 **Polycom® KIRK® DECT Application
Module 6.0**

Software User Guide

Trademark Information

© 2008, Polycom, Inc. All rights reserved. POLYCOM®, the Polycom "Triangles" logo and the names and marks associated with Polycom's products are trademarks and/or service marks of Polycom, Inc. and are registered and/or common law marks in the United States and various other countries. All other trademarks are property of their respective owners. No portion hereof may be reproduced or transmitted in any form or by any means, for any purpose other than the recipient's personal use, without the express written permission of Polycom.

All other trademarks are the property of their respective owners.

Patent Information

The accompanying product is protected by one or more U.S. and foreign patents and/or pending patent applications held by Polycom, Inc.

1© <Copyright Date> Polycom, Inc. All rights reserved.

Polycom, Inc.
4750 Willow Road
Pleasanton, CA 94588-2708
USA

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Polycom, Inc. Under the law, reproducing includes translating into another language or format.

As between the parties, Polycom, Inc., retains title to and ownership of all proprietary rights with respect to the software contained within its products. The software is protected by United States copyright laws and international treaty provision. Therefore, you must treat the software like any other copyrighted material (e.g., a book or sound recording).

Every effort has been made to ensure that the information in this manual is accurate. Polycom, Inc., is not responsible for printing or clerical errors. Information in this document is subject to change without notice.

Contents

1	Preface	7
	Prerequisite Skills and Tools	7
	Skills	8
	Tools	8
	Related Documentation	8
	Acronyms	9
2	Getting Started	10
	Software Overview	10
	Application Overview	11
	Extending the MMI	11
	Extending the DSP	11
	Importing the EKGAP DDE into Eclipse	11
	Developing a Basic Application	13
	Running the Basic Application	13
	Downloading Firmware to KDAM 6.0	14
	Configuring the KDAM 6.0	14
	Using Tiny Serio to Change the Configuration	16
	Tuning the KDAM 6.0	17
3	Verification Scenarios	18
	Scenarios	18
	Subscribing a Handset to a Module FP	18
	Registering Handset Location	19
	Sending an MSF to a Module FP	19
	Subscribing a Module PP to a KWS600v	19
	Sending an MSF to a Module PP via KWS600v3	20
	Establishing a Voice Connection Between Two Handsets	20
	Establishing a Data Connection Between Two Module PPs	20
	Extending the Sample Application	20
4	Advanced Features	22
	Simulation	22
	Tracing Scenarios	22
	Replaying Traced Scenarios	23
	Running in a Hosted Environment	23

5	Module API	25
	Overview of Event Driven Software Execution	25
	Signals and Signal Handlers	25
	Mails and Tasks	26
	Subscription and Location Registration	27
	MSF	28
	Call Establish	28
	Speech Buffers and Audio Control	29
	Interrupt Handling	29
	Other Functionality	30
6	Advanced Programming Examples	33
	Routing Speech via the PCM Interface	33
	Establishing a Connection with a PP	33
	Looking up the Speech Buffer Index	34
	Connecting the RX Speech Buffer to the DSP	34
	Starting the DSP	34
	Connecting DSP Output to the PCM interface	35
7	Troubleshooting Applications	36
	My Board Enters Boot Mode Unintentionally	36
	I Cannot Subscribe a Module PP/DECT Handset to a Module FP	37
	I cannot run the host.exe process inside Eclipse	37
	Eclipse Cannot Invoke the GCC Compiler	37
	How do I Set Up a KDAM 6.0 PP to Subscribe to a KWS600v3?	38
	How Does Call Release Work?	38
	How Do I send a Command or Data from the PC to the Module?	38
	How Do I Send Data from the Module to the PC?	39
	How do I Enable Echo Cancellation?	39
	How do I get the RSSI of the Locked Base?	39
8	Setting up a Communication Channel	40
	Subscribing the KDAM 6	40
	Subscribing handsets	42
9	Communication Between KWS and a Third-Party Application	44
	Using the MSFDLL.DLL	44
	MSFDLL.DLL Reference	44
	MSF Structure	47
	MSF Commands	47

Dial	47
Answer Connection	48
Hang Up	48
Hang Up Acknowledge	49
10 AT Commands	50
AT Commands	50
AT Command Reference	50

Preface

There are three approaches to using the Polycom KIRK DECT Application Module 6.0 (KDAM 6.0):

- Using AT commands
- Using the Virtual Handset
- Developing applications

This document describes all three approaches, but with main focus on how you use the (EKGAP DDE) to develop applications for the KDAM 6.0.

Through the EKGAP DDE you can build and run your own applications on top of the DECT protocol stack, and you can change the behavior of parts or all of the pre supplied software.

The EKGAP DDE is integrated with the Eclipse integrated development environment, and based on open source software.

The API that is supported is a low level signalling API.

The first section of this guide takes you through the basic steps of building and running a basic application and of downloading firmware to the module. The next sections deal with the more advanced features of the KDAM 6.0.

The final three sections focus on using the KIRK Virtual Handset and AT commands.

Prerequisite Skills and Tools

Developers who want to develop applications for the KDAM 6.0 using the EKGAP DDE are expected to already have knowledge of some of the technologies that are used and the general challenges of embedded software development.

For prerequisites for using the KIRK Virtual Handset and AT commands, see [“Setting up a Communication Channel”](#) on page 40.

Skills

To develop applications for the KDAM 6.0, you should have knowledge of the following.

- Knowledge of the Eclipse development environment.
- Knowledge of the C programming language and embedded software development.
- Knowledge of the IAR compiler.
- Knowledge of a Polycom KIRK Wireless Server and/or DECT handset if you want to test your application against any of these. For more information see www.polycom.com.

If you would like a deeper understanding of the KDAM before you start development, ECT A/S, a Polycom partner, offers specialized training in, for example, how to use Simulink for programming the Gen2DSP. ECT A/S has expert knowledge of both DSP and CR16 software modules for the KDAM, and also offers custom KDAM development on a contract basis. For more information, see www.ect.dk

Tools

Before you can use the EKGAP DDE, you must install the following software.

- Eclipse development environment. See, <http://www.eclipse.org/downloads>. Select Eclipse IDE for C/C++ Developers.
- IAR compiler. See www.iar.com for details.
- Java: <http://java.com/en/download/>
- C/C++ Development Toolkit (CDT) for Eclipse. See the Eclipse documentation for details.
- Eclipse KGAP plug-in from Polycom: is distributed by Polycom as a single jar file, and must be placed in the Eclipse plug-in folder
- Optionally, Install the GCC compiler.

Related Documentation

For detailed information about the module hardware, see the Polycom KIRK DECT Application Module 6.0 Hardware Guide. You can download the guide from www.polycom.com

Acronyms

Table 1 Acronyms

Acronym	Meaning
ADPCM	Adaptive Differential Pulse Code Modulation
ARI no.	Access Rights Identity - Serial number of the DECT system
DECT	Digital Enhanced Cordless TELEcommunications
DSP	Digital Signal Processing
EEPROM	Electronically Erasable Programmable Read Only Memory
EKGAP DDE	EKGAP Desktop Development Environment
FP	Fixed Part (Base Station)
GAP	Generic Access Profile
IPEI	International Portable Equipment Identity - Serial number of the handset - SN
KDAM 6.0	KIRK DECT Application Module 6.0
LPCM	Linear Pulse Code Modulation
PCM	Pulse Code Modulation
PP	Portable Part (handset)
UART	Universal Asynchronous Receiver/Transmitter

Getting Started

This section describes the software layers of the module, and the units that make up a software application for the module. Furthermore, this section describes how to build and run a basic application for the module.

Software Overview

The software of the EKGAP DDE is divided into the following three layers.

- **MAC layer**

This layer is the lowest software layer. The MAC layer runs on the pre-installed radio module, which is supplied as part of the development package. The radio module communicates with a PC via a USB-RS232 interface cable.

- **EKGAP DECT protocol stack**

This layer implements the KIRK DECT protocol. It is supplied as a number of object files that you can link with the application you develop.

- **Application layer**

This is the layer that you develop by changing and extending the default behavior of the pre-supplied software.

During development, you can build and run the middle and upper layers on the KDAM 6.0 or a host machine (a PC.) The latter process is further described in [“Advanced Features”](#) on page 22.

You can package and deploy the software to the radio module so that it runs as an autonomous application. However, you need to install the IAR compiler to package the software for deployment to the radio module. For more information see the IAR compiler documentation.

If your product architecture employs an external MAC layer, you do not have to package and deploy the software to the MAC layer on the radio module.

