GX Reference Manual

v. 7.2.10

Document version: 2014.10.10

Table of Contents

About this manual 2
The GX System 3
    How to start? 3
System description 4
Levels of the GX system 5
GX on Microsoft Windows operating systems 6
    Directory structure after installation 6
    Installation 6
GX on GNU/Linux systems 7
    Directory structure after installation 7
    Installation 8
The Application Programming Interface (API) of the GX System 9
Error handling and exceptions 10
    General error codes 10
    Module error codes 11
    Error messages 11
    Error handling in the GX system 11
Reference handling 12
Memory handling 13
GX system variants 14
Module handling 15
    Description of module handling 15
    Opening a module 15
    Module properties 15
    Description for module handling functions 16
    Internals 16
    Properties 16
    Module handling classes 16
Compression handling 17
    Module properties 17
Image handling 18
        Module properties 19
        Properties of the 'gxbmp' submodule 19
        Properties of the 'gxjpeg8' and 'gxjpeg12' submodules 19
        Properties of the 'gxjpeg2k' submodule 22
        Properties of the 'gxwsq' module 25
Logging 25
    Module properties 25
Property handling 28
    Structure of the GXSD.DAT file 29
Transformation handling 31
Watchdog handling 31
Motion detection 32
    Module properties 35
Appendices 36
    Module properties 36
    Abbreviations 38
    FAQ - Frequently Asked Questions 39
Contact Information 41

About this manual

This manual contains information related to GX based products.

Since GX is the base system for the majority of the ARH. Inc. products, some sections in this manual contain product specific information (e.g. the motion detection module is used only for ANPR purposes). Consequently, some expressions and abbreviations can be unfamiliar to customers since these do not apply to the purchased product(s).

For better understanding of this manual, familiarity with the following expressions is necessary:

ANPR - Automatic Number Plate Recognition -
The process of identifying license plate characters from still images without human intervention.

ACCR - Automatic Container Code Recognition -
The process of identifying characters of container codes from still images without human intervention.

FXVD4
Frame grabber card of the ARH Inc. used for ANPR, ACCR etc. purposes.

NNC - Neural Network Controller -
Co-hardware for the ANPR, ACCR etc. software.

PR
Passport **R**eader is a GX based travel document reader with analyzer software by ARH Inc**.**

Beside the above-mentioned products, ARH Inc. has other products as well. For the full product list, please contact ARH Inc. Support Team.

The GX System

7.2.10

The **GX** is the base system for the majority of the **ARH Inc.** products. It is a collection of loadable modules and library functions and it provides an easy to program interface to the hardware devices.

The product's functionalities are implemented using specific **GX modules** that can be accessed through the **gx kernel module (hereafter: gxsd kernel module)**. Apart from the GX module handling functions, the gxsd kernel module has general functions for error handling, property handling, memory handling and image handling.

How to start?

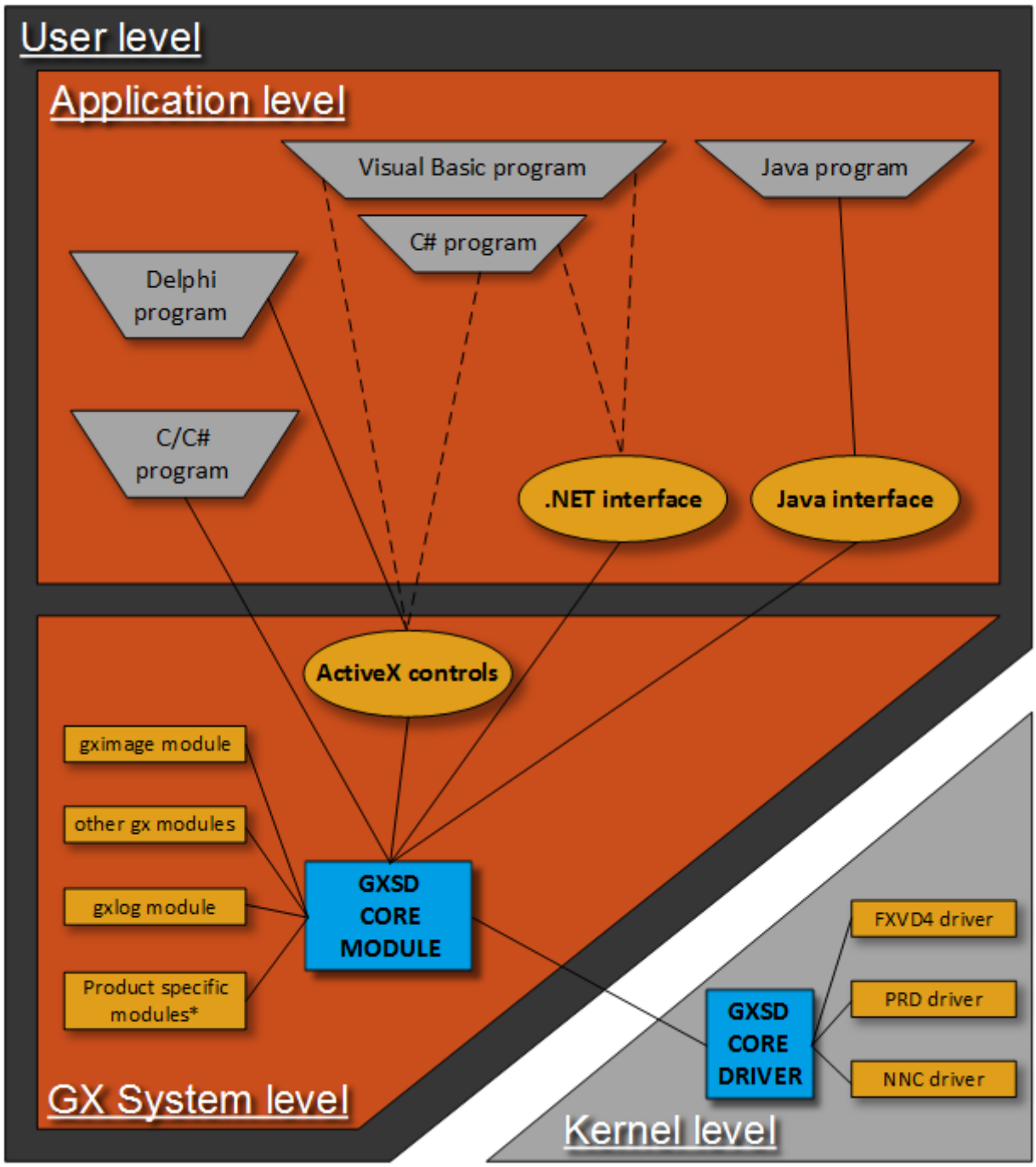
1. Please read the API documentation
2. Compile and try the example programs
3. Read FAQ pages

System description

The GX system is a collection of system files, shared (dynamically linked) objects, interface files for several programming languages, sample programs, utilities and documentation files. These files are arranged properly on your operating system by the installer supplied with the software package.

IMPORTANT: Before installation read carefully chapter 'GX on GNU/Linux systems' or 'GX on Microsoft Windows operating systems'.

EXTRACT FROM THE GX SYSTEM



* E.g. FXVD4, PRAPI, CLR, CMANPR, CPD etc.

Levels of the GX system

a) Kernel level

- *GXSD drivers (kernel modules):*
  This level consists of procedures belonging to various hardware devices or other system level processes. User level programs can access to the GX kernel modules only through user level GX modules.

- *GXSD core driver:*
  The user level modules communicate with the devices through device driver files. One file is assigned to each device driver. Identical devices are handled by the same device driver.

b) GX system level

- *GX system level modules:*
  Precompiled binaries and implemented function libraries, used by other GX based modules or user applications, belong to this level. These libraries are necessary for running GX based programs and for developing new applications. The user programs and applications can access to the kernel drivers and to the kernel module services only through user level GX modules. The modules have properties that can be altered by internal functions or by changing the settings in the *gxsd.dat* property file.

