

# **USING THE NOKIA 3650 CAMERA**

**Version 1.0**

**06 Sept 02**

# Table of Contents

<b>1. INTRODUCTION</b> .....	<b>3</b>
1.1 PURPOSE AND SCOPE .....	3
1.2 HARDWARE AND SOFTWARE REQUIREMENTS .....	3
<b>2. IMAGING OVERVIEW</b> .....	<b>4</b>
<b>3. TAKING A PICTURE</b> .....	<b>5</b>
3.1 CONNECTING TO AND DISCONNECTING FROM THE CAMERA SERVER.....	5
3.2 TURNING THE CAMERA ON AND OFF .....	5
3.3 CAMERA SETTINGS.....	7
3.4 TAKING A PICTURE .....	7
<b>4. DISPLAYING, STORING, AND RETRIEVING AN IMAGE</b> .....	<b>9</b>
4.1 DISPLAYING A CAPTURED IMAGE.....	9
4.2 STORING AN IMAGE .....	9
4.3 FETCHING AN IMAGE .....	10
<b>5. CONCLUSION</b> .....	<b>11</b>

## Change history

06 Sept 02	Version 1.0	Document published in Forum Nokia.
------------	-------------	------------------------------------

### Disclaimer:

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to the implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time without notice.

### License:

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

## 1. INTRODUCTION

The Nokia 3650 imaging phone is based on the open Series 60 Platform standard and includes a built-in camera capable of capturing images at resolutions up to 640 x 480 pixels with a pallet of up to 16 million colors. Camera services are implemented as a shared resource that can be accessed simultaneously by several applications.

### 1.1 Purpose and Scope

The following tutorial is intended for application developers who want to make use of the camera and imaging capabilities built into Nokia 3650, which is based on Nokia's Series 60 Platform. This document presents a brief introduction to accessing the imaging API on the handset to:

- 1) Connect to the camera server and take a picture or capture an image
- 2) Display, store, and retrieve those captured images

### 1.2 Hardware and Software Requirements

The code snippets included in this tutorial are extracted from a fully functional source-code example listed in the file `3650camexercise.cpp` available for download at [www.forum.nokia.com](http://www.forum.nokia.com). To successfully compile and run the example code found in this tutorial, developers should have a system that meets various hardware and software requirements. These include:

- 1) Operating system – Microsoft Windows NT 4.0, Service Pack 6, or Windows 2000. Other operating systems may or may not work with the examples, but these and more recent Microsoft platforms are likely to yield the best results.
- 2) A minimum of 500 megabytes free hard-disk space.
- 3) Series 60 SDK for Symbian OS. This can be downloaded from [www.forum.nokia.com/tools](http://www.forum.nokia.com/tools) or obtained via a Nokia-supplied CD.
- 4) Microsoft Visual C++ 6.0 with Service Pack 3 – this is strongly recommended, especially if developers expect to build and debug code.
- 5) Finally, the system should meet the processor speed and RAM requirements called for by Microsoft Visual C++.

Final device testing, of course, requires access to a Nokia 3650 handset. Camera client applications cannot be tested in emulators. Executable packages are transferred to the phone via an infrared link or other means. Therefore, the development system must have one of these transfer methods available. For a complete description of the SDK, the simulator, and the transfer methods please see the documentation supplied with the SDK.

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

## 2. IMAGING OVERVIEW

Nokia 3650 is built on Series 60 Platform, an open systems platform for one-handed smartphones and imaging phones. A camera server, as the name implies, controls the handset's built-in camera. The camera server provides the interface for setting up the camera and the means to fetch pictures taken by the camera. The speed of image capture depends on a number of factors, including the resolution and color depth of the image. Developers should experiment and determine the best settings for a given situation, especially if image capture speed is important for a particular application. Testing should also be done to ensure that critical timing issues are not a problem.

Nokia 3650 provides tools to manipulate images once captured. There are codecs to encode and decode imaging data. The handset also offers an animation framework. A Series 60 API enables the conversion of an image between various formats.

Finally, the handset and Series 60 Platform supply a Photo Album engine. This API handles the filing chores associated with storing an image. All images and pictures should come through this. An important incentive to do so is that the API for the Photo Album manages the creation of thumbnails. Having these on hand allows users to quickly select an image from an array of stored pictures.

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

## 3. TAKING A PICTURE

The camera on Nokia 3650 is a shared resource and operates, at times, in an asynchronous mode. For these reasons, control and management of the camera in the handset has been implemented using a client/server architecture. The camera server should not be bypassed. Doing so can lead to unexpected problems and unexpected results. The following section describes how to connect to the camera server and capture an image.