Application Overview

A complete KDAM software application consists of the following software units.

- The DECT protocol stack running on the CR16 processor
- A DSP software module running on the Gen2DSP processor from SiTel
- A user application (MMI) running on top of the DECT protocol stack

The KDAM development platform contains a full implementation of the DECT protocol stack, which is distributed in binary form. It also contains a documented API that can be used when you develop an application.

Furthermore, the platform contains a sample user application (MMI), and a very simple DSP software unit. Both examples are intended as starting points for the development of an application that contains a product specific MMI and DSP module.

Extending the MMI

You can extend the MMI by using the API to add, for example, special DECT logon/subscription behavior, user management features, or special purpose voice/message call scenarios. You program the application in Eclipse and compile the software with the IAR compiler or with the open source GCC compiler tool chain for CR16 processor.

Extending the DSP

You can extend the DSP unit, for example, by adding Echo Cancellation, voice switching, or other advanced audio features. You do the programming in Eclipse or by using the Simulink modeling tool [ref needed] with the appropriate plug-ins [ref needed]. Simulink works well with the GCC compiler. If you decide not to use Simulink, you can use the IAR compiler to generate Gen2DSP code.

Importing the EKGAP DDE into Eclipse

Before you can start developing applications, you have to import the EKGAP Desktop Development Environment into Eclipse, and activate the Polycom plug-in.

Note that this guide does not contain a detailed description of how to work with the Eclipse development environment. For further information, see the Eclipse documentation: <http://help.eclipse.org/ganymede/index.jsp>.

To import the EKGAP DDE

- 1 Create a new folder on your C: drive, and give the folder a meaningful name such as 'workspace'. Eclipse uses this folder for your project source files.
- 2 The EKGAP DDE is distributed as a zipped Eclipse project. Unzip the kgap_devel.zip, and use the Eclipse Import Wizard to import the project into Eclipse. For more information, see the Eclipse documentation.
- 3 From the kgap_devel/eclipse folder, copy Eclipse_IAR_1.0.0.jar to the Eclipse plug-in folder. This folder is created when you install Eclipse.
- 4 Restart Eclipse.
- 5 When the EKGAP DDE is successfully imported, you will see the kgap_devel project in the project explorer. The project contains the following.
 - sample application
 - tools
 - Polycom DECT protocol stack as object files for both the module and a PC host.
 - Documentation

To Install and Activate the EKGAP plug-in

- 1 Install the IAR compiler. For more information, see www.iar.com.

When the Polycom EKGAP plug-in is configured correctly, it automatically invokes the IAR compiler, parses the IAR project files, and builds the software for the radio module
- 2 In **Eclipse**, from the **Window** menu, click **Preferences**, and then select **IAR Preferences**.
- 3 Click the **Restore Defaults** button. The default settings appear.
- 4 In the **IAR install directory** list, verify that the location of the IAR compiler is correct. If not, click the **Browse** button to locate the correct directory.
- 5 If no GCC compiler is installed, type C:\ in the **GCC install directory** field. Otherwise, click the **Browse** button to locate the correct directory.
- 6 Click **OK**.
- 7 From the **Window** menu, point to **Show View**, and then click **Other...** The **Show View** window is displayed.
- 8 Locate and expand the **KIRK IAR** folder, click **IAR Project**, and then click **OK**.

Developing a Basic Application

The following describes how to develop a simple basic application for the KDAM 6.0.

You'll be working from the C/C++ Project explorer, which only displays items that are relevant to C and C++ project files.

To Build a Project

- 1 Click the **C/C++ Project** tab, and expand the **kgap_devel** folder. It has the following folder structure.
 - Includes
 - app: contains the helloworld project
 - bin: contains the object files that makes up the DECT protocol stack linked against the application
 - doc: contains documentation
- 2 Expand the **app** folder, and then expand the **kt4585** folder. The layout of the project is as follows.

```
./app/kt4585/kt4585-target.ewp
./app/kt4585/kt4585-host.ewp
./bin/obj/*.o
./include/*
```
- 3 Click the **kt4585-target.ewp** file, which is the build file. The build file is loaded and parsed. Below the Editor area is a plug-in view of the project where you can build and clean the project. On the **IAR Project** tab, click **Build** to compile and link the application with the DECT protocol stack.
- 4 Click the **Console** tab to follow the build progress.
- 5 Press **F5** to refresh the **kgap_devel** folder in the **C/C++ Project** explorer. The **kt4585.hsx** file has been added to the **kt4585** folder.
- 6 To delete the files that have been produced during the build process, on the **IAR Project-Debug** tab, click **Clean**. This removes all files generated by the compiler.

Once you have compiled and linked the application successfully with the DECT protocol stack, you can begin the actual testing of the application.

Running the Basic Application

When you have build the kt4585.hsx application, you can run it on either a module FP or a module PP.

If you want to trace the execution of the application, you have to attach the radio module to the host PC using the USB-RS232 interface cable.

Downloading Firmware to KDAM 6.0

The following describes how you download the firmware you develop to the KDAM 6.0.

Note that the KDAM 6.0 also comes with firmware that is pre-installed in the bin/firmware folder in your workspace. If you experience problems downloading your own firmware, you can try using the pre-installed firmware.

To Download New Firmware

- 1 In the Windows File Explorer go to the C:\workspace\tools folder, and then double-click KIRK_FlashLoader_SC14480 to open the KIRK Flash Loader.
- 2 From the **Com Port** list, select the relevant **COM port**.
- 3 From the **Product** list, select **Type KT4580_01_230400: KT4580 Data Module - SC14480**.
- 4 From the **Flash File (Intel Hex/binary)** field, click **Find** and locate C:\workspace\kgap_devel\bin\firmware\kt4585.hsx.
- 5 On the radio module, press the **Boot** button and then simultaneously press the **M** (reset) button. Let go of the **M** (reset) button, and then let go of the **Boot** button.
- 6 In the KIRK Flash Loader, press the **Start Load** button. When the download has completed successfully, the module is in stand-by mode.
- 7 To start the module, press the **M** (reset) button.

Configuring the KDAM 6.0

The KDAM 6.0 is delivered with default EEPROM values. However, it is possible to change the values if required. You do this in the tool 'Tiny Serio', which is located in the tools folder.

You also have to give the board a default EEPROM layout when there is a new release of the platform.

The following is a list of the EEPROM parameters you can use to configure the radio module. All numbers are in Hexadecimal.

Table 1 EEPROM parameter

Description	Address Range	Default Hexadecimal Value	Comments
ARI/EPEI The FP and PP serial number	0x00->0x04	00 00 00 00 20	<ul style="list-style-type: none"> All DECT units must have a proper, unique, legal serial number.
Run mode	0x8	0x01	<ul style="list-style-type: none"> Starts up the module in full mode. 0x00 starts up the module in minimal mode.
Antenna select	0x09	0x01	<ul style="list-style-type: none"> Means use antenna 1. 0x02 means use antenna 2. 0x00 means use automatic antenna diversity
Output mode	0x0a	0x00	<ul style="list-style-type: none"> Means no output. 0x01, 0x02 means a textual trace is printed to the serial line. 0x4 streams a binary trace of the execution to the serial line and can be used for replaying the board behavior on a PC host. Values can be ORed together.
Frequency adjustment	0x23	0x9a	<ul style="list-style-type: none"> Calibration of the clock crystal.

Table1 EEPROM parameter

Description	Address Range	Default Hexadecimal Value	Comments
Modulation	0x42	0x3a	<ul style="list-style-type: none"> Signal modulation strength
User subscription data	0x10a-0x1b0	0xff	<ul style="list-style-type: none"> Subscription data.
DECT band	0x20	0x00	<ul style="list-style-type: none"> DECT band. Default is European DECT. Set to 0x04 to configure US DECT

Using Tiny Serio to Change the Configuration

Tiny Serio is used to send commands to the module to change the configuration. The following describes the commands you can send using Tiny Serio.

To View Print Out Statements

- To view print out statements from the module, from the **Debug** menu, select **Mail Monitor**.

To Run the Software on a Host PC

- To run parts of the software on a host PC, click the **Connect with host** button. For more information, see ["Running in a Hosted Environment"](#) on page 23.

To Run the Software in Normal Mode

- To run the module in normal mode without a host process, click the **Run standalone** button.

To Set the Verbosity Level

- To set the verbosity level, click Set output level.
 - 0x01 provides output
 - 0x06 enables tracing
 - 0x00 turns off output

To write the ARI/EPEI to the module

- ☞ Right-click the Write ARI/EPEI button. A window appears with a list of hexadecimal values. The last 5 bytes are the ARI/EPEI number.

