UM10277_1

TED-Kit 2 Programmer's Manual

Rev. 1.29 — 22 June 2011 **User manual**

Document information

Info	Content
Keywords	TED-Kit 2, Programmer’s Manual
Abstract	This document describes step-by-step how to write software for the TED-Kit 2 and its components using the API Library.

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Document Purpose

The purpose of this document is to describe how to write software which uses the TED-Kit 2 and its components including base stations and transponders. The intended audiences are software engineers planning to create their own custom software incorporating the TED-Kit 2.

All examples shown in this manual are written in C, C++ or C#. Although not tested, no problems are expected to use other programming languages capable of using functionality dynamically linked (DLL) during run-time. Such languages are Objective-C, Java, Visual Basic or even scripting languages like Perl.

The development environment used to develop and run the examples is Microsoft Visual C++ 2005 Express. It is available from Microsoft free of charge. Nevertheless, any other development environment for C++ running on Windows is expected to work because no features specific to that development software are used.

1.1 What this Document is Not

This document will not explain how NXP's immobilizers, Remote- or Passive Keyless Entry transponders and LF- or UHF base station products work. For a detailed explanation about e.g. the configuration settings, state machines or timings refer to the appropriate data sheet.

2. Introduction

This section will explain the software and hardware required to program and run software for the TED-Kit 2 as well as the basic concepts of the API Library usage.

2.1 Requirements

The following ingredients are required to write and later run the software for the TED-Kit 2 system:

Table 1. Required Ingredients

Item	Description	Type
	Compiling	
tk2.lib	The API library stub, required to compile the software.	SW
C/C++ Header Files	The header files containing the class and function declarations of the TED-Kit 2 Library (for C and C++ programming).	SW
TED-Kit 2 API.cs	C# library wrapper (for C# programming).	SW
	Executing	
tk2.dll	The TED-Kit 2 library, required to run the software.	SW
ftd2xx.dll	The FTDI driver library, required to run the software.	SW
FTDI Driver	The FTDI driver (installed on the host system).	SW
TED-Kit 2	The TED-Kit 2 hardware including an XBoard (e.g. an ABIC1).	HW
Transponder	The transponder to communicate with.	HW
tk2.dll	The TED-Kit 2 library, required to run the software.	SW

The search path for the include-files of the C/C++ compiler needs an entry pointing to the root of the include file/folder structure. The C/C++ linker needs to be configured to link the DLL-stub tk2.lib to the software.

3. “Hello World” in C

This section will explain a fully working example written in C to illustrate the basic steps required to make use of the TED-Kit 2 library functionality. To be specific, this sample application will show how to:

1. Retrieve a list with all connected NXP TED-Kit 2 devices
2. Open a single device and retrieve detailed information.
3. Read and show the firmware version of that TED-Kit 2 device.
4. Clean-up and close the device.

All explanations refer to the example code in:

[TED-Kit 2 installation]\Development\API\doc\examples\example.c

3.1 Epilog

The epilog of this C example program is rather simple. It includes all the declarations necessary to access the TED-Kit 2 library:

```
1  #include "intfs\IphcsApiInt\inc\phIcsApiInt.h"
2  #include "types\phTedKitStatus.h"
3  #include "types\phTedKitCommands.h"
```

3.2 Initialization

The actual functionality will be found in the main function of the example. The very first step is to declare a handle to the TED-Kit 2 Library to actually be able to use its functionality:

```
32  void *api = phcsApiInt_Alloc();
```

To exchange data between this application and the TED-Kit 2 library, an instance each of the data structures is required:

```
34  phTedKit_IoData_t ioData;
    …
36  phTedKit_BaseData_t baseData;
```

3.2.1 Count FTDI Devices

To retrieve the number of all connected FTDI devices, the function GetDeviceNumber (see section 10.7, page 44) of the API’s I/O layer is called. To do that, the `ioData` structure is prepared by assigning the function’s ID to the `Function` attribute of the structure.

```
39  ioData.Function = PHTEDKITIOFKT_GET_DEVICE_NUMBER;
```

Afterwards, the `Run` method of the API is called:

```
41  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
```

The first parameter is the handle of the API instance. The second parameter indicates an I/O layer call and the third parameter is the reference to the data structure. If successful, the user is informed about the number of devices found by printing the content of the `DeviceNum` attribute of the `ioData` structure:

```
47  printf("Number of devices found: %d.\n\n", ioData.DeviceNum);
```

To ensure proper operation of the software, it will stop if no devices were found.

3.2.2 Get Device Details

The next step is to show the device details and determine whether the FTDI devices are actually NXP TED-Kit 2 devices. This will be done by evaluating the vendor and product ID of each FTDI device connected.

This example application will use the first NXP TED-Kit 2 device out of all FTDI devices found for all further actions. To find that device, a loop over all available devices is created.

To actually retrieve the device details, the API function named GetDeviceInfoDetail (see section 10.6, page 42) out of the I/O layer is used. The `ioData` structure is prepared accordingly:

```
58  ioData.Function = PHTEDKITIOFKT_GET_DEVICE_INFODETAIL;
```

Now, the loop over all available devices can run and repeat that function call for each of them. The loop's body will set the device/port number and than actually call the run method with the proper parameters:

```
59  for (i = 0; i < (int) ioData.DeviceNum; i++) {
60    ioData.Port = i;
61    phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
      …
81  }
```

After calling the `Run` function, its `StatusCode` attribute of `ioData` is evaluated. Depending on the result, either the device information or an error message is shown. In case of success, the device information is printed by just listing all the items which are available:

```
63  printf("Number       : %d\n", i);
64  printf("Flags        : %d\n", ioData.Flags);
65  printf("               -> opened=%s\n",
66      ((ioData.Flags & 1) ? "yes" : "no"));
67  printf("Type         : %d\n", ioData.Type);
68  printf("DeviceID     : 0x8X\n", ioData.ID);
69  printf("LocId        : %d\n", ioData.LocId);
70  printf("SerialNumber : %s\n", ioData.SerialNumber);
71  printf("Description  : %s\n", ioData.Description);
```

After showing the device information, the vendor-and-product-ID is compared with the one expected by NXP TED-Kit 2 devices. In case the IDs match, the index is kept for further processing.

```
74  if ((ioData.ID == PHTEDKITUSB_VIDPID) && (index == -1)) {
      …
76  }
```

After the loop has finished, the device being used is shown. If no proper device was found, the software shows an error message and aborts. Now, the application knows what devices are detected and which one to use for the remaining steps.