  Each GX module can be accessed through a common interface, which is described in the gxsd7 library. This function library is a collection of functions that constitute the core of the GX modules (**hereafter: GXSD core module**). It is the gateway for the other user level modules, as it implements the module handling functions (open, close, function call, etc.), the memory management functions (e.g. global memory allocation and deallocation), the property handling functions (load, save, etc.) and the error handling functions (e.g. get, set).

- *GX user level services/daemons:*
  User level programs running in the background belong to this level (e.g. a web server).

- *GX shared data files:*
  The GX system has a main property file named gxsd.dat. It contains the system's default properties. It is a simple XML file, which can be edited even with an ordinary text editor.

c) Application level

This is the highest level of the GX system. Applications written by users, sample programs and applications for the modules belong to this group. These applications use the GX system level modules.

From version 7.2 both 32 and 64 bit architectures are fully supported. The installer handles the convention differences between operating systems and architectures. Differences between installations on different operating systems and platforms are discussed in the next chapter.

The following operating systems are supported:

- Microsoft Windows systems
- GNU/Linux systems

GX on Microsoft Windows operating systems

Currently supported Microsoft operating systems:

- Windows XP
- Windows 2003 Server
- Windows 2008 Server
- Windows Vista
- Windows 7

Directory structure after installation

- **device drivers:** *%SystemRoot%\system32\drivers*
  The **gx core** driver (gxsd.sys) is the main driver of the GX system. It is exclusively used by the **gx core** module (gxsd7.dll). It handles the GX compatible devices in the GX system.
- **shared data files:** *%SystemRoot%\system32\gx*
- **user level modules:** *%SystemRoot%\system32\gx*
  The only exception is the gxsd7.dll which is located in the %SystemRoot%\system32 directory.
- **documentation:** *%ProgramFiles%\gx\docs*
- **interface files:** *%ProgramFiles%\gx\sdk*
- **sample programs:** *%ProgramFiles%\gx\sdk*

Installation

The GX system is delivered in one or more compressed packages with a setup program that executes the proper installation.

NOTE: Update your Windows operating system before installation. Older Windows systems may not compatible with certain setup components. For more information see “User’s FAQ”.

GX on GNU/Linux systems

Directory structure after installation

NOTE: The following text describes the default post-installation directory structure of the GX, it can be altered. For more information see the “environment variables” section in the readme.txt.

- **kernel modules:** */lib/modules/KERNEL_VERSION/extra*
  The kernel is the core of a GNU/Linux system. File systems and various hardware devices can be accessed through this interface. It is responsible for the timing of processes, the communication between them and for the distribution of system resources.

  For proper installation of the GNU C/C++ compiler, the attached software development kit and (the skeleton of) the kernel source is required. In some distributions, the kernel binary (executable, compiled modules), header and source files are in different packages. In this case, the relevant packages must be installed. The configuration of the kernel source is absolutely necessary.

  If your distribution does not contain the kernel source, please download the appropriate one that matches with the version of the target system's from: http://www.kernel.org.

  IMPORTANT: The main GX kernel module (the *gxsd* kernel module) - must be compiled and loaded before loading any other GX kernel module.

  The **gxsd** kernel module defines a kernel interface that is necessary for the operation of the GX system (e.g. communication between drivers and processes). All the other GX kernel modules access to the kernel services through the *gxsd* kernel module.

  When compiling the GX kernel modules, follow the installation instructions. It can occur that the kernel settings and patches are not compatible with the GX precompiled modules and with the whole GX system. In some special cases (e.g. in case of a new or modified Linux kernel) the modules might not compile. In this case, please note down the kernel version, the distribution and the cause of the error before contacting the support service.

- **device driver files:** */proc/gx*
  These files are used by the *gxsd* kernel module and by the *gxsddrv* kernel driver. GX kernel drivers create status files into this directory. With the help of these files the user can query a module's actual status (e.g. the /proc/gx/fxvd4 status file contains the status of the frame grabber card - whether there is a video signal or not, the number of the captured images etc.)

- **read-only shared data files:** */usr/share/gx*
  e.g. ANPR data files

- **shared modules:** */usr/lib/gx, /usr/lib64/gx*
  All modules can be found in these directories. The only exception is the libgxsd7.so which is located in the /usr/lib directory.

IMPORTANT: In those operating systems where the usr/lib and usr/lib64 is the same directory (either of them is a symbolic link to the other one) the use of the copy_all.sh script results in improper installation because during the installation the 32 bit files mix with the 64 bit ones. In case of these kinds of operating systems, (e.g. Ubuntu) the unpack_all.sh has to be used for unpacking and the necessary files (32 or 64-bit ones only) have to be copied manually.

- **documentation:** */usr/share/doc/gx*
  All modules' documentation can be found in this directory. The file formats of the documentations can be the following: plain text, HTML and PDF.

- **interface files:** */usr/include/gx*
  C, C++ header files can be found in this directory.

- **sample programs:** */usr/src/gx*
  Beside all sample programs (C,C++ and Java) this directory also contains:

  - Java interface files

  - utility programs (e.g. firmware update program)

  - the GX kernel source

  - sample files (e.g. sample images, sample config. files)

- **writable shared data files:** /var/gx

  - property settings of the GX modules (in gxsd.dat)

  - GX modules may also create files into this directory (e.g. global locks, named pipes).

Installation

The GX system is delivered in one or more compressed packages with an install script that helps the proper installation. For more help see readme.txt.

The Application Programming Interface (API) of the GX System

The Application Programming Interface (API) of the GX system provides an easy to use interface for both low level and high level programming languages.

The natively supported programming languages are:

- C/C++
- Java
- C#, VB.NET (Only on the GX supported Windows systems)

On Windows systems, the following ActiveX controls are available to use in other RAD type programming environments such as Visual Studio or Borland Delphi. (They also can be used in browsers, which support Java scripting):

- GxImage.OCX
- GxLabel.OCX
- GxMotdet.OCX
- GxVariant.OCX

Error handling and exceptions

The GX system gives the opportunity to find and handle run-time errors in user applications. The general error codes are defined in the gxerror header/interface file while the module specific error codes are described in the module header/interface files. Error codes below 0x1000 have the same function as the corresponding codes in the C programming language (e.g. ERRNO variable). The rest of them are the GX system's own error codes.

General error codes

Mnemo	Code	Description
GX_ENOERR	0x0000	No error
GX_ENOENT	0x0002	Entry not found (ENOENT)
GX_ENOMEM	0x000c	Memory allocation error (ENOMEM)
GX_EACCES	0x000d	Permission denied (EACCES)
GX_EFAULT	0x000e	Bad address or program error (EFAULT)
GX_EBUSY	0x0010	Resource busy (EBUSY)
GX_EEXIST	0x0011	File exists (EEXIST)
GX_ENODEV	0x0013	No such device (ENODEV)
GX_EINVAL	0x0016	Invalid parameter (EINVAL)
GX_ERANGE	0x0022	Data out of range (ERANGE)
GX_EDATA	0x003D	No data available (Linux - ENODATA)
GX_ECOMM	0x0046	Communication error on send (Linux - ECOMM)
GX_ETIMEDOUT	0x006E	Function timed out (Linux - ETIMEDOUT)
GX_EOPEN	0x1000	File open error
GX_ECREAT	0x1001	File creation error
GX_EREAD	0x1002	File read error
GX_EWRITE	0x1003	File write error
GX_EFILE	0x1004	Invalid file content
GX_EINVIMG	0x1010	Invalid image

GX_EINVFUNC	0x1011	Invalid function
GX_EHWKEY	0x1012	Hardware key does not work properly
GX_EVERSION	0x1013	Invalid version
GX_EASSERT	0x1014	Assertion occurred
GX_EDISCON	0x1015	Device is disconnected
GX_EIMGPROC	0x1016	Image processing failed
GX_EAUTH	0x1017	Authenticity cannot be determined
GX_ENOMODULE	0x8000	The specified module cannot be found

Module error codes

The module specific error codes have the following form:

GX_ENNNN = 0xMMMM8XXX where NNNN is the short name of the error, MMMM is the group code of the specified module and XXX is the error code. The module error codes are defined in the module header/interface.