To Reset EEPROM Settings

- ☞ To restore all default EEPROM settings, in **P1** enter 01, and then click the **Soft/Hard default** button.

To restore all default EEPROM settings except the tuning parameters, in **P1** enter 00, and then click the **Soft/Hard default** button.

To Read Address Content

- ☞ In **P1**, enter the desired EEPROM address, and then click the **Read Address** button.

Tuning the KDAM 6.0

The KDAM 6.0 is tuned on delivery. However, if you, for example, connect the KDAM 6.0 with some other hardware, you may have to tune the board again.

When you tune the KDAM 6.0, you find the proper values for the xtal frequency and signal modulation.

To set the xtal frequency

- ☞ Find the xtal frequency by measuring the oscillation of port P2.7 on the module.

The proper adjustment consolidates the oscillation at 273600.00x MHz when running at room temperature.

If you do not have the right equipment for measuring the oscillation, keep the default adjustment values, which will work in most cases.

Verification Scenarios

You can use the scenarios in this section to verify that the module runs as expected, and you can use them as samples for further development.

Scenarios

The following scenarios illustrate the usage of the KIRK Application Module 6.0.

Subscribing a Handset to a Module FP

This scenario demonstrates if the module FP is configured properly and is able to accept wireless connections. The scenario only activates software inside the DECT protocol stack. It does not involve application layer code.

The scenario has completed successfully when the DECT handset is subscribed and has a stable lock on the module FP.

To Subscribe Handsets

- 1 Clear any existing user subscription data. For more information, see [“To Reset EEPROM Settings”](#) on page 17.
- 2 Start a subscription procedure on the DECT handset. The procedure varies from handset to handset. For more information, see the documentation for each handset on www.polycom.com.
- 3 Open the dump file `dump.txt` in an editor. The dump file contains traces of all communication between the user application and the MAC layer while running.
- 4 In the dump file locate the following line: `[FP]: [RunMode = full] 000344371307-001c8f9639`. The number 000344371307 is the system ARI for the FP. Write down the number, as you will need it to make the subscription.
- 5 While the application is running, start the subscription procedure on the handset, where you search for the system ARI.

When the ARI has been found and the subscription procedure started, you will see a number of signals on the application output.

When the subscription has completed successfully, the radio symbol in the handset is lit, and the handset shows a new display text. The default display text is 'Test'.

When the subscription has completed successfully this is noted in persistent memory on the FP MAC layer.

Registering Handset Location

- ☞ Turn off a previously registered DECT handset, and then turn it on again. Verify that this starts a location registration procedure on the module.

The scenario has completed successfully when the DECT handset is subscribed and has a stable lock on the module FP.

Sending an MSF to a Module FP

- ☞ When the subscription and location registration has completed, send an MSF from the handset to the module FP.

The module FP receives the MSF inside the FPConnectionHandler implemented in FPConnectionHandler.c. When the module receives the MSF, the sample software reports the textual content to the MSF to the serial line, if the verbosity level allows it.

Currently, the sample application cannot send the MSF to another endpoint.

Subscribing a Module PP to a KWS600v

This scenario is similar to "[Subscribing a Handset to a Module FP](#)" on page 18, only this time you use a module PP and communicate with a KWS600v3. If no KWS600v3 is available, you can use a module FP.

To subscribe a module PP to KWS600v3

- 1 Open the user_main.c file, and review the source code that illustrates how to program a PP subscription and location registration scenario.
- 2 Change the ARI of the FP to which you are going to subscribe the module PP to either the ARI of the KWS600v3 or the ARI of the module FP.
- 3 Set the verbosity level to 0x01. The module PP then reports the progress of the scenario on the serial line. If you want to capture a trace of the scenario, see "[To make a trace of a program execution](#)" on page 22-

Sending an MSF to a Module PP via KWS600v3

To send an MSF from a DECT Handset to a Module PP

- 1 Verify that you have a DECT handset and a module PP that are both subscribed to a KWS600v3.
- 2 Send an MSF from the DECT handset to the module.

The MSF ends up in the module PP application which reports the textual content of the MSF to the serial line.

The scenario has completed successfully, if the MSF content is seen on the serial line of the module PP that receives the MSF.

Establishing a Voice Connection Between Two Handsets

To establish a voice connection

- 1 Subscribe two DECT handsets to the same module FP as described in “[Subscribing a Handset to a Module FP](#)” on page 18.
- 2 Dial the number 1000 from one handset, and dial the number 1001 from the other handset.

The sample module FP application contains basic code to connect the two channels. As a result the DECT protocol stack routes the speech between the two handsets.

Establishing a Data Connection Between Two Module PPs

The data that is entered in the speech buffer of one PP is streamed to the other PP. The scenario has completed successfully, if the data that is entered programmatically in the speech buffer of the sending modulePP is reported on the serial line of the receiving module PP.

Extending the Sample Application

The immediate interface that the application layer has to the DECT protocol stack is the FIWU signal handler.

To Subscribe Signals to FIWU

- Use the following code to subscribe signals to FIWU. The code shows how you can easily install user specified signal handlers that can be use to completely change the behavior of the supplied DECT protocol stack.

```
#include "../..//include/hlkgap/duos/include/duos.h"
```

```
#include "../../include/hlkgap/common/include/dispatch.h"

#include <stdio.h>

static void (*FIWUSignalHandler)(SignalType * TheSignal) = 0;

static void KWS500FIWUSignalHandler(SignalType * TheSignal)
{
    printf("KWS500FIWUSignalHandler - ignoring...\n");
    if (FIWUSignalHandler != 0) {
        FIWUSignalHandler(TheSignal);
    } else {
        printf("Error - unable to obtain FIWUSignalHandler\n");
    }
}

// This gives the user of the EKGAP an opportunity to do private setup
// before the
// software starts running
int user_main(int argv, char **args)
{
    printf("Starting user configuration...\n");

    FIWUSignalHandler = getSignalHandler(IWU_PROCESS_ID, 0);

    setSignalHandler(IWU_PROCESS_ID, KWS500FIWUSignalHandler, 0);

    printf("done\n");
}
```

You must implement the `user_main` function. It is called once right after the MAC layer starts up.

In the code example, a reference to the default FIWU signal handler is obtained, and a new signal handler is installed. As a result, signals to FIWU now go to the signal handler you installed rather than to the default signal handler.

The FIWU signal handler implemented in `KWS500FIWUSignalHandler` prints a message on the standard output, and then sends the signal to the default signal handler. It thus acts as a proxy that relays the signal to the originally intended receiver.

Advanced Features

The following section describes the advanced features of the KIRK DECT Application Module 6.0.

Simulation

Typically, you cannot debug a real-time application in an interactive debugger. However, the EKGAP DDE package contains facilities for recording real-time scenarios and replaying them in debug time. This enables you to rerun relevant scenarios as often as required after the scenario has been executed in real-time only once.

To rerun scenarios

- 1 The simulation facility uses the file `dump.text` to rerun recorded scenarios.
- 2 To start the simulation, you must first build the simulation. You do this using the build file `kt4585-host.ewp`.

If the recording was made while subscribing a handset to the FP, you can now use the simulator to retrace what happened during the actual subscription scenario.

Tracing Scenarios

To make a trace of a program that is executed on the module, you have to configure the target to send a trace of its actions to a host process running on your PC.

To make a trace of a program execution

- 1 In **Tiny Serio**, set the EEPROM verbosity level to `0x06`, and then click **Recording on**.

- 2 Build the host process using the build file kt4585-host.ewp. A successful build creates the executable host.exe.
- 3 host.exe takes two parameters to start it up as a sniffer that receives the trace from the board. The first parameter is the name of the file that is used for saving the trace. The second parameter is the name of a file that tells the process which COM port the board is attached to. As an example, see ports.cfg.
- 4 Start the host.exe process from a command prompt or create an Eclipse run configuration.
- 5 Restart the board when the host process has started up. During execution, you will see a textual trace as output from the host process and a binary trace appearing in the dump file. You can now reproduce the scenario you want to examine in more detail.

Replaying Traced Scenarios

When you have traced a scenario, you can rerun the scenario and view what happened during the execution.

To Rerun a Previously Captured Trace

- 1 Make a trace of the scenario as described in [“To make a trace of a program execution”](#) on page 22.
- 2 Restart the host process, but set the second parameter to the word `simulated`. The host process reads the trace file and injects the captured signals into the DECT protocol stack.