3.2.3 Open Device

The final step of the initialization section is to open the device for sending and receiving data:

```
92  ioData.Port = index;
93  ioData.Function = PHTEDKITIOFKT_OPEN;
94  ioData.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
95  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
```

First, the `ioData` structure is properly populated: The `Port` attribute specifying the FTDI device number gets the value of the first NXP TED-Kit 2 device found. The function called is Open (see section 10.10, page 47) and the `OpenMode` is set to normal (`PHTEDKITIOFKT_NORMAL_OPERATION`).

After the call, the `StatusCode` attribute of the `ioData` structure is evaluated and in case of failure, a message is printed and the application is aborted.

3.3 Calling API Functions

This example will call only one function: GetFWVersion (see section 11.9, page 61) out of the BaseApi. The instance of `phTedKit_BaseData_t` is populated with the proper data required for the call. In case of this simple function, only the function name needs to be set:

```
105  baseData.Function = PHTEDKITBASEAPIFKT_GETFWVERSION;
```

The method `Run` is called afterwards:

```
106  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

To finish that call, the `StatusCode` attribute is evaluated and eventually an error message is shown. In case of success, the returned values stored in the same instance of the `baseData` structure are processed:

```
108  printf("Firmware Code    : %d\n", baseData.RxData1);
109  printf("Firmware Version : %d.%d\n",
110      baseData.RxData2, baseData.RxData3);
```

3.4 Clean-Up

After executing all the functions required, the device in use shall be closed by calling the API function Close (see section 10.1, page 36):

```
121  ioData.Function = PHTEDKITIOFKT_CLOSE;
122  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
```

The `ioData` structure is properly populated and the `Run` function is called. After evaluating the `StatusCode` (and eventually showing an error message), the final step – freeing the resources – is done.

```
129  phcsApiInt_Destroy(api);
```

4. “Hello World” in C++

*The C++ interface of the TED-Kit 2 library as described in this document is obsolete and shall not be used anymore. Instead, the **TED-Kit 2 Foundation Classes** library shall be used.*

This is a class library build on top of the TED-Kit 2 library and greatly simplifies the programming of the TED-Kit 2. It is available from the same source as this TED-Kit 2 software package.

This section will explain a fully working example written in C++ to illustrate the basic steps required to make use of the API Library functionality. To be specific, this sample application will show how to:

1. Retrieve a list with all connected NXP TED-Kit 2 devices
2. Open a single device and retrieve detailed information.
3. Read and show the firmware version of that TED-Kit 2 device.
4. Clean-up and close the device.

All explanations refer to the example code in:

[TED-Kit 2 installation]\Development\API\doc\examples\example.cpp

4.1 Epilog

As any C++ program, this one also has an epilog section including function- and constant declarations:

```
1  #include "comps\phcsApiInt\inc\phcsApiInt.hpp"
2  #include "types\phTedKitStatus.h"
3  #include "types\phTedKitCommands.h"
```

4.2 Initialization

The actual functionality will be found in the main function of this example program. The very first step is to define an instance of the API Library to actually be able to use its functionality:

```
34  phcs_TedKit2::ApiInt api;
```

To exchange data between this application and API Library, an instance of the data structure specified for that purpose is required:

```
37  phcs_TedKit2::phTedKit_IoData_t ioData;
```

4.2.1 Count FTDI Devices

To retrieve the number of all connected FTDI devices, the function GetDeviceNumber (see section 10.7, page 44) of the API’s I/O layer is called. To do that, the `ioData` structure is prepared by assigning the function’s ID to the `Function` attribute of the structure.

```
40  ioData.Function = phcs_TedKit2::PHTEDKITIOFKT_GET_DEVICE_NUMBER;
```

Afterwards, the `Run` method of the API is called:

```
42  api.Run(PHTEDKITCOMPID_IO, &ioData);
```

The first parameter indicates an I/O layer call; the second parameter is the address of the data structure. If successful, the user is informed about the number of devices found by printing the content of the `DeviceNum` attribute of the `ioData` structure:

```
48  std::cout << "Number of devices found: " << ioData.DeviceNum << "."
49      << std::endl << std::endl;
```

To ensure proper operation of the software, it will stop if no device were found.

4.2.2 Get Device Details

The next step is to show the device details and determine whether the FTDI devices are actually NXP TED-Kit 2 devices. This will be done by evaluating the vendor and product ID of each FTDI device connected.

This example application will use the first NXP TED-Kit 2 device out of all FTDI devices found for all further actions. To find that device, a loop over all available devices is created.

To actually retrieve the device details, the API function named GetDeviceInfoDetail (see section 10.6, page 42) out of the I/O layer is used. The `ioData` structure is prepared accordingly:

```
59  ioData.Function = phcs_TedKit2::PHTEDKITIOFKT_GET_DEVICE_INFODETAIL;
```

Now, the loop over all available devices can run and repeat that function call for each of them. The loop's body will set the device/port number and than actually call the run method with the proper parameters:

```
60  for (int i = 0; i < static_cast<int>(ioData.DeviceNum); i++) {
61    ioData.Port = i;
62    api.Run(PHTEDKITCOMPID_IO, &ioData);
      …
84  }
```

After calling the `Run` function, its `StatusCode` attribute of `ioData` is evaluated. Depending on the result, either the device information or an error message is shown. In case of success, the device information is printed by just listing all the items which are available:

```
64  std::cout
65      << "Number       : " << i << std::endl
66      << "Flags        : " << ioData.Flags << std::endl
67      << "               -> opened="
68      << ((ioData.Flags & 1) ? "yes" : "no") << std::endl
69      << "Type         : " << ioData.Type << std::endl
70      << "DeviceID     : 0x" << std::hex << std::uppercase << ioData.ID
71          << std::dec << std::endl
72      << "LocId        : " << ioData.LocId << std::endl
73      << "SerialNumber : " << ioData.SerialNumber << std::endl
74      << "Description  : " << ioData.Description << std::endl
75      << std::endl;
```

After showing the device information, the vendor-and-product-ID is compared with the one expected by NXP TED-Kit 2 devices. In case the IDs match, the index is kept for further processing.