Error messages

The error messages are registered in the GX system and stored in the main property tree under the *default/errors* node. During the installation of a new module, the new error codes and messages are registered in the GX system. The error messages may be translated into different languages freely.

Error handling in the GX system

In low level programming languages (C, C++) errors appear as returned values of functions. These returned values can be true or false. False is represented with 0 while true is represented with any kind of value but 0. To get the error code and message, the error handling functions have to be used.

In high level programming languages (for example in Java, C++ or C#), where the use of classes and exceptions are enabled, an exception will be thrown by the GX system whenever an error occurs. In this case, the thrown exception class contains the error's code and message.

In the C++ programming language the NO_GX_EXCEPTIONS define can be used to force the GX system not to throw exceptions and return “false” when an error occurs. In case of this kind of forcing, the error code and message is accessible using the error handling functions of the GX system.

Reference handling

For performance, reasons many of the GX system objects use reference counters that ease the allocation, handling and safe deletion in multithreaded environments.

Reference counters indicate the number of modules, threads or processes using the same object (e.g. an image) and the allocation type of the data (dynamic or static). Negative reference counter values show that the structure was dynamically allocated while positive reference counter values indicate that the structure was statically allocated.

NOTE: Except gxVARIANT.

Reference counters with static allocations

If a process wants to work on a previously allocated object then it references it (increments the value of the object's reference counter by one) and after finished using it, unreferences it (decrements the value of the object's ref. counter by one) to signal that it does not use the object anymore. If the object is not referenced by any other processes then the allocated memory is freed instantly.

So, in a static allocation, the value of an object's reference counter always shows the number of threads, modules or processes that are using it.

Reference counters with dynamic allocations

In a dynamic allocation the value of an object's reference counter is decremented by one when a process references it and it is increased by one when a process unreferences it.

So, in a dynamic allocation, the reference counter's absolute value gives the number of modules, threads or processes that using the object.

The GX classes do not use the static allocation mode (it is problematic in object-oriented languages because it can cause memory leakage) but use the dynamic allocation with automatic reference handling.

The static and dynamic objects must be initialized with different functions. In case of static objects, a static variable has to be declared and the gx_init...() function has to be used while in case of dynamic objects, a pointer has to be created and the gx_alloc...() function has to be used for initialization.

IMPORTANT: The gx_init and gx_alloc functions have to be used with the appropriate allocations. Using these functions with inappropriate allocations leads to program error.

Static example:

```
gxIMAGE image;
gx_initimage(&image);
...
```

Dynamic example:

```
gxIMAGE *pimage;
gx_allocimage(&pimage);
...
```

Examples:

```
gx_refimage(&image);
gx_unrefimage(&image);

gx_refimage(pimage);
gx_unrefimage(pimage);
```

In low-level multithreading applications, the usage of the reference increasing and reference decreasing functions is recommended.

NOTE: The protection of the allocated structures in multithreaded environments is the user's responsibility.

In case of using class implementation instead of global functions (e.g. gxImage), handling the reference counter is not needed because, in this case, the classes use dynamic allocation and handle the reference counter automatically.

Memory handling

The GX system has several low level memory handling functions. In low level programming languages, the use of these functions is recommended if the user application uses GX module(s) and allocates or calls functions that allocate memory space(s).

In high level programming languages the GX classes automatically perform the memory handling tasks, memory handling functions are not available in those languages.

GX system variants

The GX system variant is a variant type, constructed directly for GX based systems to hide the complexity of handling the native variant types in different programming languages and to give a common way to hold and to process data.

The GX based systems use this new data type for storing and handling internal (cannot be accessed by the user) or external (can be accessed by the user) data in the modules.

The gxVariant has the following important features:

- Uses run-time type information for automatic data conversion on user request. (type)
- Stores id, which can be used to identify data in a complex structure. (id)
- Stores children. The variants can be connected together in a way that they form a tree through parent-child relationships. (nchildren), (children)
- Uses reference counting to save memory and to form a closed tree. (ref)[1]
- It can be a list, in which the items are connected according to parent-children hierarchy. The list and list items can have children too.

Beside the above-mentioned attributes in brackets (e.g: type) the following ones can also be queried only by variant manipulating functions.

- size - size of the variant data
- data - the data stored in the variant type

[1] In case of the gxVariant the reference counter is always positive in spite of the fact that it can be allocated only dynamically.

Module handling

Description of module handling

The user applications reach the GX modules through a defined interface. In non-object-orientated languages, the modules must be opened before use. In the opening process, the system allocates a handler (called *handle*) for the module. From this point on, this handle is used for communication with the module. In object-oriented languages, this handle is stored as an internal variable, when using inline functions - defined in the module headers - this handle must be supplied as a function parameter.

Modules of the GX system:

- Compression handling
- Image handling
- Logging
- Property handling
- Transformation handling
- Watchdog handling
- Motion detection

Opening a module

Opening a module is different in plain C and in object-oriented languages. The name of the module must be supplied first as a parameter for the module opening function.

- On GNU/Linux systems, the modules can be opened by simply giving the base name of them). No path, nor extension nor "lib" prefix specification is needed.
- On Windows systems, searching for the module starts with a standard DLL searching sequence, described at the LoadLibrary function in the MSDN and ends with the %SystemRoot%\system32\gx directory. The explicit specification of the directory path to the module is only needed, when the module is placed in a non-standard location.

In object oriented languages when instantiating an object from a GX class, the required module automatically opens.

Module properties

Most of the GX modules have properties. The properties can be read only, readable and writable or write only data that can:

- reflect the values of internal variables
- control the module functioning
- give information about the state of the module

The properties are specific in every module so the properties of a module are independent from the properties of other modules.

The properties have default values that can be overwritten in the main property file called gxsd.dat. It is a system wide unique XML file that has a *default* branch and other, user definable branches. The properties of a module are grouped by a node that has the same name as the module. A node can have one or more property entries or subnodes that can have other property entries or subnodes too. On module starting, the values specified for the module properties in the .dat file overwrite the default values. These values can be changed or saved by using property handling functions.

The branch from which the property values are read can be specified on module opening. If 'default' or none is specified then the default branch is used.

For more information on properties, see "Property handling" section.

Description for module handling functions

Though programming in C++, the low-level C functions are also available, in this case, the use of the implemented classes is recommended instead of the inline functions.

The GX SDK mainly targets the object-oriented programming languages although there is support for low level C programming. From this point on, only the class methods will be mentioned in examples or descriptions, not forgetting that each of them has an inline function pair.

Internals

Each module function call is in fact a gx_call with the right arguments set. The gxsd core module takes care of redirecting function calls to the proper modules. For more information on low-level programming, check the C header files in the SDK.

Properties

The property setting functions set property values only for opened handles; they have only runtime effect on the properties.

Module handling classes

In object oriented programming languages instantiating an object creates an internal handler and opens the module related to it. For memory saving concerns, in some cases different handle and data classes are created.

For example, the image handling routines can use a single instance of the gxImageHandler and one or more instances of the gxImageData class. In this way, less resource is allocated then using the gxImage class.

The handler class also handles the properties and the changes are global to all data classes. For detailed information, see the gxHandle class. The other GX classes are inherited from this base class.