Note if you run the simulation from a command prompt, the host process completes very quickly. For the simulation to be useful, run the simulation in a debugger, for example in Eclipse. You can then add breakpoints to the parts of the code to which you have the source, for example, `user_main.c`, and then slowly examine in the debugger what happened during execution.

Running in a Hosted Environment

- 1 Open Tiny Serio.
- 2 Set the board run mode to 0x00, and then press the **Connect with host** button.
- 3 Set the verbosity level to 0x02, and then press the **Output** on button.

All interaction between the MAC layer and the rest of the DECT protocol stack - including your application - is now routed out of the module and over to the host process that runs on the PC.

As a result, you can run part of the software on a host PC or an external processor. This is useful for debugging the application, or if you do not have an IAR compiler available.

Module API

The following section provides information about the KIRK DECT Application Module 6.0 programming API.

The API is divided into two parts. One part is a low level signalling API that defines how to add new signal handlers, send signals, and subscribe to existing signals. The other part defines a number of utility functions you can use when developing software.

Overview of Event Driven Software Execution

The DECT protocol stack and application is implemented on top of an event driven operating system (OS). In an event driven OS, code can only be executed by sending an event to an event handler.

An event is similar to a piece of data, and the event handler is a function to which the data of the event is supplied. A dispatcher, which is part of the OS, maintains a linked list of events, and dispatches the events to their respective event handlers in a First In, First Out (FIFO) manner.

New events can be inserted into the event queue by handlers sending events to other handlers, or by interrupt handlers that insert events into the queue. Interrupt handlers are a special type of event handlers that are scheduled by hardware, and they can interrupt the event handlers scheduled by the OS.

There are two types of event handlers: signal handler and task handler. Events that are sent to signal handlers are called signals, and events that are sent to tasks are called mails.

In the pre-supplied software stack, tasks and mails are used in the MAC layer, and signals and signal handlers are used in the DECT protocol stack and in the application. You have access to both types of event handling systems.

Signals and Signal Handlers

When you run the software, a number of signal handlers are already installed and running. These signal handlers are identified by their ID. You can find the list of available IDs in `dispatch.h`. The signal handlers are required to run the DECT protocol stack.

DispatcherFunction getSignalHandler(ProcessIdType ProcessId, int sessionId)

Returns a function pointer to the signal handler installed to handle signals sent to `ProcessId`. The parameter `sessionId` should always be 0 (zero). This applies to this function and all other functions described below.

void setSignalHandler(ProcessIdType ProcessId, DispatcherFunction handler, int sessionId)

Sets the signal handler of all signals sent to `ProcessId` to `handler`.

Boolean NewSignal(int BufferLength, void **BufferPointer)

Allocates and initializes a new signal of size `BufferLength`. `BufferPointer` must be a pointer to a variable of type `SignalType` (defined in `dduos.h`) or a sub type thereof. New signals can be defined as sub types of `SignalType`,

```
typedef struct _mySignal
{
    SignalType signalHeader; // must be the first element
    unsigned char data;      // add more data here
} MySignal;
```

void SendSignal(SignalType * Signal, ProcessIdType ProcessId, EventType Event, SubEventType SubEvent)

Sends a signal to a signal handler. This is an asynchronous message dispatch: the signal is put on a FIFO queue and dispatched at some later time.

void DeleteSignal(SignalType * Signal)

Signals are allocated using `NewSignal` and must be deallocated using `DeleteSignal`. It is the responsibility of the signal handler to deallocate the signal when it is no longer referenced.

Mails and Tasks

UByte OSGetNumberOfTasks(void)

Returns the number of tasks currently installed in the MAC layer OS.

const TaskDefType *OSGetTask(UByte taskId)

Returns a function pointer to the task handler that handles all mails to `taskId`.

UByte OSRegisterTask(void (*handler)(MailType *MailPtr), char *name)

Registers a new task in the MAC layer OS. Returns the `taskId` of the new task.

char* OSGetTaskName(UByte taskId)

Returns the name of a task. The name does not influence how the task is handled by the MAC layer OS.

void SendMail(TaskIdType taskId, BYTE Length, MailType * MailPtr)

Sends a mail to the task with id `taskId`. The mail must be cast to the Type `MailType` but can contain any type of data. When the mail is dispatched to the receiver, the receiver will be called supplied with the `MailPtr` as argument.

Subscription and Location Registration

void start_pp(int sessionID)

When a module PP starts up, the subscription/location registration procedure is started by calling `start_pp`.

After a while, a signal with Event `IWU_PRIMITIVE` and sub event `OM_SUBS_STATUS_ind` is sent to the application process with information about whether the module PP is subscribed or not. The sample applications illustrate how to handle this event and continue the procedure.

void CallForSearchRFPI(SearchModeType SearchMode, int sessionID)

If `searchMode` is `SM_FindRfpi` the module PP will start searching for bases. Bases found are reported in an `IWU_PRIMITIVE` with sub event `OM_SEARCH_RFPI_ind` sent to the PP application. The sample applications illustrate how to decode this signal.

void CallForSubscription(UByte *rfpi, unsigned long Code_AC, int sessionID)

Starts a PP subscription procedure and is used if the module is not subscribed already. Progress is reported as an `IWU_PRIMITIVE` with sub event `OM_SUBS_CREATE_cfm` sent to the module PP application. The sample applications illustrate to decode this signal is illustrated

Currently the AC code is not supported.

void CallForSubsSelect(BYTE SubscriptionNo, int sessionID)

Starts a PP location registration and is used if the module is already subscribed.

MSF

void SendMsfMessage(char *Destination, char *Message)

Is used by a module PP to send an MSF. If the module PP is locked to an MSF capable FP (such as the KWS600v3) then the MSF will be sent to the destination.

When an MSF is sent to a module PP, it ends up as an APPLICATION_PRIMITIVE with sub event MSFInd in the module PP application. The sample applications illustrate to decode this signal.

When an MSF is sent via a module FP it ends up as an APPLICATION_PRIMITIVE with sub event SS_MSF_prim in the module FP application. The sample applications illustrate to decode this signal.

Note that a module FP does not contain software for sending the MSF to its originally intended receiver.

Call Establish

void callUser_PP(char * number, int sessionId)

If a module is subscribed and locked to a FP, it can start a connection scenario using this function. When an IWU_PRIMITIVE with sub event CC_CONNECT_prim is received in the module PP application the connection has been established.

If a module PP is called, it will get an IWU_PRIMITIVE with sub event CC_SETUP_prim in the module PP application. The sample applications show how to decode this signal.

When the remote party hangs up an IWU_PRIMITIVE with sub event CC_RELEASE_prim is sent to the module PP application.

If a FP module receives a call connection request, an APPLICATION_PRIMITIVE with sub event APPCCSetupInd is sent to the module FP application.

UByte callUser(char * callee, int length, char * caller)

A module FP uses the callUser function to dial a module or a PP. The first parameter is a null terminated text string that contains the portable part PMID to dial.

The second parameter is not used.

The third parameter is the text that is displayed in the in the PP when it is being paged. If the PP accepts the call, a signal with Event IWU_PRIMITIVE and sub event CC_CONNECT_prim is sent to the FP application.

Speech Buffers and Audio Control

UByte* getPPSpeechBufferAddr(UByte index)

Is called by the module PP and returns a pointer to the 40 byte speech buffer that is being used to transmit data or voice over a connection. If connection index is 0, the buffer is the transmit buffer. If connection index is 1, a reference to the receive buffer is returned. The content of this array is sent to or received from the other endpoint every 10 ms.

UByte * getSpeechBufferAddressInRam(BYTE Pmid[3], UByte rxtx)

This function is called by the module FP. Given an identification of a connection (Pmid) a reference to the 40 byte speech buffer used to transmit/ receive data or voice is returned. The Pmid is known from the APPCCSetupInd signal sent to the application at call establish. Rxtx equal to 1 will give the receive buffer. Rxtx equal to one will give the transmit buffer.

UByte getSpeechBufferIndex (BYTE Pmid [3])

Returns an index to the speech buffer that is used by the connection identified by Pmid. For more information, see Chapter 6, "Routing Speech via the PCM Interface,".

Interrupt Handling

At the entry into user interrupt handlers nested interrupts has been enabled by the EKGAP. At the exit from the handler, the interrupt must be cleared through the use of RESET_INT_PENDING_REG. All handlers installed by the EKGAP clears the interrupt. User handlers installed through functions below are executed in context of the interrupt on the interrupt stack.

```
typedef void (*InterruptHandler) (void);
```

InterruptHandler get_interrupt_handler(unsigned short id)

Returns the InterruptHandler currently installed to handle interrupt vector id. If no handler is currently installed, 0 is returned.

void set_interrupt_handler(unsigned short id, InterruptHandler)

Sets the interrupt handler to handle interrupt vector id. If a handler is currently installed, this handler should be saved by a call to get_interrupt_handler and the saved handler should be called at the end of this interrupt handler.

