnp> no </enableupnp>

<controllerid> </controllerid>

<adminport> 20000 </adminport>

<smtpport> 2500 </smtpport>

<devkey> </devkey>

<username> </username>

<password> </password>

<servers>

<server>

<address> client1.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client2.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client3.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client4.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client5.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client6.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client7.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client8.gotocamera.com </address>

<port> 443 </port>

</server>

<server>

<address> client9.gotocamera.com </address>

<port> 443 </port>

</server>

</servers>

<devices>

<device>

<name> </name>

<description> </description>

<address>type=tcp;host=127.0.0.1;port=81</address>

<id>type=airlink;subtype=generic;sysid=000000000000</id>

<credentials>

<username>admin</username>

<password></password>

</credentials>

</device>

</devices>

</setting>

The top-level settings are explained in this table:

|  |  |
| --- | --- |
| **Setting** | **Description** |
| <enableupnp> | Enable UPnP ? Set to yes only for routers. Should always be no for IP cameras. |
| <controllerid> | For internal usage only. Leave it empty. |
| <adminport> | Port for administration service. |
| <smtpport> | Port for SMTP service. |
| <devkey> | Unique device key for first-time remote configuration. Set by camera manufacturer. If left empty, the Proxy program will automatically use MAC address as device key. |
| <username> | GotoCamera account username. |
| <password> | GotoCamera account password |
| <servers> | List of GotoCamera servers. Proxy may connect to any of them. |
| <devices> | List of cameras. For IP camera, there should be only one entry. |

The entries in <devices> have the following structure:

|  |  |
| --- | --- |
| **Setting** | **Description** |
| <name> | Camera name |
| <description> | Camera description. |
| <address> | Address of the camera’s ‘control’ service. For cameras with a CGI interface, this should point to its internal HTTP service. |
| <id> | Camera device ID. GotoCamera uses this to uniquely identify a camera. |
| <credentials> | Optional. User credentials required accessing the cameras CGI interface. |

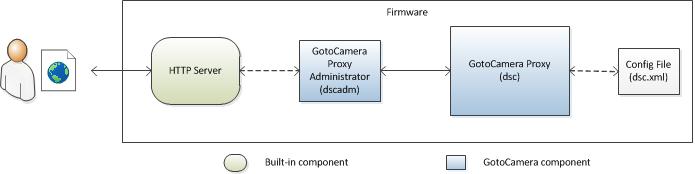
The <id> tag that uniquely identifies a camera has the following format:

type=airlink;subtype=generic;sysid=000000000000

where <type> identifies the make of the device, <subtype> its model name. <sysid> must match a device specific unique identifier, such serial number or Ethernet address. *However, if the device’s host address is localhost or 127.0.0.1, <sysid> can be set to any string as server ignores its value anyway*. When the proxy runs inside the camera’s firmware, it always connects to local service at 127.0.0.1, <sysid> can be set to a generic value for all the cameras. This simplifies firmware integration.

# GotoCamera Proxy Administrator

This utility serves as the firmware’s interface to the Proxy program. It uses Proxy’s administration service but hides all the implementation details behind an easy-to-use command line interface.



The Administrator program is designed to simplify the configuration of the Proxy program through the camera’s built-in web interface, as illustrated in above figure. User performs a GotoCamera configuration related operation in the web browser that results in a POST request with form data sent to the camera’s built-in HTTP server. The HTTP server invokes the Administrator program with proper command line options and returns its result back to the user’s browser to display. As far as the HTTP server is concerned, the Administrator program is just a simple command line utility that prints its output to standard output. The exact details of how it interfaces with the Proxy program are irrelevant.

Running the program without any options prints out help:

$ ./dscadm

rc ok

Description The program writes regular output to stdout and debug messages to stderr, to separate debug from regular, redirect stderr to something else, or use the '-d' or '-o' options described below

Usage ./dscadm <options> <command> <command\_parameters>

Options

-h <host> dsc address, default to 127.0.0.1

-p <port> dsc port, default to 20000

-v <level> debug message level from 0 (quite) to 3 (verbose), default to 0

-f <format> output format, can be json or text, default to text

-o <file> name of file for output, default to stdout

-d <dbgfile> name of file for debug message output, default to stderr

Commands

version Print version info

help Print help

get Get dsc properties

set Set dsc properties

notify Send notifications to dsc

restart Request dsc to restart

reset Reset dsc settings to factory default

rconf Remote configuration control

To get help on a particular command, use

./dscadm help <command>

Note the following:

* The program writes its regular output to stdout and debug messages to stderr. To process program results correctly, only output to stdout should be used.
* The first line (or object if in JSON format, see below) of program output always shows run status named **rc**. If its value is **ok**, the program has run successfully and the following lines have the detailed information. Otherwise program run failed and the value of rc is an error string. This convention is followed in all situations.