Compression handling

The compression module of the **GX** system is designed to compress data and to decompress compressed data. It is based on the ZLIB package (http://www.zlib.org) and MiniZIP package. It is capable of handling both ZIP and GZ formats.

The ZLIB is a general-purpose data compression library. It is the work of Jean-loup Gailly, Mark Adler and others. The MiniZIP package is written by Gilles Vollant. For more information, see http://www.zlib.org or http://www.gzip.org.

The data format used by the zlib library is described in RFC (Request for Comments) 1950, 1951 and 1952. These standards can be found at http://www.ietf.org/rfc/rfc1950.txt (zlib format), http://www.ietf.org/rfc/rfc1951.txt (deflate format) and http://www.ietf.org/rfc/rfc1952.txt (gzip format).

Functionalities:

- compressing/decompressing data in memory
- reading from and writing to files in GZip format (.gz)
- reading from and writing to files in ZIP format (.zip)
- handling ZIP files with comments and passwords

Module properties

This module does not use properties.

Image handling

The gximage module provides an easy to use image-manipulating library.

The GX system implements functions for:

- loading/saving images (from/to file or memory)
- converting images into different image formats
- mirroring an image (horizontal and/or vertical)
- rotating an image (by 0, 90, 180 and 270 degrees)
- handling 8/12 bit grayscale and 8/12 bit RGB/ BGR images
- zooming in/out an image (nearest neighborhood and linear interpolation algorithms)

The supported image formats are:

- BMP : stores images using lossless or no compression. Supported BMP formats: MS Windows 3.0 BMP file format handling: 2 colors, 16 colors (RLE4 and no compression), 256 colors (RLE8 and no compression), 24 bits per pixel (RGB without compression) and read-only 16 and 32 bits per pixel modes.

- JPEG : stores images using lossy compression. The gxjpeg module uses the IJG JPEG library. (http://www.jpeg.org) The quality of the image can be set on a scale from 0 to 100, where 0 stands for the worst and 100 stands for the best quality. The GX system handles 8/12 bit grayscale and 8/12 bit (per color components) RGB / BGR images. Small messages/data (markers) can be included in the JPEG images. Every message has a type set by the application that created it. (e.g. these kinds of markers are included in digital cameras to store information about lighting) The universally used marker code is GX_JPEGMARKER_COM. Other applications work in the range of GX_JPEGMARKER_APP0 to GX_JPEGMARKER_APP15. More than one marker with the same type can be placed in an image.

- JPEG-2000: stores pictures using lossless or lossy compression. The codec of the JPEG-2000 is based on wavelet/subband coding techniques. It handles both lossy and lossless compression of single-component and multicomponent image representations.

- The gxjpeg2k module uses the JasPer (http://www.jpeg.org) library and handles RGB and YCbCr color space with unsigned components. The module can write and read files in JP2 (JPEG-2000 file format) and JPC (JPEG-2000 code stream) formats. You can use it as an input and output method for the GX image with arbitrary pixel format.

- If you prefer the BGR order to RGB and you use the JPC format then the components of the image will be stored in reversed order. As the JPEG-2000 standard does not specify any means for encoding color space information in a JPEG-2000 code stream (JPC), the decoder has to make certain assumptions about the color space of an image. If accurate color representation is important, the use of the JPEG-2000 code stream (JPC) format is not advisable. The JPEG-2000 JP2 file format is recommended instead of it.

- PNG : Stores images using lossless compression. The gxpng module uses the libpng (http://www.libpng.org) library. PNG provides a patent-free replacement for GIF and can also replace many common uses of TIFF. Indexed-color, grayscale and true color images are supported, plus an optional alpha channel.

- WSQ : Stores grayscale images using a lossy compression. This format is developed to store fingerprint images. The gxwsq module uses the Free FingerPrint Imaging Software Math Library (http://ffpis.sourceforge.net) library.

Module properties

zoom mode - Zoom type (integer)

```
<default>
   <zoom mode value="0"/>
</default>
```

It can be:

- 0 : the zoom function uses the nearest neighborhood algorithm (faster than the linear interpolation algorithm)
- 1 : the zoom function uses the linear interpolation algorithm (better than nearest neighborhood algorithm)

Properties of the 'gxbmp' submodule

The *gxbmp* module properties are located under the *bmp* node in the property tree.

```
<default>
   <bmp>
      <force24bpp value="0"/>
      <enablerle value="0"/>
</default>
```

bmp/force24bpp - forces the 24 bit per pixel mode

It can be:

- 0 : the indexed mode is used
- 1 : the function uses 24 bit per pixel (RGB) mode

bmp/enablerle - enables the RLE compression

It can be:

- 0 : disables the RLE compression
- 1 : enables the RLE compression

Properties of the 'gxjpeg8' and 'gxjpeg12' submodules

The GX system has two JPEG submodules:

- The *gxjpeg8* module handles the 8 bit (per color components) images.
- The *gxjpeg12* module handles the 12 bit (per color components) images.

The submodule properties are located under the *jpeg*, *jpeg8* or *jpeg12* nodes in the property tree.

If a property is not set in the jpeg8 or jpeg12 branches its value is set to the value specified in the jpeg branch.

```
<default>
   <jpeg>
      <decompress>
         <scale_num value="1"/>
         <scale_denom value="1"/>
         <dct_method value="0"/>
         <do_fancy_upsampling value="0"/>
         <do_block_smoothing value="0"/>
      <compress>
         <quality value="80"/>
         <progressive value="0"/>
         <dct_method value="0"/>
         <optimize_coding value="1"/>
         <restart_interval value="0"/>
         <restart_in_rows value="0"/>
         <smoothing_factor value="0"/>
   <jpeg8>
                ...
   <jpeg12>
                ...
</default>
```

jpeg/decompress/scale_num - image scaling

According to the JPEG library documentation:

> 'Scale the image by the fraction scale_num/scale_denom. Default is 1/1, or no scaling. Currently, the only supported scaling ratios are 1/1. 1/2. 1/4 and 1/8. Smaller scaling ratios permit significantly faster decoding since fewer pixels need be processed and a simpler IDCT method can be used.'

jpeg/decompress/scale_denom - image scaling

See also:

jpeg/decompress/scale_num - image scaling

jpeg/decompress/dct_method - DCT method

Selects the algorithm used for the DCT step.

It can be:

- 0 : default method (JDCT_DEFAULT)
- 1 : slow but accurate integer algorithm (JDCT_ISLOW)
- 2 : faster, less accurate integer algorithm (JDCT_IFAST)
- 3 : floating-point method (JDCT_FLOAT)

jpeg/decompress/do_fancy_upsampling - image sampling

It can be:

- 0 : A faster but less accurate method is used.
- 1 : More careful upsampling of the chromatic components.

The visible difference between the more careful and the faster method is often insignificant.

jpeg/decompress/do_block_smoothing - image smoothing

Interblock smoothing is applied or disabled.

It can be:

- 0 : Disables interblock smoothing.
- 1 : Interblock smoothing is applied in the early stages of decoding progressive JPEG files.

Early progression stages look 'fuzzy' with smoothing, 'blocky' without it. In any case, block smoothing ceases to be applied after the first few AC coefficients are known to full accuracy, so it is relevant only when using buffered-image mode for progressive images.

jpeg/compress/quality - compression quality

The quality of a JPEG image. 0 stands for the worst quality, 100 stands for the best. The default value is 80 in the GX system and 75 in the original JPEG library.

jpeg/compress/progressive - compression mode

It enables or disables creating progressive JPEG files.

- 0 : disables progressive JPEG
- 1 : enables progressive JPEG

jpeg/compress/dct_method - DCT method

Selects the algorithm used for the DCT step.