```
77  if ((ioData.ID == PHTEDKITUSB_VIDPID) && (index == -1)) {
      …
79  }
```

After the loop is finished, the device being used is shown. If no proper device was found, the software shows an error message and aborts. Now, the application knows what devices are detected and which one to use for the remaining steps.

4.2.3 Open Device

The final step of the initialization section is to open the device for sending and receiving data:

```
96  ioData.Port = index;
97  ioData.Function = phcs_TedKit2::PHTEDKITIOFKT_OPEN;
98  ioData.OpenMode = phcs_TedKit2::PHTEDKITIOFKT_NORMAL_OPERATION;
99  api.Run(PHTEDKITCOMPID_IO, &ioData);
```

First, the `ioData` structure is properly populated: The `Port` attribute specifying the FTDI device number gets the value of the first NXP TED-Kit 2 device found. The function called is Open (see section 10.10, page 47) and the `OpenMode` is set to normal (`PHTEDKITIOFKT_NORMAL_OPERATION`).

After the call, the `StatusCode` attribute of the `ioData` structure is evaluated and in case of failure, a message is printed and the application is aborted.

4.3 Calling API Functions

This example will call only one function: GetFWVersion (see section 11.9, page 61) out of the BaseApi. To achieve this, an instance of the proper data structure for BaseApi calls is required:

```
107  phcs_TedKit2::phTedKit_BaseData_t baseData;
```

Now, that instance can be populated with the proper data required for the call. In case of this simple function, only the function name needs to be set:

```
111  baseData.Function = phcs_TedKit2::PHTEDKITBASEAPIFKT_GETFWVERSION;
```

The method `Run` is called afterwards:

```
112  api.Run(PHTEDKITCOMPID_BASEAPI, &baseData);
```

To finish that call, the `StatusCode` attribute is evaluated and eventually an error message is shown. In case of success, the returned values stored in the same instance of the `baseData` structure are processed:

```
114  std::cout
115    << "Firmware Code    : " << baseData.RxData1 << std::endl
116    << "Firmware Version : " << baseData.RxData2 << "."
117    << baseData.RxData3 << std::endl;
```

4.4 Clean-Up

After executing all the functions required, the device in use shall be closed by calling the API function Close (see section 10.1, page 36):

```
127  ioData.Function = phcs_TedKit2::PHTEDKITIOFKT_CLOSE;
128  api.Run(PHTEDKITCOMPID_IO, &ioData);
```

The `ioData` structure is properly populated and the `Run` function is called. After evaluating the `StatusCode` (and eventually showing an error message), the application is closed. This will trigger the destructor of the API class which releases all resources used for the TED-Kit 2.

5. “Hello World” in C#

This section will explain a fully working example written C# to illustrate the basic steps required to make use of the API Library functionality. To be specific, this sample application will show how to:

1. Retrieve a list with all connected NXP TED-Kit 2 devices
2. Open a single device and retrieve detailed information.
3. Read and show the firmware version of that TED-Kit 2 device.
4. Clean-up and close the device.

All explanations refer to the example code in:

[TED-Kit 2 installation]\Development\API\doc\examples\example.cpp

In contrast to the C and C++ examples shown in the previous sections, C# cannot directly access the library functions and definitions. To be able to write software in C#, a wrapper class is required. It is provided as part of the installation of this software:

[TED-Kit 2 installation]\Development\API\doc\examples\TED-Kit 2 API.cs

This class is required to run all examples shown in this manual.

5.1 Epilog

The epilog of this C# example program is rather simple. The use of the `System` and the `phcs_TedKit2` namespaces is declared. The example class itself is put into its own namespace:

```
  1  using System;
  2  using phcs_TedKit2;
  3
  4  namespace com.nxp.cai.tedkit2 {
       …
136  }
```

5.2 Initialization

The actual functionality will be found in the main function of the class Example. The very first step is to declare a handle to the TED-Kit 2 Library to actually be able to use its functionality:

```
 25  API api = null;
       …
 29  api = new API();
```

To exchange data between this application and the TED-Kit 2 library, an instance of the data structure specified for that purpose is required:

```
 32  API.IoData ioData = new API.IoData();
```

In this case, it’s an instance of the I/O data structure because that is needed first.

5.2.1 Count FTDI Devices

To retrieve the number of all connected FTDI devices, the function GetDeviceNumber (see section 10.7, page 44) of the API’s I/O layer is called. To do that, the `ioData` structure is prepared by assigning the function’s ID to the `Function` attribute of the structure.

```
35  ioData.Function = API.Function.PHTEDKITIOFKT_GET_DEVICE_NUMBER;
```

Afterwards, the Run method of the API is called:

```
36  api.Run(API.Layer.PHTEDKITCOMPID_IO, ioData);
```

The first parameter indicates an I/O layer call and the second parameter is the reference to the data structure. If successful, the user is informed about the number of devices found by printing the content of the DeviceNum attribute of the ioData structure:

```
42  Console.WriteLine("Number of devices found: "
43      + ioData.DeviceNum + "\n");
```

To ensure proper operation of the software, it will stop if no devices were found.

5.2.2 Get Device Details

The next step is to show the device details and determine whether the FTDI devices are actually NXP TED-Kit 2 devices. This will be done by evaluating the vendor and product ID of each FTDI device connected.

This example application will use the first NXP TED-Kit 2 device out of all FTDI devices found for all further actions. To find that device, a loop over all available devices is created.

To actually retrieve the device details, the API function named GetDeviceInfoDetail (see section 10.6, page 42) out of the I/O layer is used. The ioData structure is prepared accordingly:

```
55  ioData.Function = API.Function.PHTEDKITIOFKT_GET_DEVICE_INFODETAIL;
```

Now, the loop over all available devices can run and repeat that function call for each of them. The loop's body will set the device/port number and than actually call the run method with the proper parameters:

```
56  for (ushort i = 0; i < ioData.DeviceNum; i++) {
57    ioData.Port = i;
58    api.Run(API.Layer.PHTEDKITCOMPID_IO, ioData);
      …
79  }
```

After calling the Run function, its StatusCode attribute of ioData is evaluated. Depending on the result, either the device information or an error message is shown. In case of success, the device information is printed by just listing all the items which are available:

```
60  Console.WriteLine("No           : {0}", i);
61  Console.WriteLine("Flags        : {0}", ioData.Flags);
62  Console.WriteLine("Type         : {0}", ioData.Type);
63  Console.WriteLine("DeviceID     : {0:X}", ioData.ID);
64  Console.WriteLine("LocID        : {0}", ioData.LocId);
65  Console.WriteLine("SerialNumber : {0}",
66      API.toString(ioData.SerialNumber));
67  Console.WriteLine("Description  : {0}",
68      API.toString(ioData.Description));
```

After showing the device information, the vendor-and-product-ID is compared with the one expected by NXP TED-Kit 2 devices. In case the IDs match, the index is kept for further processing.