This is the standard text format output. It can also format its output in JSON format:

$ ./dscadm -f json

{

"rc": "ok",

"Description": "The program writes regular output to stdout and debug messages to stderr, to separate debug from regular, redirect stderr to something else, or use the '-d' or '-o' options described below",

"Usage": "./dscadm <options> <command> <command\_parameters>",

"Options":

{

"-h <host>": "dsc address, default to 127.0.0.1",

"-p <port>": "dsc port, default to 20000",

"-v <level>": "debug message level from 0 (quite) to 3 (verbose), default to 0",

"-f <format>": "output format, can be json or text, default to text",

"-o <file>": "name of file for output, default to stdout",

"-d <dbgfile>": "name of file for debug message output, default to stderr"

},

"Commands":

{

"version": "Print version info",

"help": "Print help",

"get": "Get dsc properties",

"set": "Set dsc properties",

"notify": "Send notifications to dsc",

"restart": "Request dsc to restart",

"reset": "Reset dsc settings to factory default",

"rconf": "Remote configuration control"

},

"To get help on a particular command, use ":

{

"./dscadm help <command>": ""

}

}

The help command can be used to display the details of each command.

## Common Usage Examples

The following examples show how to do some common tasks with JSON output.

### Get GotoCamera username and password

$ ./dscadm -f json get sys

{

"rc": "ok",

"tid": "1",

"properties":

{

**"CTRL\_STAT":**

{

…

**"username": "joe@xyz.com",**

**"password": "abc",**

**…**

}

Note that the get sys command returns extensive information on GotoCamera proxy. In another example below, it is used to retrieve system status information as well.

### Set GotoCamera username and password

$ ./dscadm -f json set sys un=joe@gmail.com pw=abc

{

"rc": "ok",

"tid": "1",

"errcode": "0"

}

### Activate GotoCamera proxy remote setup

$ ./dscadm -f json rconf

{

"rc": "ok",

"tid": "1",

"properties":

{

"RC\_ACTIVATION":

{

"rc": "ok",

"url": "https://www.gotocamera.com/deviceSetup?devkey=0026b0d62f14"

}

}

}

The rc property under RC\_ACTIVATION (not the top level rc) should be checked for result status. If the value of rc is not ok, it’s an error message that indicates the reason of failure. If rc is ok, an additional property url will be present, whose value is the destination to which the user’s browser should be redirected to to start remote setup.

### Get GotoCamera proxy device key

$ ./dscadm -f json get devkey

{

"rc": "ok",

"tid": "1",

"properties":

{

"devkey": "0026b0d62f14"

}

}

### Get GotoCamera proxy network status

$ ./dscadm -f json get netstat

{

"rc": "ok",

"tid": "1",

"properties":

{

"NETSTAT": "Online (client3.gotocamera.com:443), Online"

}

}

### Get GotoCamera proxy log

$ ./dscadm -f json get log

{

"rc": "ok",

"tid": "1",

…

}

### Get GotoCamera proxy status, device key and client version

$ ./dscadm -f json get sys

{

"rc": "ok",

"tid": "1",

"properties":

{

**"CTRL\_STAT":**

{

"platform": "i386",

"os": "Darwin",

"client\_type": "4",

**"client\_ver": "10001",**

**"client\_build": "102",**

"protocol\_ver": "10000",

"system\_id": "0026b0d62f14",

**"stat\_description": "Registration Error (Authentication failed)",**

"ctrlid": "71002",

"id": "1",

"tmcreate": "1335759928",

**"stat": "3",**

"tmstat": "1335759928",

**"devkey": "0026b0d62f14",**

…

"smuxes":

{

…

},

"admsmuxes":

{

…

},

"didlut":

{

…

},

**"datsrcs":**

{

"1":

{

…

"LUT\_ID": "0",

"LUT\_STATUS": "DEV\_LUT\_UNPLUGGED",

"USERNAME": "admin",

**"stat\_description": "Camera offline",**

"id": "2",

"tmcreate": "1335759928",

**"stat": "1",**

"tmstat": "1335759928",

…

}

}

}

}

}

As we can see, all the required information can be found in the result of get sys. Device key is named “devkey” under **CTRL\_STAT**. Client version consists of two parts, a version number and build number.