It can be:

- 0 : default method (JDCT_DEFAULT)
- 1 : slow but accurate integer algorithm (JDCT_ISLOW)
- 2 : faster, less accurate integer algorithm (JDCT_IFAST)
- 3 : floating-point method (JDCT_FLOAT)

jpeg/compress/optimize_coding - code optimization

If the value is true then the compressor generates optimal Huffman coding tables for the image. It requires an extra pass over the data so, it costs a good amount of space and time:

- 0 : the compressor uses the supplied or default Huffman tables
- 1 : the compressor computes optimal Huffman coding tables

jpeg/compress/restart_interval - restarting interval

According to the JPEG documentation:

> 'To emit restart markers in the JPEG file, set one of these nonzero. Set *restart_interval* to specify the exact interval in MCU blocks. Set *restart_in_rows* to specify the interval in MCU rows. (If *restart_in_rows* is not 0, then *restart_interval* is set after the image width in MCUs is computed.) Defaults are zero (no restarts). One restart marker per MCU row is often a good choice.'
>
> **NOTE:** "The overhead of restart markers is higher in grayscale JPEG files than in color files, and much higher in progressive JPEGs. If you use restarts, you may want to use larger intervals in those cases"

jpeg/compress/restart_in_rows - restarting interval in rows

See also:

jpeg/compress/restart_interval - restarting interval

jpeg/compress/smoothing_factor - compression smoothing factor

If non-zero, the input image is smoothed; the value varies between 1 and 100 (minimal and maximal smoothing).

Properties of the 'gxjpeg2k' submodule

The property section of the module is *jpeg2k* in the property tree.

```
<default>
   <jpeg2k>
      <imgareatlx value="-1"/>
      <imgareatly value="-1"/>
      <tilegrdtlx value="-1"/>
      <tilegrdtly value="-1"/>
      <tilewidth value="-1"/>
      <tileheight value="-1"/>
      <prcwidth value="32768"/>
      <prcheight value="32768"/>
      <cblkwidth value="64"/>
      <cblkheight value="64"/>
      <mode value="int"/>
      <rate value="-1"/>
      <ilyrrates value=""/>
      <prg value="0"/>
      <nomct value="0"/>
      <numrlvls value="6"/>
      <sop value="0"/>
      <eph value="0"/>
      <lazy value="0"/>
      <termall value="0"/>
      <segsym value="0"/>
      <vcausal value="0"/>
      <pterm value="0"/>
      <resetprob value="0"/>
      <numgbits value="2"/>
</default>
```

jpeg2k/imgareatlx - X-coordinate of the image area.

Sets the x-coordinate of the top-left corner in the image area. If you use a negative value, the property will be ignored.

jpeg2k/imgareatly - Y-coordinate of the image area.

Sets the y-coordinate of the top-left corner in the image area. If you use a negative value, the property will be ignored.

jpeg2k/tilegrdtlx - X-coordinate of the tiling grid.

Sets the x-coordinate of the tiling grid's top-left corner. If you use a negative value, the property will be ignored.

jpeg2k/tilegrdtly - Y-coordinate of the tiling grid.

Sets the y-coordinate of the tiling grid's top-left corner. If you use a negative value, the property will be ignored.

jpeg2k/tilewidth - The nominal tile width.

Sets the nominal tile width. If you use a negative value, the property will be ignored.

jpeg2k/tileheight - The nominal tile height.

Sets the nominal tile height. If you use a negative value, the property will be ignored.

jpeg2k/prcwidth - The precinct width.

Sets the precinct width. The property must be an integer power of two. The default value is 32768. If you use a negative value, the property will be ignored.

jpeg2k/prcheight - The precinct height.

Sets the precinct height. The property must be an integer power of two. The default value is 32768. If you use a negative value, the property will be ignored.

jpeg2k/cblkwidth - The nominal code block width.

Sets the nominal code block width. The property must be an integer power of two. The default value is 64. If you use a negative value, the property will be ignored.

jpeg2k/cblkheight - The nominal code block height.

Sets the nominal code block height. The property must be an integer power of two. The default value is 64. If you use a negative value, the property will be ignored.

jpeg2k/mode - The coding mode.

Sets the coding mode. This property must have one of the following values:

- int,0: integer mode
- real,1: real mode

In case of lossless coding, the integer mode has to be used. By default, the integer mode is employed. The choice of mode also determines which multicomponent and wavelet transforms (if any) are used.

jpeg2k/rate - The target rate.

Specifies the target rate. The property is a positive real number. As the rate of one corresponds to no compression, explicit specification of a rate grater than one is never needed. By default, the target rate is considered to be infinite.

jpeg2k/ilyrrates - The rates for any intermediate layers.

Specifies the rates for any intermediate layers. The ilyrrates property to this option is a comma separated list of N rates. Each rate is a positive real number. The rates must increase monotonically. The last rate in the list should be less than or equal to the overall rate (as specified with the rate property).

jpeg2k/prg - Progression order.

Sets the progression order. This property must have one of the following values:

- lrcp,0: layer-resolution-component-position progressive (e.g. adjustable rate)
- rlcp,1: resolution-layer-component-position progressive (e.g. adjustable resolution))
- rpcl,2: resolution-position-component-layer progressive
- pcrl,3: position-component-resolution-layer progressive
- cprl,4: component-position-resolution-layer progressive

By default, LRCP progressive ordering is employed. Note that the RPCL and PCRL progressions are not valid for all possible image geometries.

jpeg2k/nomct - Disallows the use of any multicomponent transform.

Set to 1 to disallow the use of any multicomponent transform.

jpeg2k/numrlvls - Number of resolution levels.

Sets the number of resolution levels. It must be an integer that is greater than or equal to one. The default value is 6.

jpeg2k/sop - Generates SOP marker segments.

Set to 1 to generate SOP marker segments.

jpeg2k/eph - Generates EPH marker segments.

Set to 1 to generate EPH marker segments.

jpeg2k/lazy - Enables lazy coding mode.

Set to 1 to enable lazy coding mode (a.k.a. arithmetic coding bypass).

jpeg2k/termall - Terminates all coding passes.

Set to 1 to terminate all coding passes.

jpeg2k/segsym - Uses segmentation symbols.

Set to 1 to use segmentation symbols.

jpeg2k/vcausal - Uses vertically stripe causal contexts.

Set to 1 to use vertically stripe causal contexts.

jpeg2k/pterm - Uses predictable termination.

Set to 1 to use predictable termination.

jpeg2k/resetprob - Resets the probability models.

Set to 1 to reset the probability models after each coding pass.

jpeg2k/numgbits - Number of guard bits.

Set the number of guard bits (1 to 8). Default value is 2.

Properties of the 'gxwsq' module

wsq/bitrate – bitrate parameter of the wsq algorithm

Suggested values: 0.75 – 3.0

Logging

The gxlog module helps to use and to create log files for your GX based applications.

Module properties

```
   <default>
      <log>
         <ident value="my_application"/>
         <file value="mylogfile.txt"/>
         <format value="$y-$o-$d $h:$m:$s ($l:$L) [$i] {$F:$I:$P} > $M\n"/>
         <filter value="5"/>
         <levels>
            <L0 value="NONE"/>
            <L1 value="EMER"/>
            <L2 value="FATL"/>
            <L3 value="ERRO"/>
            <L4 value="WARN"/>
            <L5 value="INFO"/>
            <L6 value="INF2"/>
            <L7 value="INF3"/>
            <L8 value="DBG1"/>
            <L9 value="DBG2"/>
          </levels>

<maxfilesize value="10000"/>
           <maxfilenum value="5"/>
      </log>
   </default>
```

ident - identification string

Identification string of the application (e.g. 'MYAPPLICATION').

file - name of the log file

This property contains the name of the property file (e.g. 'mylog.txt'). If it is empty, the log module forwards the log messages to the system logger (*syslogd* on Linux and event logging system on Windows).

format - format string

You can define the line format of the log file. The logger reads it and replaces the specified arguments with the specified values.