Second Level Interrupt Handling

It is best practice to keep interrupt handlers as short as possible. The logic that handles the interrupt must be deferred to a task, rather than being implemented in the interrupt handler itself.

The interrupt handler collects the information that is needed to handle the interrupt, and then sends a mail to a task. To send mails the following functions must be use.

```
void PutInterruptMail (uint8 Element)
```

Places one byte of data in a buffer. The byte is used to build the mail that is sent to the receiving task one byte at a time.

```
void DeliverInterruptMail (uint8 TaskId)
```

Places the mail in a queue for safe dispatch to the task that performs the second level interrupt handling.

Example

The example is from inside the first level handler.

```
PutInterruptMail (SCAN_COMPLETE_ind) ;
```

```
DeliverInterruptMail (HSCSFTASK) ;
```

The sends a mail with the length of 1 byte. The mail primitive is SCAN_COMPLETE_ind. There is no additional data.

The task (HSCSFTASK) receives the following mail.

```
void HsCsftask (MailType * MailPtr)
```

```
{
    switch (MailPtr->Primitive)
    {
        case SCAN_COMPLETE_ind:
            //Do 2nd level handling here
            break;
    }
}
```

Other Functionality

```
void mac_turnOnLed(void)
```

Is an asynchronous message sent to the MAC layer to turn on the LED.

void mac_turnOffLed(void)

Is an asynchronous message sent to the MAC layer to turn off the LED.

Device Register Access

Device register access is used to control hardware. You can, for example, use the data input/output registers P0[0-7], P1 [0-7], P2 [0-7], and P3 [0-7]. Of these registers, P2[0] and P1 [6] are used by the KDAM 6.0, but you can use the remaining registers. The Sitel data sheet explains the layout and behavior of the device registers.

void requestDeviceRegister(DWORD address)

Requests the content of a device register. Then answer is sent to APPLICATION_PROESS_ID_FP(PP) as a signal with event APPLICATION_PRIMITIVE and sub event MAC_APP_TUNNEL with Signal->Data[0] (LSB) and Signal->Data[1] (MSB).

void setDeviceRegister(DWORD address, WORD value)

Sets the desired device register to value.

Reading from the EEPROM from an Application Process

From inside the user application it is possible to read data from the EEPROM.

First a signal is constructed and sent to request data from the EEPROM. The following is an example of a reading of the first 5 byte from the EEPROM (the ARI/EPEI).

```
static Ubyte eepromData[5] ;
EeReadDataReqType *EeReadDataReq;
NewSignal (sizeof EeReadDataReqType) + sizeof(void *), (void **)
&EeReadDataReq;
EeReadDataReq->ReturnTIPD = TIPD_APPLICATION;
EeReadDataReq->DataTypes = EEDT_ANYADDRESS;
EeReadDataReq->DataPtr[0] = (void *) &eepromData;
memset (&eepromData, 0x00, 5) ;
setAddressData (0, 5) ;
SendSignal ((SignalType *) EeReadDataReq, EE_PROCESS_ID,
EE_PRIMITIVE, EE_READ_DATA_req) ;
```

When this signal has been sent, the application process receives an EE_PRIMITIVE with the sub event EE_READ_DATA_cfm.

The signal itself does not contain the data that have been read. The data that have been read, are placed in DataPtr, which is set when the signal is constructed. In the above example, the data from the EEPROM is saved in the eepromData variable.

Writing to the EEPROM from an Application Process

To write to the EEPROM from the application process, a specific signal is sent with that relevant data.

The following is an example of how to write the value 42 to the address EEP_UserDataArea. You can view the file eeplayout.h to see the location and length of the area in the EEPROM that is allocated to user data. This signal contains no confirmation.

```
EeStoreDataReqType *EeStoreDataReq;

NewSignal (sizeof(EeStoreDataReqType) + 1, (void **)
&EeStoreDataReq);
EeStoreDataReqType->Data[0] = 42; // Write 42 to eeprom
EeStoreDataReqType->DataTypes = EEDT_ANYADDRESS;
EeStoreDataReqType->SubsNo = 0;
setAddressData(EEP_UserDataArea, 1) ;
SendSignal ((SignalType *) EeStoreDataReq, EE_PROCESS_ID,
EE_PRIMITIVE, EE_STORE_DATA_req) ;
```

Advanced Programming Examples

This section contains examples that demonstrate how to use some of the advanced programming features of the KDAM 6.0

Routing Speech via the PCM Interface

The following example describes how you can extend the FP software to route the speech from one connection via the PCM interface as linear PCM.

When the connection to the portable part is established, the FP starts feeding the DSP with the ADPCM that is available in the speech buffer used by the connection.

The DSP converts the ADPCM into LPCM and routes it via the PCM interface using PCM channel 0.

Establishing a Connection with a PP

The first step of the process is to establish a connection with a PP.

To Establish a Connection with a PP

- 1 Subscribe the PP to the FP.
- 2 Have the PP dial a number. The value of the dialled party is not important. The software that is supplied with the module allows the FP to accept all calls from subscribed PPs by default.

When the connection has been established, the FP receives a signal of the type `APPLICATION_PRIMITIVE` with the sub event `APPCCSetupInd`.

- 3 You now add code to the signal handler, to respond to the signal.

The signal contains a PMID value that identifies the connection. The PMID value can be found in the first 3 bytes of the signal address (`Signal->Address`). The PMID is used for all further API calls. The PMID is comparable to a file descriptor in UNIX, only it designates a connection rather than a file.

Looking up the Speech Buffer Index

Once the connection has been established, the next step is to lookup the RX speech buffer index that is used by the connection. This is necessary to get hold of the RX data that is streamed through the wireless link to redirect it to go via the DSP and out the PCM interface.

To Lookup the RX Speech Buffer Index

- 1 Use the following function

```
UByte getSpeechBufferIndex(BYTE Pmid [3])
```
- 2 Convert the index ($x = \text{getSpeechBufferIndex}(pmid)$) one more time by looking up the following map.

```
extern const WORD BufferOffsetDefines[]
```
- 3 Lookup the speech buffer.

```
SpeechBufferID = BufferOffsetDefines[x+1]
```

By adding one, you get the RX buffer ID. (If you add 0, you get the ID of the TX buffer.

Connecting the RX Speech Buffer to the DSP

You can now begin to stream the ADPCM in the speech buffer into the DSP. The DIP can do this automatically, when the following instructions have been executed.

```
ExecOneBmcInstructionFP((BK_MA<<8) | ((SpeechBufferID &
0xf800) >>8)); set main bank for ADPCM
ExecOneBmcInstructionFP((BK_A<<8) | ((SpeechBufferID &
0x0700) >> 3)); set bank for ADPCM
ExecOneBmcInstructionFP((A_LDR<<8) | ((SpeechBufferID &
0x00ff)); set ADPCM 0 read pointer offset
ExecOneBmcInstructionFP((A_LDW <<8) | ((SpeechBufferID &
0x00ff) +40)); // set ADPCM 0 writer pointer offset
ExecOneBmcInstructionFP((A_RX <<8)|0); //use ADPCM 0
ExecOneBmcInstructionFP((A_TX<<8)|0); //use ADPCM 0
ExecOneBmcInstructionFP((FP((WORD) (A_NORM <<8)));
```

The content of the speech buffer that is used for this connection is streamed into the DSP.

Starting the DSP

See the template software for an example of how to start up the DSP and have it convert the ADPCM in the LPCM.

Refer to the `DSPSignalHandler.c` function `StartDSP`. You can call this function directly or send a signal with the event `STARTDSP_EVENT` to the DSP signal handler.

Connecting DSP Output to the PCM interface

The default DSP code that is supplied with the module, streams 14 bit LPCM samples at 8 KHz into the CODEC. This is used to get audio in the module headset.

In this example, we are going to stream into the PCM rather than to the CODEC.

To stream into the PCM, you reconfigure the default DSP code to stream into the DSP_PCM_OUT0_REG.

By default, the DSP destination is saved in the following:

```
extern WORD adpcm_CODEC_Dest_CODECDataOutDEST;
```

To change the destination to DSP_PCM_OUT0REG, do the following:

```
DSP_PARAM(adpcm_CODEC_Dest_CODECDataOutDEST) = 0xBF3;
```

The DSP now streams out the connected audio from the PP as 14 bit LPCM samples at 8 KHz.