Since the proxy program maintains connections to both GotoCamera server and the camera, its status is described by two components, controller status and data source status. Controller status, as **stat** an integer and **stat\_description** a user-friendly description of the stat, shows the status of the proxy’s connection to GotoCamera server. The following values are defined:

|  |  |
| --- | --- |
| stat | description |
| 1 | Offline. Not connected to server. |
| 2 | Connected to server, proxy registration is in progress. |
| 3 | Connected to server, but failed to register proxy with GotoCamera server. |
| 4 | Online. Connected and registered. |

Data source status shows the status of the camera under the proxy’s control. Similarly, it’s represented as **stat** an integer and **stat\_description** a user-friendly description, which may have the following values:

|  |  |
| --- | --- |
| stat | description |
| 1 | Offline. Not connected to camera. |
| 2 | Connected to camera, but the camera can not be identified. |
| 3 | Camera connected and identified, waiting to register with server. |
| 4 | A connection to server has been assigned to the camera. |
| 5 | Camera registration is in progress. |
| 6 | Camera registration failed. |
| 7 | Online. Camera registration is successful. |

### Get camera settings

$ ./dscadm -f json get cam

{

"rc": "ok",

"tid": "1",

"properties":

{

"1":

{

"NAME": "airlink",

"DESCRIPTION": "",

"ADDRESS":

{

"type": "tcp",

"host": "10.3.141.11",

"port": "80"

},

"USERNAME": "admin",

"PASSWORD": "",

"DEVICE\_ID":

{

"type": "airlink",

"subtype": "generic",

"sysid": "0023D302407C"

}

}

}

}

### Set the credentials required to access camera’s HTTP interface

$ ./dscadm -f json set cam op=set cred=un=admin\;pw=admin

{

"rc": "ok",

"tid": "1",

"errcode": "0"

}

### Set local HTTP server’s address

$ ./dscadm -f json set cam op=set addr=type=tcp\;host=127.0.0.1\;port=81

{

"rc": "ok",

"tid": "1",

"errcode": "0"

}

### Request dsc to restart

$ ./dscadm -f json restart

{

"rc": "ok",

"tid": "1",

"errcode": "0"

}

### Request dsc to reset its configuration to the specified file and restart

$ ./dscadm -f json reset dsc\_factory\_default.conf

{

"rc": "ok",

"tid": "1",

"errcode": "0"

}

## Sending Motion Detection

The Administrator program offers another way of integrating camera’s motion detection with GotoCamera. If the camera is capable of capturing an image or video and saving it on local file system as, for example, /tmp/md\_314159.jpg, then the following command:

$ ./dscadm -f json notify md cam stat=1 delete=yes file=/tmp/md\_314159.jpg

would cause the file to be uploaded to GotoCamera server as a motion detection alert. The delete flag instructs dscadm to remove the file after usage. The camera needs to ensure that unique filenames are given to different images.

# GotoCamera Proxy Monitor

This simple program starts the Proxy program and monitors its health by exchanging some heartbeat messages with it. If, for some reason, the Proxy process has exited or stopped replying to heartbeat requests, the Monitor restarts it. To run it, giving it the full path to the proxy program and the necessary parameters like this:

$ /usr/local/bin/dscmon /usr/local/bin/dsc –v –o /log/dsc.log

Running the program without any options prints out help:

dscmon v1.0, 12 March 2012 build 255

Usage: ./dscmon [-h <host>] [-p <port>] [-v <log\_level>] [-o <log\_file> [-m <max\_log\_size>]] <dsc\_path> ...

-h dsc address, default to 127.0.0.1

-p dsc port, default to 20000

-v debug message level from 0 (quite) to 3 (verbose), default to 0

-o output file name, default to stdout

-m maximum log file size in bytes before wrapping around, default to 10000

Some options are the same as those of Proxy Administrator. Like Administrator, it relies on Proxy’s administration service to work.

# Integration

This section outlines the general steps to integrate GotoCamera proxy into firmware.