The arguments can be:

- $y : year (4 digits)
- $r : year (2 digits)
- $o : month (2 digits)
- $d : day (2 digits)
- $h : hour (2 digits)
- $m : minute (2 digits)
- $s : second (2 digits)
- $z : milliseconds (3 digits)
- $l : level (number)
- $L : level (string)
- $i : identification (string)
- $F : name of the file (string)
- $I : line in the file (number)
- $P : name of the function (string)
- $M : message (string)
- $p : process id (8 digits hex number)
- $t : thread id (8 digits hex number)
- $$: one dollar character (1 character)

filter - message filter

Messages are arranged into levels according to their importance (see the example below). The message filter value determines the maximum level of messages to be written to the log file. If it is 0 (GX_NONE), it disables logging.

levels - strings for level codes

Every message level has a description string. The logger writes it into the log file when the $L is specified in the format string.

Example: Log levels.

```
<levels>
   <L0 value="NONE"/>
   <L1 value="EMER"/>
   <L2 value="FATL"/>
   <L3 value="ERRO"/>
   <L4 value="WARN"/>
   <L5 value="INFO"/>
   <L6 value="INF2"/>
   <L7 value="INF3"/>
   <L8 value="DBG1"/>
   <L9 value="DBG2"/>
</levels>
```

maxfilesize – maximal length of a log file

maxfilenum – maximum number of log files

The logger removes the last files when there are more files than the specified maxfilesize value.

Example:

```
logfile.txt,   logfile.txt.1, logfile.txt.2, logfile.txt.3
```

If the *maxfilenum* value is 2 then the logger removes the *logfile.txt.2* and *logfile.txt.3* files and creates a new *logfile.txt*.

Property handling

The GX modules and applications can create and read properties. This section describes the general handling of properties. The description of the properties of each module can be found in the related module sections, e.g. the properties of gximage can be found in the Image handling chapter.

The properties are arranged into a tree structure and consist of a name and a value, both of them are string data types.

Example of a property tree:

```
/
  default
     anpr
        contrast   1
        size      10
     fxvd4
        channel    1
        xsize    640
        ysize    480

  sample_application_1
     anpr
        contrast   1
     fxvd4
        xsize    768
        ysize    576

  sample_application_2
     anpr
        contrast   5
        size      20
     fxvd4
        channel    2
```

At the root of the property tree there is a default branch which contains module names and their property settings. Users can create new branches at the level of the default branch with custom property settings in it. When referring to these properties from user applications, it is enough to specify the name of the new branch because the modules always refer to the properties relevant for them (module *fxvd4* to *sample_application_1/fxvd4* or module *anpr* to *sample_application_1/anpr*).

Property values

There is a special branch at the root of the tree, called *default* branch that holds the default values of all properties. Only string, integer and floating-point data types can be used as property values.

If a property is not set in a specific branch its value is set to the value specified in the default branch. More, it is possible to define a parent branch for every branch, its property values are included in the branch in which the *parent* value is specified.

If the system cannot find the property in the branch or in one of its parents then loads its default value from the default branch.

Module properties

Module properties are arranged into a specific structure in the property tree. After opening a module, its properties are available for the user application for querying, changing or saving in the main property file. In the opening process, the main branch for locating the module properties can be specified. If nothing is set then the *default* branch is used. The property values are read into and stored in the memory.

The property values are connected to the module handlers so changing the properties of a module opened with a specific handle, does not affect the properties of the module opened with another handle. More, the changed property values are not saved on the mass storage device and the changes are discarded when the module is closed. To save these property values, the module property saving function has to be called.

The main property file

The main property file is the System32\gx\gxsd.dat file on Windows systems and the /var/gx/gxsd.dat file on Linux systems. In case of changing or adding a property, the file refresh generally takes 2-3 seconds but for some modules it is necessary to stop/close and to restart/reopen it.

Editing

The property file can be edited with an external text editor, e.g. Notepad or WordPad on Windows systems or mcedit, and vi on Linux systems only when no other applications are running in background that modify the property file.

The property parameters of the Open() functions and constructors specifies the location of the module's properties in the property tree. If the parameter is empty or “default”, the module works with default settings. The Open() function reads properties from this location and stores them in the memory. For more information see the Module handling section.

Structure of the GXSD.DAT file

The property entries can contain subproperties arranged into a tree structure. A property name can contain the letters of the English alphabet (a-z), numbers (0-9) and underscore (_). The value of the property has to be set according to the following format: *value="...*". The use of inverted commas is compulsory.

Description of a property without subproperties:

```
<property_name value="property_value"/>
```

Description of a property containing subproperties:

```
<property_name>
   <subproperty_name value="property_value"/>
</property_name>
```

or

```
<property_name value="property_value">
   <subproperty_name value="property_value"/>
</property_name>
```

Example:

```
<sample_application>
   <fxvd4>
      <xsize value="768"/>
      <ysize value="576"/>
   </fxvd4>
</sample_application>
```

The property entry can contain a parent link, which refers to a node in the property tree. If the system cannot find the property in the selected branch, the searching continues from the node specified by the parent link.

Example:

```
<default>
   <fxvd4>
      <ysize value="480"/>
   </fxvd4>
</default>

<sample_one>
   <fxvd4>
      <xsize value="768"/>
   </fxvd4>
</sample_one>

<sample_application>
   <fxvd4 parent="/sample_one"/>
</sample_application>
```

The value of */sample_application/fxvd4/xsize* becomes 768 and the value of */sample_application/fxvd4/ysize* becomes 480.

Transformation handling

The GX system supports transformations between different coordinate systems. By using these functions, it is easy to realize transformations from a plain rotation to the more difficult image straightening ones. Every transformation structure contains mappings needed for the transformation. These mappings describe the location of a coordinate system's specific point in another coordinate system.

Watchdog handling

An application may freeze or exit with an error. In these cases, it is particularly critical for the system to be able to restart itself in order to fix the error. This is the function of the watchdog device.

Microsoft Windows operating systems support hardware watchdog devices while the Linux system supports the use of hardware-, and software watchdog devices as well. More detailed description can be found in the Linux Kernel documentation (in watchdog.txt and watchdog-api.txt files).

The GX system implements the handling of the hardware watchdog devices connected to it. Such is, for example, the FXVD4 card.

The watchdog device has a timer, which is reseted periodically. If the reset does not happen and the timer reaches a previously set value then the watchdog device tries to reboot the system or application.

CAUTION: Connecting the reset connector of the FXVD4 card to the computer's reset wire reboots the whole system. This can cause data loss.

Motion detection

The motion detection module of the GX system is designed to detect changes between frames in a stream. For this procedure, images are reduced and divided into blocks.

The motion detector function executes edge detection and a motion analyzer algorithm calculates the differences between the current and the previous frame. As output, the algorithm locates a rectangle shaped area from the total still image where the changes (motion) have occurred.

The function detects a rectangle shaped area, which contains all the blocks with pixel differences over a specified limit. The returned parameters are the coordinates of the area and the minimal and maximal pixel differences calculated in the specified area.

On startup, passing the first still of the stream, the algorithm returns the "no difference" value (-1). In the next step, the correct value is returned.

IMPORTANT: Setting the relevant properties is obligatory.

How does the module work?