```
70  if ((ioData.ID == API.PHTEDKITUSB_VIDPID) && !found) {
      …
73  }
```

After the loop is finished, the device being used is shown. If no proper device was found, the software shows an error message and aborts. Now, the application knows what devices are detected and which one to use for the remaining steps.

5.2.3 Open Device

The final step of the initialization section is to open the device for sending and receiving data:

```
92  ioData.Port = index;
93  ioData.Function = API.Function.PHTEDKITIOFKT_OPEN;
94  ioData.OpenMode = API.PHTEDKITIOFKT_NORMAL_OPERATION;
95  api.Run(API.Layer.PHTEDKITCOMPID_IO, ioData);
```

First, the `ioData` structure is properly populated: The `Port` attribute specifying the FTDI device number gets the value of the first NXP TED-Kit 2 device found. The function called is Open (see section 10.10, page 47) and the `OpenMode` is set to normal (`PHTEDKITIOFKT_NORMAL_OPERATION`).

After the call, the `StatusCode` attribute of the `ioData` structure is evaluated and in case of failure, a message is printed and the application is aborted.

5.3 Calling API Functions

This example will call only one function: GetFWVersion (see section 11.9, page 61) out of the BaseApi. To achieve this, an instance of the proper data structure for BaseApi calls is required:

```
103  API.BaseData baseData = new API.BaseData();
```

Now, that instance can be populated with the proper data required for the call. In case of this simple function, only the function name needs to be set:

```
108  baseData.Function = API.Function.PHTEDKITBASEAPIFKT_GETFWVERSION;
```

The method `Run` is called afterwards:

```
109  api.Run(API.Layer.PHTEDKITCOMPID_BASEAPI, baseData);
```

To finish that call, the `StatusCode` attribute is evaluated and eventually an error message is shown. In case of success, the returned values stored in the same instance of the `baseData` structure are processed:

```
111  Console.WriteLine("Firmware Code    : {0}", baseData.RxData1);
112  Console.WriteLine("Firmware Version : {0}.{1}",
113    baseData.RxData2, baseData.RxData3);
```

5.4 Clean-Up

After executing all the functions required, the device in use shall be closed by calling the API function Close (see section 10.1, page 36):

```
121  ioData.Function = API.Function.PHTEDKITIOFKT_CLOSE;
122  api.Run(API.Layer.PHTEDKITCOMPID_IO, ioData);
```

The ioData structure is properly populated and the Run function is called. After evaluating the StatusCode (and eventually showing an error message), all resources are released:

```
129  api.dispose();
```

6. Handling Multiple Devices

The whole software system is able to handle multiple connected TED-Kit 2 devices at the same time. This chapter will illustrate how this shall be handled by custom software written in C. The strategy shown here can be applied to all other languages as well.

6.1 One-after-Another

The key to handle multiple devices one-after-another is to open and close each of the devices for using. The example code explained in section 3: “Hello World” in C uses only the very first TED-Kit 2 device found. The extensions necessary to handle all TED-Kit 2 devices connected are illustrated below.

Instead of opening just one device, the code will open each TED-Kit 2 device, ask for the firmware and close it. Thus, the open and close function calls are now inside the loop:

```
for (int i = 0; i < (int) ioData.DeviceNum; i++) {
  …
  if (ioData.ID == PHTEDKITUSB_VIDPID) {
    …
    ioData.Port = i;
    ioData.Function = PHTEDKITIOFKT_OPEN;
    ioData.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
    phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
    …
    ioData.Function = PHTEDKITIOFKT_CLOSE;
    phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
    …
  }
  …
}
```

The pointer `api` points to the TED-Kit 2 specified during the opening process. With the loop above, the example application ripples through all the TED-Kit 2 devices available, one per loop cycle.

6.2 In Parallel

If an application shall be able to handle multiple TED-Kit 2 devices in parallel, the sample shown in section 6.1 is of no use. The key to handle multiple devices in parallel is to create multiple instances of the TED-Kit 2 API library. For each device a new instance is created and an open-use-close cycle can be used independent from the other instances.

For the initial analysis of the connected devices one instance is required. After retrieving the number of devices, each device asks for the details and if it is a TED-Kit 2 device stored for later use:

```
void** tedKit2List = malloc(…);
…
if (ioData.ID == PHTEDKITUSB_VIDPID) {
  /* create a new instance of the ApiInt class */
  void* tedKit2 = phcsApiInt_Alloc();

  /* open the device */
  ioData.Port = i;
  ioData.Function = PHTEDKITIOFKT_OPEN;
  ioData.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
  …
  /* store reference in the list */
  tedKit2List[i] = tedKit2;
}
```

Afterwards, each device can be operated independent from each other by accessing it through the array index:

```
phcsApiInt_Run(tedKit2List[i], PHTEDKITCOMPID_BASEAPI, &baseData);
```

7. Interaction with a Transponder

This section will show detail how to setup and execute a communication sequence with a HITAG2-Extended (PCF7937EA) transponder. The following steps are executed and explained:

1. Finding a TED-Kit 2 with an ABIC1 on the USB
2. Enabling the TED-Kit 2
3. Configuring the ABIC1 XBoard
4. Configuring the data transmission (from ABIC1 to HITAG2-Extended)
5. Configuring the data reception (from HITAG2-Extended to ABIC1)
6. Reading the XMA configuration of the transponder.
7. Executing an authentication in ciphered mode (default secret key)
8. Selecting an appropriate XMA segment and block
9. Reading all pages from the block.
10. Writing a page of a block.
11. Read that page back.
12. Shut down.

All explanations refer to the example code in:

[TED-Kit 2 installation]\Development\API\doc\examples\PCF7937-in-out.c

In order to run this example, a TED-Kit 2 with an ABIC1 XBoard as well as a HITAG2-Extended transponder (PCF7937AS) has to be available. The transponder must be configured for ciphered authentication with the default secret key. The equalizer mode must be set to normal, The XMA configuration must contain at least 2 segments where the second segment is accessible in ciphered mode (read/write).

For details about the configuration, refer to the data sheet of the transponder.

7.1 Finding a TED-Kit 2 with an ABIC1

Finding a TED-Kit 2 with an ABIC1 on the USB is done similar to the previous examples. First, a TED-Kit 2 API is created:

```
173  api = phcsApiInt_Alloc();
```

Second, the number of FTDI/TED-Kit 2 devices is determined:

```
186  ioData.Function = PHTEDKITIOFKT_GET_DEVICE_NUMBER;
187  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
```

Third, a loop over all devices is defined:

```
214  for (i = 0; i < (int) ioData.DeviceNum; i++) {
       …
300  }
```

Inside that loop, the device details for each FTDI device are read and evaluated to find the actual TED-Kit 2s using function GetDeviceInfoDetail (see section 10.6, page 42):

```
213  ioData.Function = PHTEDKITIOFKT_GET_DEVICE_INFODETAIL;
     …
215  ioData.Port = i;
216  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
     …
221  if (ioData.ID == PHTEDKITUSB_VIDPID) {
       …
296  }
```

For each TED-Kit 2, the XSlot information is read in order to find a proper ABIC1 XBoard:

```
239  baseData.Function = PHTEDKITBASEAPIFKT_GETXSLOTINFO;
240  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

For each XBoard, it is checked whether it is an ABIC1 with an LF-Antenna connected:

```
244  if (   (baseData.RxData1 == PHTEDKITXBOARD_ABIC1)
245      && ((baseData.RxData5 & PHTEDKITXBOARD_FEAT_LF) != 0)
246      && (xSlotPort == -1)) {
247    xSlotPort = 0;
248  }
```

The first ABIC1 is used for the rest of the program. The TED-Kit 2 port ID is stored in `tedKit2Port`; the XSlot ID is stored in `xSlotPort`.

7.2 Enabling the TED-Kit 2

One the TED-Kit 2 and XBoard being used for the rest of the program has been determined; it can be finally opened for normal operation:

```
323  ioData.Port = tedkit2Port;
324  ioData.Function = PHTEDKITIOFKT_OPEN;
325  ioData.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
326  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
```

After successfully opened, the TED-Kit 2 tick time has to be retrieved to be able to correctly set all the timings afterwards. This is achieved using BaseApi function GetDeviceStatus (see section 11.8, page 59):

```
336  baseData.Function = PHTEDKITBASEAPIFKT_GETDEVICESTATUS;
337  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
     …
344  frequency = bytesToLong(baseData.RxBuf1, 6);
345  tickTime = 1000000.0 / frequency;
```

The bytes 6 to 9 of `RxBuf1` contain the frequency of the TED-Kit 2's µc in Hertz. The variable `tickTime` contains now the timing reference suitable for timings given in µs.

7.3 Configuring the ABIC1 XBoard

The configuration of the XBoard is done using the BaseApi's function SetXBoardConfig (see section 11.22, page 81):

```
358  baseData.Function = PHTEDKITBASEAPIFKT_SETXBOARDCONFIG;
359  baseData.Device = PHTEDKITEXTAPIDEVICE_ABIC1;
360  baseData.TxData1 = xSlotPort;
     …
362  baseData.TxData2 = 0;
     …
364  baseData.TxData3 = 13;
```

The XBoard used for communication is an ABIC1 and thus, all 14 configuration elements (0…13) will be set at once. The following values taken from [UM10278_1] are set:

Table 2. ABIC1 XBoard Configuration

Parameter	Value
Interface and Mode	Non-Filtered
Data Rate	10 µs
Demodulator Sampling Phase	44
Antenna Phase	0
Diagnosis	0
ABIC1 Configuration Register 0…3	7, 0, 0, 0
Test Mode	Off

For a detailed description of the meaning of these fields (ABIC1 Configuration Registers in particular), refer to the ABIC1 data sheet. The actual values are taken from the ABIC1's data sheet and filled in the `baseData` structure:

```
365  /* 1 byte "interface & mode" -> set to 0 -> "none filtered" */
366  baseData.TxBuf1[0] = 0;
367  /* set the data rate to 10 µs ticks */
368  longToBytes((long) (10 / tickTime), baseData.TxBuf1, 1);
369  /* demodulator sampling phase */
370  baseData.TxBuf1[5] = 0x2c;
371  /* antenna phase: ignore, set to zero */
372  baseData.TxBuf1[6] = 0;
373  /* ignore, set to zero */
374  baseData.TxBuf1[7] = 0;
375  /* ABIC1 config register #0 */
376  baseData.TxBuf1[8] = 7;
377  /* ABIC1 config register #1 */
378  baseData.TxBuf1[9] = 0;
379  /* ABIC1 config register #2 */
380  baseData.TxBuf1[10] = 0;
381  /* ABIC1 config register #3 */
382  baseData.TxBuf1[11] = 0;
383  /* test mode off -> 0 */
384  baseData.TxBuf1[12] = 0;
385  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
```

7.4 Configuring the Data Transmission

The data transmission parameters are configured using the BaseApi's function SetTransmissionParams (see section 11.20, page 78). The general configuration is shown in Table 3.

Table 3. HITAG2-Extended T_x Parameter

Parameter		Value
Global Inversion		Off
Preamble Length		0
Preamble Symbol Duration		0
Header Length		0
Header Symbol Duration		0
Trailer Length		0
Trailer Symbol Duration		0
Idle Level		0
Body Encoding		BPLM
Body Encoding Parameters	T_{Pulse}	48µs
	T_{Log0}	160µs
	T_{Log1}	224µs
	T_{Stop}	288µs

The actual values are taken from the transponder's data sheet and filled in the `baseData` structure:

```
401  baseData.Function = PHTEDKITBASEAPIFKT_SETTRANSMISSIONPARAMS;
402  baseData.TxData1 = xSlotPort;
403  /* Global Inversion -> off */
404  baseData.TxData2 = 0;
405  /* PreambleLength */
406  baseData.TxData3 = 0;
407  /* PreambleSymbolDuration */
408  baseData.TxTime1 = 0;
409  /* HeaderLength */
410  baseData.TxData4 = 0;
411  /* HeaderSymbolDuration */
412  baseData.TxTime2 = 0;
413  /* TrailerLength */
414  baseData.TxData5 = 0;
415  /* TrailerSymbolDuration */
416  baseData.TxTime3 = 0;
417  /* IdleLevel */
418  baseData.TxData6 = 0;
419  /* Tx Data Coding Type -> BPLM */
420  baseData.TxData7 = PHTEDKITCODING_BPLM;
421  /* T_Pulse -> 48µs */
422  longToBytes((long) (48 / tickTime), baseData.TxBuf1, 0);
423  /* T_Log0 -> 160µs */
424  longToBytes((long) (160 / tickTime), baseData.TxBuf1, 4);
425  /* T_Log1 -> 224µs */
426  longToBytes((long) (224 / tickTime), baseData.TxBuf1, 8);
427  /* T_Stop -> 288µs */
428  longToBytes((long) (288 / tickTime), baseData.TxBuf1, 12);
429  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

7.5 Configuring the Data Reception

The data reception parameters are configured using the BaseApi's function `SetReceptionParams` (see section 11.19, page 76). The general configuration is shown in Table 4:

Table 4. HITAG2-Extended R_x Parameter

Parameter		Value
Synchronization Mode		Time based
Synchronization Delay		1.33ms
Global Inversion		Off
Header Length		10 half bits
Header Pattern		0101010101_{bin}
Header Symbol Duration		128µs
Body Encoding		Manchester
Body Encoding Parameters	Symbol Duration T_{Unit}	128µs

The actual values are taken from the transponder's data sheet and filled in the `baseData` structure:

```
445  baseData.Function = PHTEDKITBASEAPIFKT_SETRECEPTIONPARAMS;
446  baseData.TxData1 = xSlotPort;
447  /* SyncMode -> time based */
448  baseData.TxData2 = 1;
449  /* SyncDelay -> 1330µs */
450  baseData.TxTime1 = (uint32_t) (1330 / tickTime);
451  /* Global Inversion -> off */
452  baseData.TxData3 = 0;
453  /* Header Length -> 10 half-bits */
454  baseData.TxData4 = 10;
455  /* Header Symbol Duration -> 128µs */
456  baseData.TxTime2 = (uint32_t) (128 / tickTime);
457  baseData.TxData5 = PHTEDKITCODING_MANCHESTER;
458  /* Body Symbol Duration -> 128 µs */
459  longToBytes((long) (128 / tickTime), baseData.TxBuf1, 0);
460  /* Header Pattern for Manchester/EQ -> 5540hex (10 half-bits) */
461  baseData.TxBuf2[0] = 0x55;
462  baseData.TxBuf2[1] = 0x40;
463  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

The HITAG2-Extended can handle two different header patterns: the default pattern 5 bits long (11111_{bin}) and the modified pattern 7 bits long (1111110_{bin}). The patterns are always Manchester encoded regardless of the encoding scheme for the payload. The length and the waveform of the pattern must be given already encoded as Manchester. Thus, the header length for the default pattern is set to 10 (10 half bits = 5 bits) in `TxData4`. The pattern in `TxBuf2` conforms to five One's encoded as Manchester (5540_{hex} = 0101010101_{bin}). The value is left aligned and thus, the last 4 of the 16 bits are ignored.

7.6 Reading the XMA Configuration

To read the transponder's XMA configuration, two steps are necessary. First, the transponder has to be set into XMA/CFG state. Second, the memory configuration has to be read for each segment.

7.6.1 Entering XMA/CFG

To enter the XMA/CFG state, the PHTEDKITHITAG2CMD_XMACFG command executed using TransmitReceive (see section 12.2.1, page 98) is used.

This command can be executed in the transponder's WAIT state only. To ensure the transponder is indeed in WAIT state, a hard-reset (125 kHz field off/on) is configured. Such a hard reset can be issued at any time by setting TxTime1 and TxTime2 of the baseData structure to proper values (other than zero):

```
485   baseData.TxTime1 = (uint32_t) (5000 / tickTime);
      …
487   baseData.TxTime2 = (uint32_t) (6640 / tickTime);
488   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_WAIT,
489       PHTEDKITHITAG2CMD_XMACFG, &baseData);
```

The timings set correspond to $T_{RESET, DURATION}$=5ms (TxTime1) and $T_{RESET, SETUP}$=6.64ms (TxTime2) of the PCF7937AS data sheet.

An alternative to the hard (field-) reset is the use of the command PHTEDKITHITAG2CMD_SOFTRESET as explained in section 7.7.1, page 24.

After successful execution, the transponder is in XMA/CFG state. To prevent a hard reset during the next ExtApi transmit-receive sequence, the two timing parameters have to be reset to 0:

```
497   baseData.TxTime1 = 0;
598   baseData.TxTime2 = 0;
```

7.6.2 Reading the Configuration

The second step is to actually read the XMA configuration from the transponder. The transponder has 8 segments, for each of them; the appropriate commands have to be issued. A loop is defined first:

```
506   for (i = 0; i < 8; i++) {
        …
535   }
```

To read the configuration for a segment, the segment number has to be given in TxBuf1 followed by the execution of the function TransmitReceive (see section 12.2.1, page 98) with command PHTEDKITHITAG2CMD_READCFG in state PHTEDKITHITAG2STATE_XMACFG:

```
508   baseData.TxBuf1[0] = i;
509   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMACFG,
510       PHTEDKITHITAG2CMD_READCFG, &baseData);
```

After successful execution, the segment configuration is stored in RxBuf1. The first entry contains the segment's access mode; the second entry contains the size (number of blocks):

```
517   printf(
518       "segment #%i configuration:  mode=0x%02X, size=%i blocks.\n",
519       i, (baseData.RxBuf1[0] & 0x0F), (baseData.RxBuf1[1] & 0x0F));
```

For a detailed description of RxBuf1's layout, refer to [1].

7.7 Executing a Ciphered Authentication

An authentication for a HITAG2-Extended is a two-step process. First, the init sequence is executed returning the transponder's IDE. Second, the actual authentication (either password or ciphered) is carried out.

7.7.1 Preparation

To successfully authenticate, the transponder commands have to be executed in the WAIT state of the transponder. From the previous section, the transponder is still in XMACFG state. To bring it back to the WAIT state, the function `TransmitReceive` (see section 12.2.1, page 98) is executed with transponder command `PHTEDKITHITAG2CMD_SOFTRESET` in transponder state `PHTEDKITHITAG2STATE_XMACFG`:

```
552  transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMACFG,
553      PHTEDKITHITAG2CMD_SOFTRESET, &baseData);
```

After the reset, the transponder needs some time to initialize itself. The timing $T_{init\ state}$ of a PCF7937AS is specified with 6.32ms. Before sending any other command to the transponder, we have to wait that time to ensure the transponder is listening again.