### 3.1 Connecting to and Disconnecting from the Camera Server

Connecting to the camera server is straightforward and must be done prior to issuing any imaging device instructions. The following command connects to the camera server:

```
TInt RCameraServ::Connect();
```

This rather simple approach works, but a better method includes error trapping and recovery in case of a resource problem. The document entitled "Coding Idioms for Symbian OS," which is part of the information in the Series 60 SDK, details how to implement out-of-resource error handling. Thus, a better approach for camera server connection that incorporates this dictate is:

```
{  
    // Initialize a client to the camera server  
    iCamserv = new(ELeave) RCameraServ;  
    // Connect to Camera Server  
    User::LeaveIfError(iCamserv->Connect());  
}
```

It is good practice to disconnect from the camera server when finished. This avoids errors and is easy to implement. The following command disconnects a client from the camera server.

```
TInt RCameraServ::Close();
```

Again, a better method makes use of the client defined earlier to break the connection to the camera server:

```
if(iCamserv) iCamserv->Close();
```

### 3.2 Turning the Camera On and Off

Before a picture can be taken, the camera must be turned on. It is important to remember that a call to turn on the imaging device is asynchronous and may take some time to return. This must be accounted for both in any application and in all instructions issued to the camera server.

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

The following code turns the camera on and off. Returned status indicates the success or lack of success resulting from trying to turn the camera on.

```
RCameraServ::TurnCameraOn(TRequestStatus & aStatus )  
  
TInt RCameraServ::TurnCameraOff()
```

This basic approach should be augmented through the use of good Symbian programming procedures, which ensure that if problems do occur they do not result in resource leaks. Improved code for turning on the camera is given below; it makes use of the client defined earlier:

```
TurnCameraOnL();  
  
// Description: Turn on the camera. Turning on the camera //  
takes some time.  
  
// Return value: N/A  
{  
// Status variable for async function calls  
TRequestStatus status( KErrNone );  
// Turn camera ON  
iCamserv->TurnCameraOn(status);  
User::WaitForRequest(status);  
if( status.Int() != KErrNone )  
{  
// error while turning ON, closing connection to server  
iCamserv->Close();  
User::Leave(status.Int());  
}  
}
```

The same technique can be used to turn the camera off. This approach allows either an immediate camera shut down after taking a picture or at any other time:

```
TurnCameraOffL();  
  
{  
// Turn camera OFF  
User::LeaveIfError(iCamserv->TurnCameraOff());  
}
```

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

## 3.3 Camera Settings

The camera in Nokia 3650 offers two different settings that impact the imaging experience for end users. The first is concerned with the different lighting conditions; the second involves image resolution and color depth. The Nokia 3650 handset camera supports a 16-million-color, 640- x 480-pixel high-resolution setting and a 4096-color, 160- x 120-pixel low-resolution setting.

The speed with which pictures are taken depends on the lighting condition and the image resolution and color depth. The size of the resulting image files may also be important. The high-resolution image setting has 16 times the number of pixels and double the color bit depth of the low-resolution setting. The time and processing burden of the higher-quality image must be weighed against the user response to the resulting picture.

Setting up the camera for various lighting conditions involves the following code. In this example, `TLighting` is either `ELightingNormal` or `ELightingNight`. Status is returned:

```
SetLightingConditions(const TLighting aLightCondition)
```

Setting up the image quality is also clear-cut. In the following code, `TImageQuality` can be either `EQualityHigh` for a 640 x 480, 16-million color image or `EQualityLow` for a 160 x 120, 4096-color picture. Again, result status is returned:

```
SetImageQuality(const TImageQuality aQuality)
```

Prudent practice wraps error handling around these simple commands. The code below uses the client defined earlier and provides a means to *leave* if an out-of-resource error occurs. The code sets up the camera for a high-quality image under normal lighting conditions. Of course, those parameters can be changed to meet the needs of a particular application:

```
{
// Set image quality
User::LeaveIfError(iCamserv->SetImageQuality(RCameraServ::EQualityHigh));
// Set light condition
User::LeaveIfError(iCamserv->SetLightingConditions(RCameraServ::ELightingNormal));
}
```

## 3.4 Taking a Picture

With the client connected to the camera server, the camera on, and the setup done, it is time to take a picture. The result will be stored in a bit map, as per the command below. The status of the command results is also returned:

```
RCameraServ::GetImage(TRequestStatus & aStatus, CFbsBitmap & aBitmap)
```

Like other camera functions, this is an asynchronous operation. It also requires accessing the resulting bit map. The function below – `GetBmpForSnapShot` – returns a pointer to a bit map and provides access to that captured image:

```
GetBmpForSnapShot()  
{  
    // This function gets the bit map to take a new picture  
    // Reset the bmp first  
    iImageReady = EFalse;  
    iBmp->Reset();  
    return iBmp;  
}
```

The function is used in the code below to provide a bit map handle to the picture. The code also contains provisions for the asynchronous nature of picture taking, as well as a means to recover from errors and out-of-resource conditions:

```
// Get a bmp handle from our container  
CFbsBitmap* bitmap = NULL;  
bitmap = iContainer->GetBmpForSnapShot();  
// Status variable for async function calls  
TRequestStatus status( KErrNone );  
// Get an image  
iCamserv->GetImage( status, *bitmap );  
User::WaitForRequest(status);  
if( status.Int() != KErrNone )  
{  
    iCamserv->TurnCameraOff();  
    User::Leave(status.Int());  
}
```

## 4. DISPLAYING, STORING, AND RETRIEVING AN IMAGE

With a picture successfully taken, the next step is to either display it immediately or save it for subsequent use. The choice clearly depends on the application in question. Nokia 3650 and Series 60 Platform provide a number of format conversions, as well as bit map utilities, to help accomplish this. This section describes some of the choices available to developers.