When you use the methods above, you can stream 4 channels at the same time.

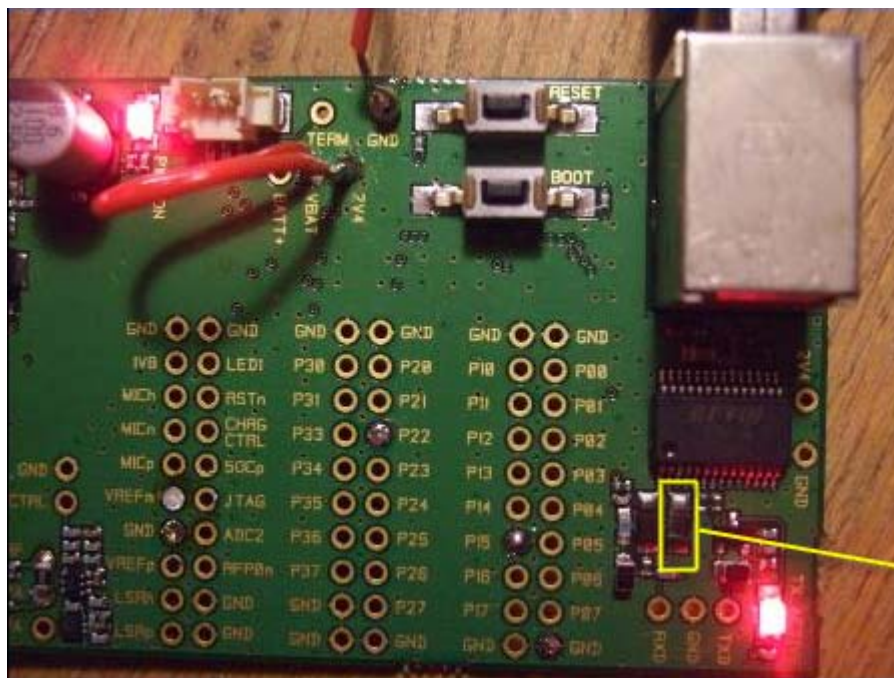
Troubleshooting Applications

The following section describes common questions and issues you may experience while working with the KIRK DECT Application Module 6.0.

My Board Enters Boot Mode Unintentionally

Because of an issue with the Windows USB driver, the board sometimes enters boot mode, and it can be difficult to get it out of boot mode again. If you experience this issue, unsolder the component outlines below.

Figure 7-1 Unsolder Components in Boot Mode



I Cannot Subscribe a Module PP/DECT Handset to a Module FP

If the subscription procedure fails, try the following.

- 1 Verify that the FP has a legally known ARI (the first 5 bytes in the EEPROM), and verify that this is the ARI you are searching for and attempting to subscribe to.
- 2 Use a Polycom DECT handset (not a KDAM 6.0) and verify that this handset can successfully subscribe to the FP.
- 3 If the handset cannot subscribe, verify that output appears from the module while the subscription takes place.
- 4 Set the verbosity level to 0x01.
- 5 If the database is full, try removing an entry from the database or clear the database completely by clearing the user subscription data in the EEPROM. See [“Configuring the KDAM 6.0”](#) on page 14.
- 6 Verify that the PP module you are trying to subscribe has a legal EPEI (the first 5 bytes in the EEPROM.) The first byte must be 0 (zero).

A good FP ARI is, for example, 1013fd2500 (this ARI cannot be used as EPEI).

A good PP IPEI is, for example, 001c8f9620 (this IPEI can also be used as an ARI).

Two modules that run in the same environment must never have the same EPEI/ARI.

I cannot run the host.exe process inside Eclipse

To solve this issue, do the following.

- 1 Run the host.exe process from a command prompt, and then check the following.
 - a When you set up the launch configuration in Eclipse, the working directory and the arguments must match. If you use relative paths to the dump and ports file, they must be accurately given relative to the working directory Eclipse uses when launching the host process.
 - b When you set up the launch configuration, have Eclipse start the process from its own shell, rather than run the process from inside Eclipse.

Eclipse Cannot Invoke the GCC Compiler

If you experience that the build process cannot find the MinGW/gcc compiler on your D:/drive, but keeps looking on C:/, do the following.

- 1 Verify that you have specified the GCC install directory as specified in “To Install and Activate the EKGAP plug-in” on page 12.
Window->Preferences->IAR preferences->GCC install directory = D:/MinGW.
- 2 Search for <GCCInstallDir>C:\MinGW</GCCInstallDir> in the kt4585-host.ewp file, and replace with the correct path.
- 3 Click the build file to reload changes.

How do I Set Up a KDAM 6.0 PP to Subscribe to a KWS600v3?

- 1 On your module PP, change the IPEI to a legal PP IPEI.
- 2 On the PP, enable output, and then restart the PP.
- 3 When the PP starts up, it reports its ARI in both octal and hexadecimal format. To create a user in a KWS600v3, you need the octal format, including the leading zeros.

For example, if you change the IPEI to 001c8f9620 and then enable output and restart the PP, it reports the following: ([PP]:[RunMode = full]004561021472-001c8f9620) where the octal format is 004561021472 and the hexadecimal are 001c8f9620. (The hexadecimal format should match the input).

How Does Call Release Work?

When the remote party releases a call, the module receives `SS_RELEASE_prim`. This signal is of the type `SsReleasePrimType`. The IE can be cast to the type `ReleaseReasonType`. The `ReleaseReasoncode` indicates the release reason code. Release reason codes are defined in `sformat.h`.

How Do I send a Command or Data from the PC to the Module?

You can send mails (events) over UART from an external host processor such as a PC or an onboard micro controller to a task (event handler) that is installed in the module software. The data you send must be in the following format.

- 0: 0xff
- 1: 0xff
- 2: 0x4d

- 3: TaskID
- 4: Length
- 5: Length bytes of data

When the UART receives the data, the data is sent as a mail to the task with the task id TaskID.

How Do I Send Data from the Module to the PC?

By default, the module uses the UART to send debugging messages over the serial line. You can turn this off by setting the verbosity level to zero. You can then transmit data from the module over the UART by using the following function.

```
void CopyToUartTxBuffer(UByte * buffer, unsigned int length)
```

The UART TX buffer is a circular buffer of a fixed limited size. The UART TX interrupt empties the buffer. The UART is configured to run at 115200 baud.

The module also contains SPI and PCM interfaces that can be used for sending data to an external host processor.

How do I Enable Echo Cancellation?

You implement echo cancellation in the DSP. It is not currently part of the DSP code, but the DSP on the module supports it. The DSP is part of the application layer, and you can therefore extend the DSP.

How do I get the RSSI of the Locked Base?

You retrieve the The RSSI value of the base to which the PP is subscribed and locked by calling the following function.

```
UByte get_rssi_mess(void)
```

The function returns a byte that indicates the signal strength.

Setting up a Communication Channel

This section provides information about:

- Subscribing the KIRK DECT Application Module 6.0
- Subscribing handsets
- Setting up a data connection

Before you start, ensure that you have the following software installed:

- KIRK Virtual Handset
- LAN Manager
- Terminal Program

You find the software in the Tools folder.

Subscribing the KDAM 6

Before you can specify a data connection you must subscribe the KDAM 6.0 to the system using the KIRK Virtual Handset program. The KIRK Virtual Handset program provides a virtual keyboard and display.

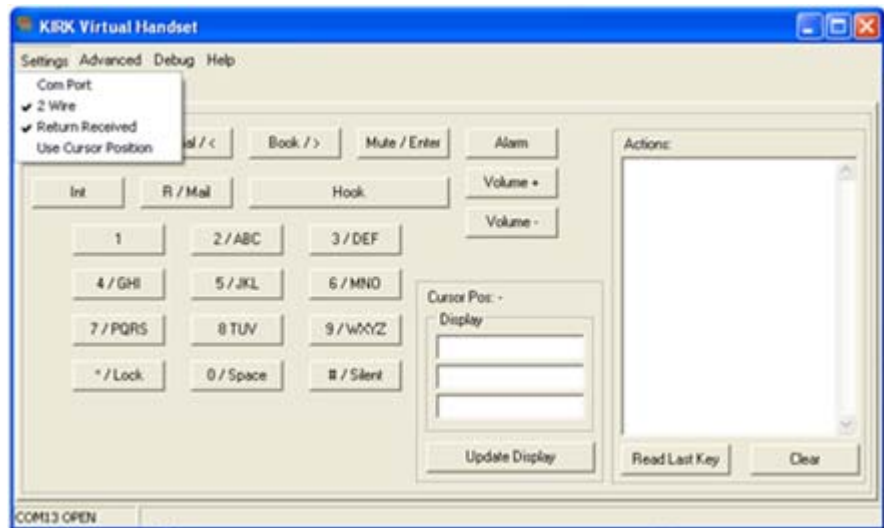
To make subscriptions, the system must allow subscriptions to be made. Some systems also require an Authentication Code (AC). Authentication Codes and system ID's (ARI codes) are provided by your system administrator.