To delay, the build-in function Delay (see section 11.1, page 50) of the TED-Kit 2 is used. The timing value `TxTime1` is first converted to µs (6.32ms = 6320µs) and then converted in TED-Kit 2 system ticks (as the TED-Kit 2 requires it):

```
562  baseData.Function = PHTEDKITBASEAPIFKT_DELAY;
     …
564  baseData.TxTime1 = (uint32_t) (6320 / tickTime);
565  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

Because the `TxTime1` attribute of `baseData` is used as field reset configuration, it must be reset to 0 to prevent any field reset during the next transmit-receive sequence:

```
573  baseData.TxTime1 = 0;
```

7.7.2 Authentication Initialization

The first step of the authentication is the initialization returning the transponder's IDE. This is achieved using the function `TransmitReceive` (see section 12.2.1, page 98) with command `PHTEDKITHITAG2CMD_STARTAUTH` in state `PHTEDKITHITAG2STATE_WAIT`. To indicate the first part of the authentication sequence, `TxBuf1` is set to 0:

```
576  baseData.TxBuf1[0] = 0;
577  transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_WAIT,
578      PHTEDKITHITAG2CMD_STARTAUTH, &baseData);
```

The returned IDE is stored in RxBuf1 and is printed on the screen:

```
586  ide = bytesToLong(baseData.RxBuf1, 0);
587  printf("transponder IDE: 0x%08X\n", ide);
```

7.7.3 Authentication Execution

The second step of the authentication is the exchange of the secure items. To indicate that now the second step is executed, `TxBuf1` is set to 2 - indicating crypto mode:

```
594  baseData.TxBuf1[0] = 2;
```

Besides that, TxBuf1 is also filled with the IDE, the challenge (0 in this example) and the secret key (the default value in this example):

```
596     longToBytes(ide, baseData.TxBuf1, 1);
        …
598     baseData.TxBuf1[5] = 0;
599     baseData.TxBuf1[6] = 0;
600     baseData.TxBuf1[7] = 0;
601     baseData.TxBuf1[8] = 0;
        …
603     baseData.TxBuf1[9] = 'M';
604     baseData.TxBuf1[10] = 'I';
605     baseData.TxBuf1[11] = 'K';
606     baseData.TxBuf1[12] = 'R';
607     baseData.TxBuf1[13] = 'O';
608     baseData.TxBuf1[14] = 'N';
```

For details about the layout of TxBuf1, refer to [1]. To execute the second step, the function TransmitReceive (see section 12.2.1, page 98) with command PHTEDKITHITAG2CMD_STARTAUTH in state PHTEDKITHITAG2STATE_WAIT is used:

```
609   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_WAIT,
610       PHTEDKITHITAG2CMD_STARTAUTH, &baseData);
```

If successful, RxBuf1 contains the transponder password which can be used by the base station to ensure the intended transponder is authenticated:

```
617   printf("transponder password: 0x%06X\n",
618       (bytesToLong(baseData.RxBuf1, 0) & 0x00FFFFFF));
```

7.8 Selecting a XMA Segment and Block

In order to access the HITAG2-Extended's memory, the transponder has to move from the AUTHENT state in the XMA state. This is achieved using the function TransmitReceive (see section 12.2.1, page 98) with transponder command PHTEDKITHITAG2CMD_XMA in transponder state PHTEDKITHITAG2STATE_AUTHORIZED. This command also selects the desired segment (#1 in this example given in TxBuf1):

```
636   baseData.TxBuf1[0] = 1;
637   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_AUTHORIZED,
638       PHTEDKITHITAG2CMD_XMA, &baseData);
```

Selecting a block of the segment, the function TransmitReceive (see section 12.2.1, page 98) with transponder command PHTEDKITHITAG2CMD_SELBLOCK in state PHTEDKITHITAG2STATE_XMA is executed. The desired block is given in TxBuf1:

```
650   baseData.TxBuf1[0] = 0;
651   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMA,
652       PHTEDKITHITAG2CMD_SELBLOCK, &baseData);
```

7.9 Reading all Pages from a Block

Each block has 8 pages of 32-bit values. To read them, an appropriate loop is defined:

```
666   for (i = 0; i < 8; i++) {
        …
678   }
```

Inside the loop, each of the 8 pages is read one after another using function TransmitReceive (see section 12.2.1, page 98) with transponder command PHTEDKITHITAG2CMD_READPAGE in transponder state PHTEDKITHITAG2STATE_XMA (the transponder is still in XMA state):

```
668   baseData.TxBuf1[0] = i;
669   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMA,
670       PHTEDKITHITAG2CMD_READPAGE, &baseData);
```

After successful reading, the value returned in RxBuf1 from the TED-Kit 2 is printed on the screen:

```
677   printf("page #%i := 0x%08X\n", i, bytesToLong(baseData.RxBuf1, 0));
```

7.10 Writing a Page of a Block

To write a page, the page number and its value have to be given in TxBuf1 (page 7 and a value of 76543210_{hex} are used in this example):

```
688   baseData.TxBuf1[0] = 7;
      …
690   longToBytes(0x76543210, baseData.TxBuf1, 1);
```

The function TransmitReceive (see section 12.2.1, page 98) is executed with transponder command PHTEDKITHITAG2CMD_WRITEPAGE in transponder state PHTEDKITHITAG2STATE_XMA (the transponder is still in XMA state):

```
692   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMA,
          PHTEDKITHITAG2CMD_WRITEPAGE, &baseData);
```

The transponder needs some time to actually carry out the memory write operation. The timing T_{Prog} of a PCF7937AS is specified with 4.92ms. Before sending any other command to the transponder, we have to wait that time to ensure the transponder is listening again.

To delay, the build-in function Delay (see section 11.1, page 50) of the TED-Kit 2 is used. The timing value TxTime1 is first converted to µs (4.92ms = 4920µs) and then converted in TED-Kit 2 system ticks (as the TED-Kit 2 requires it):

```
702   baseData.Function = PHTEDKITBASEAPIFKT_DELAY;
      …
704   baseData.TxTime1 = (uint32_t) (4920 / tickTime);
705   phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
```

Because the TxTime1 attribute of baseData is used as field reset configuration, it must be reset to 0 to prevent any field reset during the next transmit-receive sequence:

```
713   baseData.TxTime1 = 0;
```

7.11 Read that Page Back

To read that page 7 just written back from the transponder and e.g. check for correctness, the function TransmitReceive (see section 12.2.1, page 98) is executed with transponder command PHTEDKITHITAG2CMD_READPAGE in transponder state PHTEDKITHITAG2STATE_XMA (the transponder is still in XMA state):

```
724   baseData.TxBuf1[0] = 7;
725   transmitReceive(api, xSlotPort, PHTEDKITHITAG2STATE_XMA,
```

```
726       PHTEDKITHITAG2CMD_READPAGE, &baseData);
```

The result is stored in `RxBuf1` and printed on the screen:

```
733   printf("page #7 := 0x%08X\n", bytesToLong(baseData.RxBuf1, 0));
```

7.12 Shut Down

To properly shut down the application, the following steps are necessary. First, the TED-Kit 2 device is closed:

```
741   ioData.Function = PHTEDKITIOFKT_CLOSE;
742   phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &ioData);
743   if (ioData.StatusCode != PHTEDKITSTATUS_OK) {
744     printf("Unable to close device.\n");
745     return EXIT_FAILURE;
746   }
```

Second, the TED-Kit 2 library API resources are freed.