### 4.1 Displaying a Captured Image

In the example given above, a small piece of additional code inserted at the end will display the picture just taken. This extra segment is shown below:

```
// Show the image on screen

iContainer->iImageReady = ETrue;

iContainer->DrawNow()
```

The code first checks to see if the image is ready. It then displays the image when it is available. As can be seen, given a pointer to a bit map, the `DrawNow()` function will render the image on the screen. But in many cases and applications, a captured image will not be immediately displayed. There may be a need to store the image for later use or manipulation, e.g., a slide show or a case where a series of pictures are to be taken at some preset interval.

### 4.2 Storing an Image

Series 60 Platform provides a number of utilities for image storage and manipulation. In the case of image handling, a bit map may be converted to a file-based image via the `CMdaBitmapToImageFileUtility` and vice versa with the `CMdaImageFileToBitmapUtility`. The same can be done with a descriptor-based image and a bit map with the `CMdaImageDescToBitmapUtility` and `CMdaImageBitmapToDescUtility`. It is also possible to convert one bit map to another via the `CMdaImageBitmapToBitmapUtility`.

Bit maps can be stored through the Photo Album utility using the API provided in the `CPAlbImageUtil` class. These images can be retrieved, along with thumbnail representations, via the API provided in the `CPAlbImageFactory` class. There are both Copy and Move functions available for transporting images into the Photo Album. The only difference between the two is that Copy preserves the source file while Move does not.

When sending images into the Photo Album, two parameters are involved: the first is the `aSourceFile` descriptor, which contains the full path name to the source file; the second is the `aReplace Boolean`, which indicates whether or not to replace files in Photo Album. A complete description of this class can be found in the SDK help documentation.

The following code creates a utility handler for saving a bit map to a file. Most of imaging utilities implemented in the Series 60 Platform support a set of callback functions defined in the `MMdaImageUtilObserver` class. Therefore, an object which implements these callback functions pointed to by `*this`, is passed to the conversion utility handler. It makes use of `iFileSaver = CMdaImageBitmapToFileUtility::NewL(*this)`, a file-saving function that incorporates leave exception handling. Another function that is used in the code is `iFormat = new (ELeave) TMdaJfifClipFormat`; this provides the codec support needed to store the file in something other than its native bit map format:

```
// gets path where to save the image
    TFileName newFilePathAndName(KCamExerciseFilename);
    // Create Format and Codec
    // quality factor from 0 to 100 (55 used here)
        iFormat->iSettings.iQualityFactor = KJpgSavingQualityFactor;
        iFormat->iSettings.iSampleScheme =
            TMdaJpgSettings::TColorSampling(TMdaJpgSettings::EColor420);
        TMdaPackage* codec = NULL;
    // create a file to save the image into
    // in completion the MiuoCreateComplete() callback function is
    called
        iFileSaver->CreateL( newFilePathAndName, iFormat, codec,
            NULL)
    // see how to save an image to a file from the example code in these
    functions
    // MiuoCreateComplete(TInt aError)
    // MiuoConvertComplete(TInt /*aError*/)
```

Other encoding formats supported include MBM and BMP. Decoding formats cover a wide variety, including TIFF, gif, Wmf, and others. As noted in the code above, file operations are asynchronous in nature and therefore go through the Nokia 3650 file server.

### 4.3 Fetching an Image

Retrieving an image is done with the following code. Again `*this` is a pointer, pointed to an object, which implements a set of callback functions defined in the `MMdaImageUtilObserver` class. A conversion utility handles the translation between the stored file and the to-be-displayed bit map. The `KLoadFilename` variable points to the file in question and provides the path and filename:

# NOKIA

Using the Nokia 3650 Camera

Version 1.0

```
iFileLoader = CMdaImageFileToBitmapUtility::NewL(*this);  
iFileLoader->OpenL(KLoadFilename)  
  
// see how to load an image from a file from the example code  
// in this function  
  
// MiuoOpenComplete(TInt aError)
```

When this executes, the result is a pointer to the file in question, which can then be used to extract the image and display it. However, the image must be handled in a way consistent with the encoding. There may, for instance, be a need to display only the first frame or to scale it appropriately for a bit map. The exact actions to be taken depend on the image and application itself.

For example, this code snippet will create a bit map based on the size of a gif image:

```
bitmap->Create(frameInfo.iOverallSizeInPixels, KDeviceColourDepth).
```

This can then be scaled for display on Nokia 3650 using `CMdaBitmapScaler`. Images can be zoomed in or out as needed. Images can also be rotated in 90° increments using `CMdaBitmapRotator`.

## 5. CONCLUSION

This tutorial provides basic information about the camera and imaging capabilities of Nokia 3650. Further examples, help, and information can be found in the Nokia Series 60 SDK as well as other documentation available at [www.forum.nokia.com](http://www.forum.nokia.com). The complete example code for this tutorial is available for download as file `3650camexercise.cpp` at [www.forum.nokia.com](http://www.forum.nokia.com).