## Compiling

The source code distribution of the GotoCamera proxy program has the following top-level directory structure:

/polarssl the PolarSSL library, fxlib depends on it

/fxlib utility library, 3rd\_party\_clients depends on it

/3rd\_party\_clients its src subdirectory contains source code for all the proxy programs

/include fxlib header files are copied here for easy reference

/lib fxlib library is copied here for easy reference

Compile polarssl first, the top level Makefile builds all the tests as well. Since we only need the library, we can choose to build the library only, assuming you’ve extracted source code under SRC\_ROOT,

$ cd $SRC\_ROOT/polarssl/library

$ make

This uses the default Makefile. Makefile.mipsel can be used to build for MIPS platforms. Or you can create your own based on these examples. Next, we build fxlib,

$ cd $SRC\_ROOT/fxlib

$ make –f Makefile.linux

Again, there are some other Makefiles that can be used as templates if you need to create a Makefile customized to your specific platform. Finally, compile the executable,

$ cd $SRC\_ROOT/3rd\_party\_clients/src

$ make –f Makefile.linux

After successful compilation, you can find the final programs as dsc.linux, with three symbolic links **dsc**, **dscadm** and **dscmon** all linking to dsc.linux. The functionalities of the Proxy, the Administrator and the Monitor are packaged into a single statically linked binary to minimize footprint. The program looks at the first command line argument (argv[0]) to decide which functionality to dispatch. Thus it’s important to name the symbolic links exactly the same as described above.

Many important system properties are controlled by config.h. There is no config.h in the source directory initially. Makefile.linux copies config.linux to config.h before starting the compilation process. There are some other config files for various platforms included in the source code distribution, which can be used as a good starting point if you want to customize some attributes. In particular, the following macros control the locations of default configuration file and SSL certificate required by dsc:

/\* default configuration file \*/

#define DEFAULT\_CONFIG\_FILE "/etc/gotocamera/gotocamera.conf"

/\* ssl certificate \*/

#define SSL\_CERTIFICATE "/etc/gotocamera/gotocamera.pem"

Change these to point to the locations where you plan to put them on your firmware if you prefer to run dsc without specifying configuration and certificate files on command line options.

## Testing

Since the GotoCamera proxy programs communicate with the camera’s built-in services and GotoCamera server over IP connections, they can function properly regardless whether they run inside the camera as part of firmware or outside the camera on another machine connected to the same network. Let’s say your GotoCamera proxy program has the following configuration when running on IP camera firmware,

<setting>

<enableupnp> no </enableupnp>

<controllerid> </controllerid>

<adminport> 20000 </adminport>

<smtpport> 2500 </smtpport>

<devkey> </devkey>

<username> joe@gmail.com </username>

<password> abc </password>

…

<devices>

<device>

<name>home</name>

<description>My AirLink MJPEG IP Camera</description>

<address>type=tcp;host=127.0.0.1;port=81</address>

<id>type=airlink;subtype=generic;sysid=000000000000</id>

<credentials>

<username>admin</username>

<password></password>

</credentials>

</device>

</devices>

</setting>

Let’s assume the camera’s IP address is 10.0.0.5 and there is another Ubuntu Desktop at 10.0.0.6 on the same network. We can compile the dsc program on Ubuntu and run it using the same configuration file with the following changes:

<address>type=tcp;host=**10.0.0.5**;port=81</address>

<id>type=airlink;subtype=generic;sysid=**0023D302407C** </id>

where <host> should be changed to the camera’s IP and <sysid> to be the actual Ethernet address of the camera as dsc is no longer running on the same host as the camera. Furthermore, adding 10.0.0.6:2500 to the camera’s list of email targets for motion detection would enable GotoCamera motion detection notification service for the camera.

Since there are many standard utilities available on a desktop system, you can run dsc under GDB, your profiler of choice, or start a tcpdump session to monitor dsc’s network behavior. We strongly encourage you to test the GotoCamera programs on a standard Linux box first before moving them to firmware.