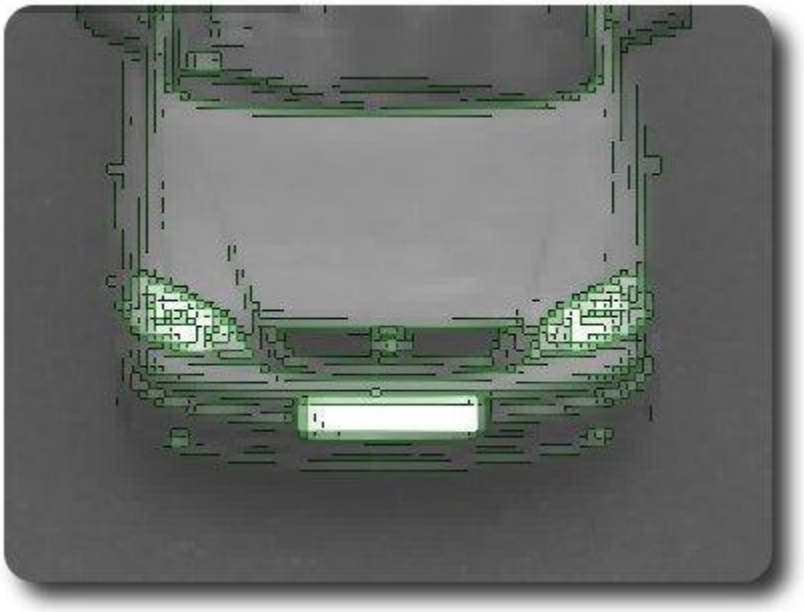
- The user adds a still to the motdet module. (xsize and ysize properties)



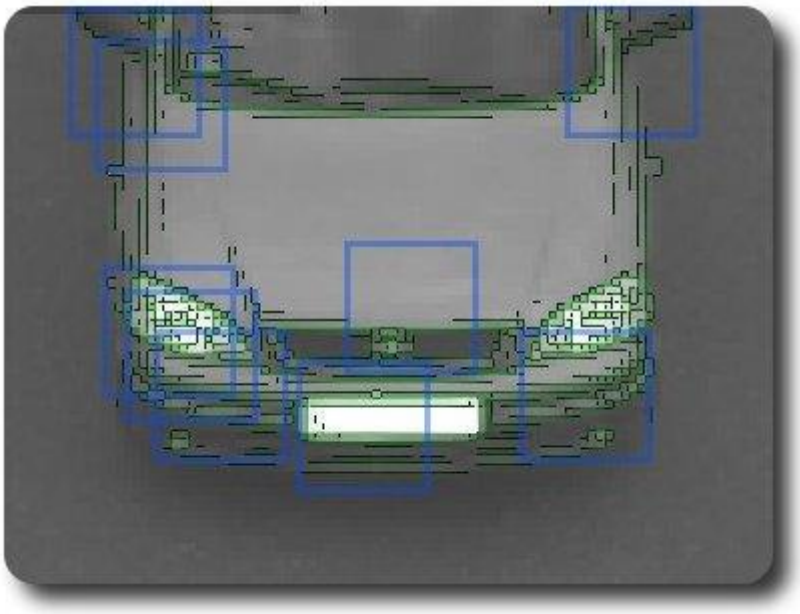
- The module reduces the image. (scale_x and scale_y properties)



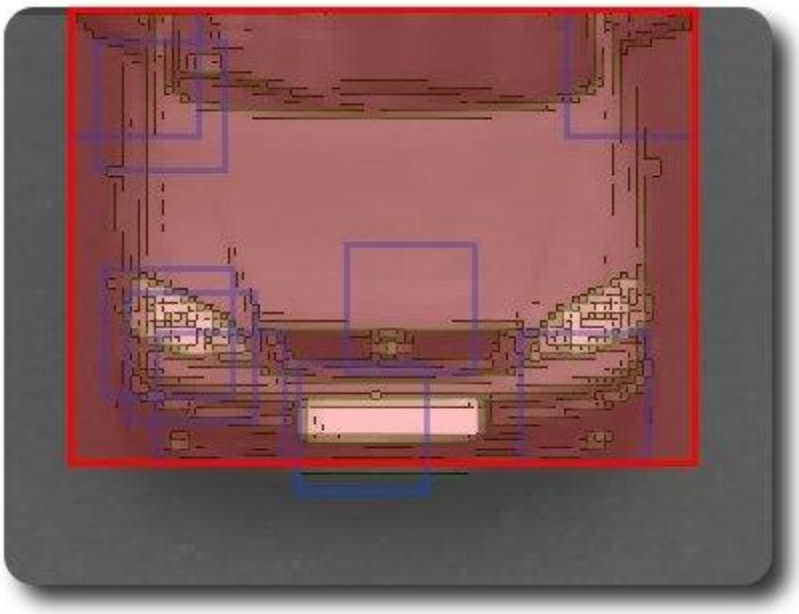
- The module executes the edge detection.



- The algorithm divides the image into blocks. (block_x and block_y properties)



- The algorithm calculates the pixel differences in each block and defines the difference area. If the pixel difference value of a block is greater than the sensibility value then the block is added to the area.



Module properties

```
<default>
   <motdet>
       <xsize value="768"/>
       <ysize value="576"/>
       <scale_x value="8"/>
       <scale_y value="8"/>
       <block_x value="8"/>
       <block_y value="8"/>
       <contrast_min value="8"/>
       <sensibility value="70"/>
   </motdet>
```

xsize - width of a still (integer)

This property has to be set to the width of the images. The algorithm does not work with images that have different sizes.

ysize - height of a still (integer)

This property has to be set to the height of the images. The algorithm does not work with images that have different sizes.

scale_x - reducing value in X orientation

It contains a reduced value in X orientation and must be a positive integer number. The default value is 8 (e.g. If an image's width is 640 pixels, the module reduces it to 80 pixels). Lower values results in preciser process but the algorithm will be slower.

scale_y - reducing value in Y orientation

It contains a reduced value in Y orientation and must be a positive integer number. The default value is 8 (e.g. If an image height is 480 pixels, the module reduces it to 60 pixels). Lower values results in results in preciser process but the algorithm will be slower.

block_x - width of a block

Width of a block in pixels. It must be less than the reduced width (xsize/scale_x). Lower values results in preciser process and quick speed but increase the probability of false hits.

block_y - height of a block

Height of a block in pixels. It must be less than the reduced height (ysize/scale_y). Lower values results in preciser process and quick speed but increase the probability of false hits.

contrast_min - minimal pixel contrast

It defines the minimal contrast between pixels at the same positions of the images. The algorithm does not differentiate two pixels if the grayscale difference is greater than or equal to the specified value. The default value is 8 (e.g. If the value is 4 it signifies 4 in the 8-bit grayscale image /256 values/).

sensibility - Sensibility

The sensibility of the algorithm. If the overall pixel difference of a block is greater than *sensibility*block_x*block_y then* the block is added to the critical areas.

Appendices

Module properties

- B -

- bmp/enablerle - Enables the RLE compression
- bmp/force24bpp - Forces the 24 bit per pixel mode

- J -

- jpeg/compress/dct_method - DCT method
- jpeg/compress/optimize_coding - Code optimization
- jpeg/compress/progressive - Compression mode
- jpeg/compress/quality - Compression quality
- jpeg/compress/restart_in_rows - Restarts interval in rows
- jpeg/compress/restart_interval - Restarts interval
- jpeg/compress/smoothing_factor - Compression smoothing factor
- jpeg/decompress/dct_method - DCT method
- jpeg/decompress/do_block_smoothing - Image smoothing
- jpeg/decompress/do_fancy_upsampling - Image sampling
- jpeg/decompress/scale_denom - Image scaling
- jpeg/decompress/scale_num - Image scaling

- jpeg2k/cblkheight - The nominal code block height.
- jpeg2k/cblkwidth - The nominal code block width.
- jpeg2k/eph - Generate EPH marker segments.
- jpeg2k/ilyrrates - The rates for any intermediate layers.
- jpeg2k/imgareatlx - X-coordinate of the image area.
- jpeg2k/imgareatly - Y-coordinate of the image area.
- jpeg2k/lazy - Enables lazy coding mode.
- jpeg2k/mode - The coding mode.
- jpeg2k/nomct - Disallows the use of any multicomponent transform.
- jpeg2k/numgbits - Number of guard bits.
- jpeg2k/numrlvls - Number of resolution levels.
- jpeg2k/prcheight - The precinct height.
- jpeg2k/prcwidth - The precinct width.
- jpeg2k/prg - Progression order.
- jpeg2k/pterm - Uses predictable termination.
- jpeg2k/rate - The target rate.
- jpeg2k/resetprob - Resets the probability models.
- jpeg2k/segsym - Uses segmentation symbols.