You can subscribe the KDAM 6.0 automatically using the KIRK Virtual Handset program.

To subscribe the KDAM 6

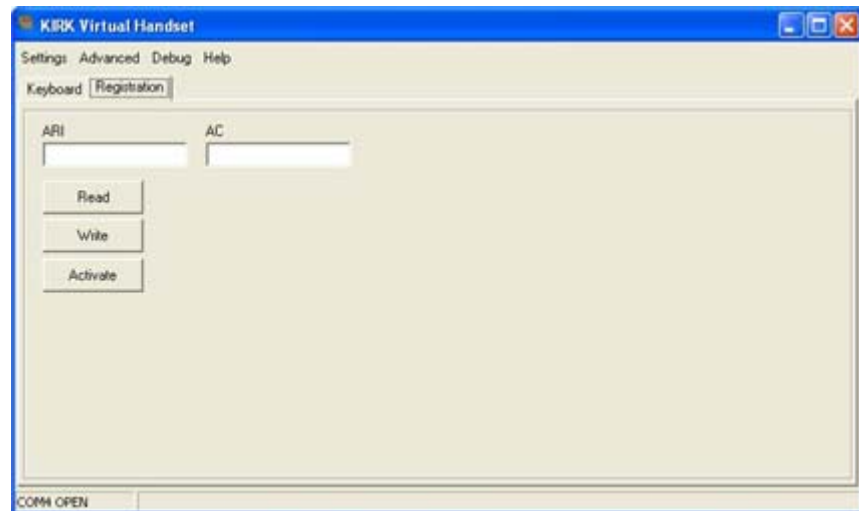
- 1 Open the Virtual Handset program, and then from the **Settings** menu, select **2 Wire** and **Return Received**.

Figure 8 Virtual Handset - Settings Menu



- 2 Click **COM port**, and then chose the preferred COM port from the list, and then click **Close** to close the COM port dialog.
- 3 Click the **Registration** tab.

Figure 9 Registration tab.



- 4 In the **ARI** field, type the ARI code of the system to which you want to subscribe. For further information, contact your system administrator.
- 5 In the **AC** field, type the authentication code. This field is optional.
- 6 Click **Write**.
- 7 Restart the KDAM 6.0.
- 8 Click **Activate**. The KDAM 6.0 subscribes automatically.

Subscribing handsets

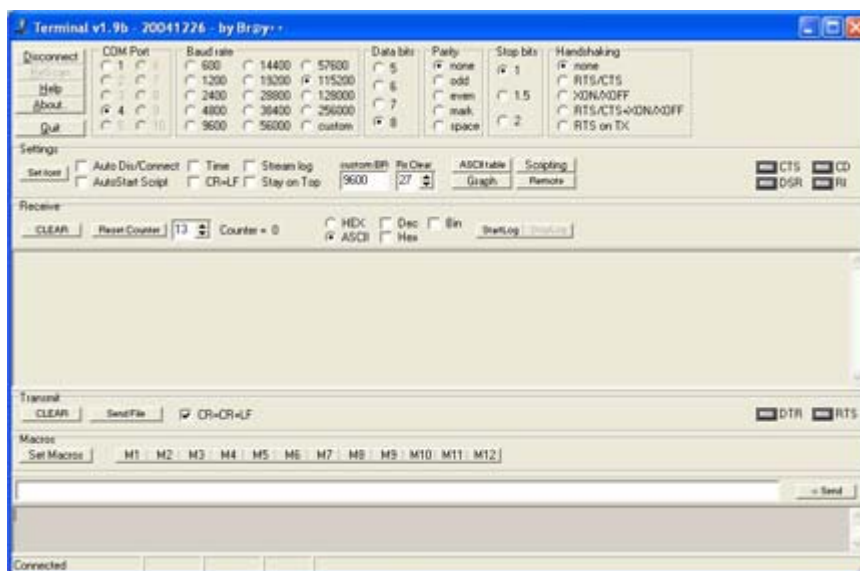
Before you can specify a data connection, you must subscribe the relevant handsets to the system. The KDAM 6.0 works with KIRK 40XX handsets and KIRK 50XX handsets. For information about how to subscribe the handsets, download the relevant user guides from www.polycom.com

To Setup a Data Connection

You use the Terminal program and the LAN Manager to set up the data connection.

- 1 Open the Terminal program
- 2 Enter the correct serial port settings (see the following figure), and then click **Connect**.

Figure 10 Serial Port Settings



- 3 Press the **M1** button to send An AT command to the KT4585. An **OK** message is displayed.
- 4 Start the LAN manager.
- 5 Enter the User Name and Password that correspond to the base station, and then click **Connect**.

Table 1 Base Station Password

Base station	User Name	Password
KWS600v3	GW-DECT/MSF/admin	ip1200
KWS6000	GW-DECT/admin	ip6000
KWS300	GW-DECT/admin	ip6000

- 6** In the **Hex String** field, type the ATD, and then press **Send**. The Terminal program displays RING.
- 7** In the Terminal Program, click the **M2** button to send ATA to the base station.

The base station displays ATA (61 at the end), and the Terminal Program displays CONNECT. You are now working in binary mode.
- 8** Click the **M5** button to send binary data to the base station. You can now send binary data to the terminal.
- 9** Click the **M4** button to exit binary mode.
- 10** Click the **M1** button. An **OK** message is displayed.
- 11** Click **M3** button to close the connection.

Communication Between KWS and a Third-Party Application

This chapter is a reference section which provides information about the communication between a KIRK Wireless Server (KWS) and the third-party application.

Using the MSFDLL.DLL

Communication from the third-party application to the KWS is established through the MSFDLL.DLL library. The MSFDLL.DLL is a library of functions that offers you an interface to connect, manage and disconnect the KWS.

The following table provides an overview of the DLL functions and the purpose of each.

Table 2 MSFDLL.DLL Functions

Function	Description
int msf_init	Initiates the connection.
void msf_xmit	Transmits a message to the target
int msf_rcv	Polls for a message and returns when the target has a message to send.
int msf_end	Closes the connection.

MSFDLL.DLL Reference

All MSFDLL.DLL functions are described in detail in the following.

Initialize the Connection

The function initiates the connection between the KWS and the third-party application.

Syntax

```
int msf_init(char* gw_name, char*user, char*password)
```

Parameters

*char*gw_name*

The IP address of the Kirk Wireless Server

*char*user*

The user name for the target including the front-header "GW-DECT/", for example "GW-DECT/admin".

*char*password*

The KIRK Wireless Server password

Return Value

Returns an integer value. If no msf-context (connection) is currently running, then an msf-context is created with the given parameters. Returns 2 if the connection is successful (correct user name/password). Returns 0 if the connection fails. Returns 1 if a connection is already established.

Transmit Data

Syntax

```
void msf_xmit(char*msg, int length)
```

Parameters

*char*msgdata*

Data to be sent.

int length

Length of the message.

Return Value

Returns no value.

Receive Data

Syntax

```
int msf_rcv(char*msg, int size)
```

Parameters

*char*msg*

Pointer to a location in the memory where received messages are stored.
The message is in binary code as in the transmit function.

int size

Currently not in use

Return Value

Returns an integer value that indicates the length of the received message.
Returns -1 if the connection fails.

Close the Connection

Syntax

```
int msf_end(char*msg, int size)
```

Parameters

*char*msg*

Currently not in use

int size

Currently not in use

Return Value

Returns -1.

MSF Structure

The following table describes the structure of the MSF commands.

Table 3 Structure of MSF Commands

Field name	Number of bytes	Description
Event	1	Describes the type of event. It is always 0x21.
Subevent	1	Describes the type of subevent. There are two types: <ul style="list-style-type: none"> 0x38 - MSF commands from the KWS to the third-party application. 0x39 - MSF commands from the third-party application to the KWS.
Length	1	Specifies the length of the MSF (PP number, status type, data length, and data bytes)
PP number	x bytes	Is the number of the DECT module as ASCII string with NULL termination.
Status type	1	Contains information about the type of the data.
Data length	1	The length of the data bytes and data length.
Data	x bytes	The data as ASCII string with NULL termination. Since it is not possible to communicate with the KWS using AT commands, the first byte of the data indicates the command.