## Final Integration

First compile the program for your target firmware platform, assuming you made the following changes to config.h to be in line with your deployment strategy,

#define DEFAULT\_CONFIG\_FILE "/etc/gotocamera/gotocamera.conf"

#define SSL\_CERTIFICATE "/etc/gotocamera/gotocamera.pem"

and the final binary is placed on your firmware filesystem as,

/usr/bin/dsc.linux

create the symbolic links as,

/usr/bin/dsc

/usr/bin/dscmon

/usr/bin/dscadm

so that,

$ cd /usr/bin; ls –l

…

lrwxr-xr-x 1 joe users 7 May 8 08:22 dsc@ -> dsc.linux

lrwxr-xr-x 1 joe users 7 May 8 08:22 dscmon@ -> dsc.linux

lrwxr-xr-x 1 joe users 7 May 8 08:22 dscadm@ -> dsc.linux

…

Put $SRC\_ROOT/3rd\_party\_clients/src/hc.pem on the firmware as,

/etc/gotocamera/gotocamera.pem

Put a configuration on the firmware as,

/etc/gotocamera/gotocamera.conf

The file should look like the sample configuration file listed in section Configuration and go through the following checklist,

1. The storage media that the file is on must be **writable** and **non-volatile**, in other words, the Proxy program must be able to update the file and the changes must survive camera restart.
2. <controllerid> must be empty. This is for dsc internal use.
3. <username> must be empty. To be setup by user.
4. <password> must be empty. To be setup by user.
5. <devices> must contain exactly one <device> entry that represents the camera to which the firmware belongs.
6. <name> of <device> must be empty. To be setup by user.
7. <description> of <device> must be empty. To be setup by user.
8. The port attribute of <address> must match the default port of the camera’s HTTP server.
9. The <username> and <password> of <credentials> must match the default username and password of the camera’s HTTP server administrator account.

In addition, put an extra copy of the file on firmware as:

/etc/gotocamera/gotocamera\_default.conf

This file can be used to restore Proxy’s configuration to factory default in case gotocamera.conf gets corrupted.

The following changes need to be made to camera’s configuration logic,

1. Add new configurations GotoCamera *username* and *password*. Whenever they need to be displayed, invoke the following to retrieve their values:  
    /usr/bin/dscadm –t json get sys  
   Whenever these values are changed (both must be non-empty), invoke the following to update their values:  
    /usr/bin/dscadm –t json set sys un=… pw=…
2. Add new configurations *camera name* and *camera description*. Whenever they need to be displayed, invoke the following to retrieve their values:  
    /usr/bin/dscadm –t json get cam  
   Whenever they are changed (camera description is optional and thus can be empty), invoke the following command to update their values:  
    /usr/bin/dscadm –t json set cam name=… desc=…
3. If user is allowed to change camera’s HTTP service port, invoke the following command whenever a change has been made:  
    /usr/bin/dscadm –t json set cam addr=type=tcp\;host=127.0.0.1\;port=…
4. Whenever the user changes administrator username or password for the camera, invoke the following command:  
    /usr/bin/dscadm –t json set cam cred=un=…\;pw=…
5. If motion detection triggered email should be integrated with GotoCamera, add a new checkbox option *Enable Motion Detection for GotoCamera*. When user selects this option, add 127.0.0.1:2500 to the list of target email servers for motion detection. And when unselected, remove it from list.
6. Similarly, if motion detection should be integrated with GotoCamera through the Administrator program, add a new checkbox option *Enable Motion Detection for GotoCamera*. When this option is selected, whenever motion is detected, an image should be captured and saved to local file system and then invoke the following command to send it to GotoCamera:  
    /usr/bin/dscadm -f json notify md cam stat=1 delete=yes file=…  
   Make sure that
   1. A unique filename is generated for every image to avoid old files from being overwritten by new ones
   2. Files are deleted by dscadm instead of camera’s internal logic since only dscadm has the knowledge of when the file is no longer required.
7. Add a fields to display GotoCamera client version/build, which can be retrieved by  
   /usr/bin/dscadm –f json get sys
8. Add a field to display GotoCamera device key, which can be retrieved by:  
   /usr/bin/dscadm –f json get sys
9. Add a field to display GotoCamera proxy status, which can be retrieved by:  
   /usr/bin/dscadm –f json get sys
10. Add a **Reset** button to restore GotoCamera’s setting to factory default. When clicked, the button triggers the following command to run:  
     /usr/bin/dscadm -f json reset /etc/gotocamera/gotocamera\_default.conf
11. Add a **Restart** button to restart GotoCamera proxy. When clicked, the button triggers the following command to run:  
     /usr/bin/dscadm -f json restart
12. Add a **Enable Remote Setup** button to activate temporary remote setup. When clicked, the button triggers the following command to run:  
     /usr/bin/dscadm -f json rconf

Finally, update the firmware’s startup logic so that after network has been brought up, run the following,

/usr/bin/dscmon /usr/bin/dsc –o /tmp/dsc.log –b 128 –n 2

Adjust dsc options accordingly based on your requirement.