- jpeg2k/sop - Generates SOP marker segments.
- jpeg2k/termall - Terminates all coding passes.
- jpeg2k/tilegrdtlx - X-coordinate of the tiling grid.
- jpeg2k/tilegrdtly - Y-coordinate of the tiling grid.
- jpeg2k/tileheight - The nominal tile height.
- jpeg2k/tilewidth - The nominal tile width.
- jpeg2k/vcausal - Uses vertical stripe causal contexts.

- L -

- log/file - Name of the log file
- log/filter - Message filter
- log/format - Format string
- log/ident - Identification string
- log/levels - Strings for level codes
- log/maxfilesize - Maximum number of files

- M -

- motdet/block_x - Width of a block
- motdet/block_y - Height of a block
- motdet/contrast_min - Minimal pixel contrast
- motdet/scale_x - Reduced value in X orientation
- motdet/scale_y - Reduced value in Y orientation
- motdet/sensibility - Sensibility
- motdet/xsize - Width of a still image (integer)
- motdet/ysize - Height of a still image (integer)

- W -

- wsq/bitrate - Bitrate parameter of the wsq algorithm

- Z -

- zoom_mode - Zoom type (integer)

Abbreviations

ActiveX – Programming language independent software technology

BGR – Image pixel format where each pixel is composed by blue (B), green (G) and red (R) components

BMP – Image file format (Device Independent BitMaP)

DCT – Direct Cosine Transformation

DLL – Dynamically Linked Library

GIF – Image file format (Graphics Interchange Format)

HTML – HyperText Markup Language

IDCT – Inverse Direct Cosine Transformation

JPEG – Image file format (Joint Photographic Expert Group)

MSDN – Microsoft Software Developer Network

.NET – Programming language independent software technology

PDF – Portable Document Format

PNG – Image file format (Portable Network Graphics)

RGB – Image pixel format where each pixel is composed by (R), green (G) and blue (B) components

RAD – Rapid Application Development

RAW – Uninitialized data format or application specific byte sequence.

SDK – Software Development Kit

TIFF – Image file format (Tagged Image File Format)

WSQ – Image file format (Wavelet Scalar Quantization)

YCbCr – Image pixel format where each pixel is composed by the luminance (Y), and the chrominance (Cb), (Cr) components.

FAQ - Frequently Asked Questions

How to compile an example?

Under Linux

Use the Makefile in the directory of the sample code or you can type the following command into the command line:

```
g++ -o sample.out sample.cpp -DLINUX -I/usr/include/gx -ldl –lpthread
```

Under Windows

Use the project generator batch files located in the sample directories to generate project files, workspaces or solutions.

How to install modules on embedded GNU/Linux systems?

Required files for minimal installation

- Property data
  - /var/gx/gxsd.dat
- The base GX library
  - /usr/lib/libgxsd.so
  - /usr/lib64/libgxsd.so (under 64 bit systems)
- Other GX libraries (/usr/lib/gx/*.so or /usr/lib64/gx/*.so)
  - Required shared libraries:
    - gxsdl.so : watchdog and NNC communication
    - gxproperty.so, gxpropfile.so : for property handling
    - gximage.so : for image handling
  - Other libraries (not needed for minimal system):
    - gxbmp.so,gxpng.so,gxjpeg2k.so : for BMP, PNG and JPEG-2000 handling
    - gxjpeg8.so : for 8 bit JPEG handling (it requires the gxjpeg.so)
    - gxjpeg12.so : for 12 bit JPEG handling (it requires the gxjpeg.so)
    - gxlog.so : for logging
    - gxmotdet.so : for motion detection
    - gxtrafo.so : for transformation (required for passport reader)
    - gxzlib.so : for compression/decompression
- In case of using additional modules, those data files have to be copied into the following directory:
  - /usr/share/gx/

The .dat files can be assigned to the .so files in the gxsd.dat property file (in this case the version numbers must be the same).

Kernel modules

Do not forget to copy or to install the kernel drivers (you can copy them from a fully installed system which has the same kernel):

- gxsd
- fxmc_usb (if you use an USB NNC device)
- fxmc_pcie (If you use PCIe NNC device)
- fxmc (if you use PCI NNC device or PCI FXVD4 card)
- fxvd4 (if you use FXVD4 card)

How to load a RAW image?

To load a RAW image, the holder structure must be initialized properly, e.g. it has to be created with the correct sline value. After initialization, the image can be loaded from the memory setting -1 for the format parameter.

User's FAQ

Q: *I can not compile the fxcam and the GX sample programs. I got an error message from the compiler saying:*

```
"ResolveAssemblyReferences: Primary reference "fxcamdotnet8". Could not resolve this
reference. Could not locate the assembly "fxcamdotnet8". Check to make sure the
assembly exists on disk. If this reference is required by your code, you may get
compilation errors."
```

and

```
"ResolveAssemblyReferences: Primary reference "gxdotnet8". Could not resolve this
reference. Could not locate the assembly "gxdotnet8". Check to make sure the assembly
exists on disk. If this reference is required by your code, you may get compilation
errors."
```

What now? By the way, I use Windows XP operating system and C# programming language.

A: Check the presence of the following directories:

```
"%SystemRoot%\Windows\assembly\GAC_32\gxdotnet8"
"%SystemRoot%\Windows\assembly\GAC_32\fxcamdotnet8"
```

If your operating system is not up-to-date, it might happen that the installer can not create the above mentioned directories. If these directories do not exist on your storage device then:

1. update your operating system
2. reinstall your gx/fxcam package

IMPORTANT: Note that samples are instructional and may not include all security mechanisms required for a production environment.

Contact Information

Should you have any problem during operating the products, our support team is at your disposal. Please try to explain the problem as detailed as possible and do not forget to send the following information to make it easier to help you:

- The name of your company (for administration purposes).
- The **exact type** of the **product** you have (serial number is appreciated).
- If you have problems **during recognition, send images** in the original file format.
- If there is any **error code or message** appearing, please send us the code snippet where it occurs (a screenshot may also be helpful).
- If you noticed the problem while running a Demo or a sample application, please let us know the **name of the application** you tested.
- If you have some problem while **developing your own** application, please specify the followings:
  - your programming language
  - your operating system
  - the name and version number of the compiler you use
  - the programming technology (e.g. native C/C++ / ActiveX / .NET)
  - If possible please send a short part of the source code. Please try to determine the place where the error occurs (e.g. “the scapture?getshutter command returns a negative number…“).

IMPORTANT NOTES:

- **Before sending back a faulty device, always contact ARH Support Team.**
- **Repairs may be executed by the manufacturer only!**

Office address:
ARH Inc.
41 Alkotás Road
HU-1123 Budapest
Hungary
Phone: +36 1 2019650
Web: www.arhungary.hu
Email: support@arh.hu

Service address:
ARH Inc.
Ipari Park HRSZ 1113/1
HU-2074 Perbál
Hungary
Phone: +36 1 2019650
Web: www.arhungary.hu
Email: rmarequest@arh.hu