MSF Commands

The following describes the MSF command codes that are used for communication between the KIRK Wireless Server and the third-party application.

Dial

Table 4 Command code 'd'

Code	Description
'd'	Is equivalent to the ATD command.

Table 5 ATD command sent from the third-party application to the KWS

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	64

Table 6 ATD command sent from the KWS to the third-party application

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	38	06	31 30 32	00	12	64

Answer Connection

Table 7 Command code 'a'

Code	Description
'a'	Is equivalent to the ATA command

Table 8 ATA command sent from the third-party application to the KWS

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	61

Table 9 ATA command sent from the KWS to the third-party application

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	38	06	31 30 32	00	12	61

Hang Up

Table 10 Command code 'h'

Code	Description
'h'	Is equivalent to the ATH command.

Table 11 ATA command sent from the third-party application to the KWS

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	68

Table 12 ATH command sent from third-party application to the KWS

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	38	06	31 30 32	00	12	68

Hang Up Acknowledge

Table 13 Command code 'H'

Code	Description
'H'	Acknowledges the hang up command. It is sent only from the KWS to the third-party application.

Table 14 ATD command sent from the third-party application to the KWS.

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	48

Table 15 Command code 'b'

Code	Description
'b'	<p>Is used to send binary data to the KDAM 6.0. The first byte of data is the command code and must be equal to 0x68, which is the ASCII code of 'b'. The binary data is converted to ASCII with null termination. The following is an example of conversion (all values are in hexadecimal):</p> <p>Data before conversion 00 01 02 03 04</p> <p>Data after conversion 30 30 30 31 30 32 30 33 30 34 00</p> <p>NULL termination is added at the end.</p>

Table 16 Binary transfer command sent from third-party application to the KWS.

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	48

Table 17 Binary transfer command sent from the third-party application to the KWS.

Event	Subevent	Length	PP number	Null termination	Status type	Data
21	39	06	31 30 32	00	11	48

AT Commands

AT Commands

The AT Command firmware enables binary mode communication with the PC application.

Table 1 *cAT Command Overview*

Command /Syntax	Action
AT	Replies with the status of the device
ATA	Answers call
ATH	Hangs up
ATE _x	Echo, x=0 -off, x=1 - on
ATD	Dials
ATO	Returns to data transmission from AT-mode when online.
+++	Escapes binary mode

Note: Each command must end with CR+LF characters.

AT Command Reference

The following reference section describes the commands in detail

Status

Syntax

AT

Reply

OK

Description

Is used to test the communication with the KDAM 6.0

Answer Call**Syntax**

ATA

Reply

CONNECT

Description

Connects the KDAM 6.0 to the KIRK Wireless Server. The connection is established when the KWS replies with ATA command. When the module receives ATA commands, it sends CONNECT to the third-party application. If the KWS initializes the connection, then the KDAM 6.0 sends RING command to the third-party application. User responds with ATA command.

Hang Up**Syntax**

ATH

Reply

OK

Description

Disconnects the KDAM 6.0 from the KIRK Wireless Server.

ATEX**Syntax**

ATE0

ATE1

Reply

OK

Description

Enables or disables local echo of the commands. If the local echo is enabled, all characters sent to the KDAM 6.0 will be sent back. Otherwise, only the replies will be sent to the third-party application.

Dial

Syntax

ATD

Reply

CONNECT

Description

Connects the KDAM 6.0 to the KIRK Wireless Server. The connection is established when the KIRK Wireless Server replies with ATA command. When the module receives ATA commands it sends CONNECT command to the third-party application. If the KIRK Wireless Server initializes the connection, the KT4584 sends RING command to the application. User responds with ATA command.

Switch to Binary Mode

Syntax

ATO

Reply

No reply

Description

If a binary connection has been established and the mode of transmission changed to AT commands, the ATO command can switch back to binary mode.

Escape Binary Mode

Syntax

+++

Reply

OK

Description

If the binary transmission is enabled, it is possible to switch to command mode without disconnecting by sending the +++ command. The ATO command switches back to binary mode.

Tables

Table 1	Acronyms	9
Table1	EEPROM parameter	15
Table 1	Base Station Password	42
Table 2	MSFDLL.DLL Functions	44
Table 3	Structure of MSF Commands	47
Table 4	Command code 'd'	47
Table 5	ATD command sent from the third-party application to the KWS ..	48
Table 6	ATD command sent from the KWS to the third-party application ..	48
Table 7	Command code 'a'	48
Table 8	ATA command sent from the third-party application to the KWS ..	48
Table 9	ATA command sent from the KWS to the third-party application ..	48
Table 10	Command code 'h'	48
Table 11	ATA command sent from the third-party application to the KWS ..	48
Table 12	ATH command sent from third-party application to the KWS	49
Table 13	Command code 'H'	49
Table 14	ATD command sent from the third-party application to the KWS. ..	49
Table 15	Command code 'b'	49
Table 16	Binary transfer command sent from third-party application to the KWS.	49
Table 17	Binary transfer command sent from the third-party application to the KWS.	49
Table 1	cAT Command Overview	50

Figures

Figure 7-1	Unsolder Components in Boot Mode	36
Figure 8	Virtual Handset - Settings Menu	41
Figure 9	Registration tab.	41
Figure 10	Serial Port Settings	42
Figure A-1	System Overview	56
Figure A-2	System Example	57

Index

A

AT commands 50

B

Boot mode 36

D

Data connection 42

G

GCC compiler 37

H

Hosted environment 23

K

KWS

Communicating with 44

M

Module FP

Subscribing 18

Module PP

Subscribing 19

MSF

Sending to module PP 20

MSF commands 47

MSF DLL 44

MSF structure 47

R

Recording scenarios 22

Replaying traces 23

Run in hosted environment 23

S

Subscribing handsets 42

Subscribing the module 40

T

Tracing 22

V

Voice connection 20

Diagrams

This appendix contains diagrams of how the module is used in a system.

Figure A-1 System Overview

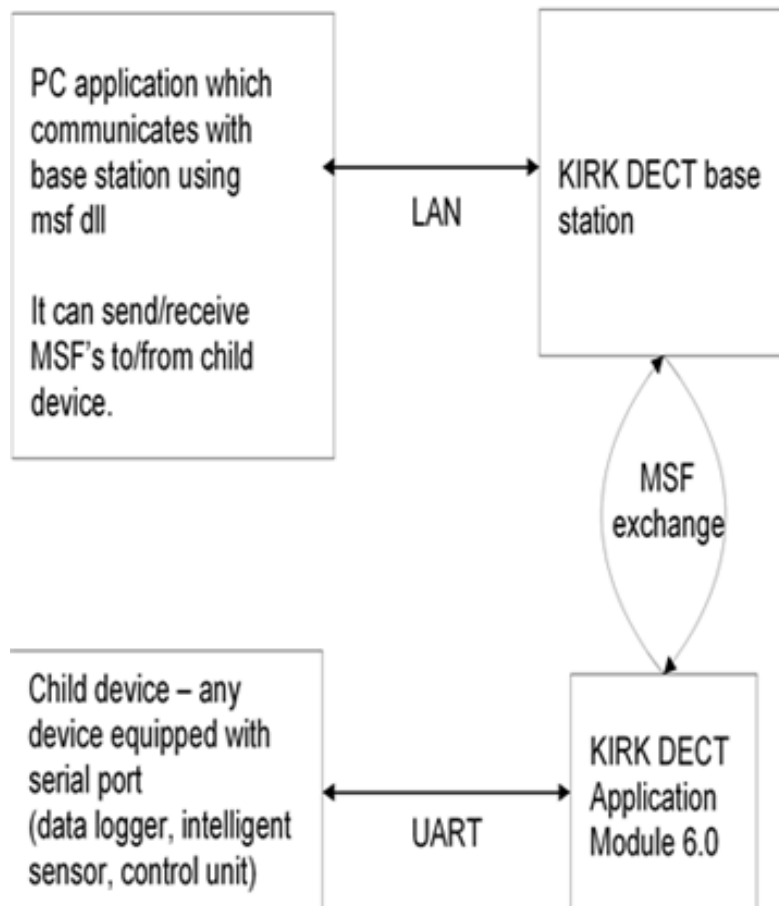


Figure A-2 System Example