```
749   phcsApiInt_Destroy(api);
```

8. Transmit/Receive Logging

All functions dealing with transmission and reception (see Table 5, page 28) of data to and from transponders log the actual bits send or received. These logging data can be used to visualize the data stream or analyze it for debugging purposes.

Table 5. Functions for T_x/R_x

Function	API Layer	Page
TransmitReceive	Base API	83
TransmitReceive	Extension API	98
PkeAuthent	Extension API	104
PkePollEnable	Extension API	106
PkePollIde	Extension API	107
PkePollMute	Extension API	109
PkeReadEeprom	Extension API	111
PkeReadVbat	Extension API	113
PkeRssiAll	Extension API	115
PkeRssiSingle	Extension API	117
PkeWriteEeprom	Extension API	119

In case of a ciphered communication, the bits logged are already encrypted or not yet decrypted.

In case of communication failures, the logging buffers still may contain data. For example if a 10 bit answer was expected but only 9 bits received, the nine bits are in the logging buffer (RxBuf3 in this case).

8.1 Storage and Format

The logging data are stored in TxBuf3 and RxBuf3 of the baseData structure. The data are stored separately for transmitted and received bits. For each transmit/receive sequence, a data record is stored in both TxBuf3 (containing the transmission part) and RxBuf3 (containing the received part).

Because there are several transponder commands with more than one transmit/receive sequence (e.g. authentication, page writing etc), the data are stored in a way to handle an arbitrary number of sequences.

Table 6, page 29 illustrates how the data are stored. The format applies to both buffers, TxBuf3 and RxBuf3.

Table 6. Logging Data Storage Format

Byte of buffer	0	1	2	3	…	x	x+1	x+2	…
Description	Number of records	Number of bits 1st record		1st data byte of 1st sequence	…	Number of bits 2nd record		1st data byte of 2nd sequence	…
		MSB	LSB			MSB	LSB		
Bit				7 6 5 4 3 2 1 0	…			7 6 5 4 3 2 1 0	…
Order Send/Received				1 2 3 4 5 6 7 8	…			1 2 3 4 5 6 7 8	…

The first byte always contains the number of records following. Each record consists of a two byte header containing the number of bits followed by the necessary number of bytes to represent the bits transmitted/received.

The data section of the record is always byte-aligned even if e.g. only 5 bits are transmitted; the next record starts at the next byte of the buffer.

For each transmit record in `TxBuf3`, a corresponding receive record exists in `RxBuf3`. This is the case even if any of them is empty (e.g. the write page command has two transmit and only one receive part).

8.2 Printing the data

The code below illustrates how to print the actual bits transmitted and received on the screen:

```
void logMessage(
    const char* const prefix, const uint8_t* buf, const uint8_t idx) {
  uint16_t offset = 1;
  uint16_t bitcount;
  int i;
  uint8_t mask;

  /* "fast forward" to the bits of the given index */
  for (i = 0; i < idx; i++) {
    bitcount = ((buf[offset] << 8) | buf[offset + 1]);
    offset += 2 + ((bitcount / 8) + (bitcount % 8 == 0 ? 0 : 1));
  }

  /* print the message on screen */
  bitcount = ((buf[offset] << 8) | buf[offset + 1]);
  printf("%s %i bits\n", prefix, bitcount);

  /* print the bits on screen (if any) */
  if (bitcount > 0) {
    for (i = 0; i < bitcount; i++) {
      mask = (1 << (7 - (i % 8)));
      printf("%c", ((buf[2 + offset + (i / 8)] & mask) ? '1' : '0'));
    }
    printf("\n");
  }
}

void logAllMessages(const phTedKit_BaseData_t* const baseData) {
  int txCnt = 0;
  int rxCnt = 0;

  while ((txCnt < baseData->TxBuf3[0]) || (rxCnt < baseData->RxBuf3[0])) {
    if (txCnt < baseData->TxBuf3[0]) {
```

```
      logMessage("Transmit", baseData->TxBuf3, txCnt);
      txCnt++;
    }

    if (rxCnt < baseData->RxBuf3[0]) {
      logMessage("Receive", baseData->RxBuf3, rxCnt);
      rxCnt++;
    }
  };
}
```

The function `logAllMessages` can be called right after the execution of an appropriate Tx/Rx command. It will loop through both buffers and print the bits transmitted (first) and received (second).

Adding those two functions to the example shown in section 7, page 18 and calling `logAllMessages` right after the write page command:

```
702  baseData.Function = PHTEDKITBASEAPIFKT_DELAY;
703  /* T_prog = 4.92ms -> 4920µs */
704  baseData.TxTime1 = (uint32_t) (4920 / tickTime);
705  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
     logAllMessages(&baseData);
706  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
707    …
710  }
```

produces the following output:

```
…
Transmit 10 bits
0000010010
Receive 10 bits
0001110110
Transmit 32 bits
10010001010000011000111111101000
Receive 0 bits
…
```

This is exactly how the transmit/receive sequence for the write page command of the HITAG2-Extended transponder is defined. The data shown are the actual data transmitted/received and thus, are encrypted.

9. API Reference - Overview

This and the following sections give a detailed description of all the functions offered by the API and all the functionality which can be accessed through this API.

9.1 Functions

The API gives access to the functionality of the TED-Kit 2. This is achieved by using only a few functions. Depending on the programming language, the names and parameters differ a bit. Each application using the TED-Kit 2 and its API will go through three parts:

1. Initialize the API and the TED-Kit 2 software and hardware.
2. Execute one or more functions of the TED-Kit 2
3. Clean-up the API and TED-Kit 2 library and resources.

For each of the three steps, the API offers one function. There naming and parameters differ a bit as well as the location of the declaration depending on the used programming language.

9.1.1 Initialization

This function must be called before any other interaction with the API can take place. Each of the three functions returns a (language specific) handle to an instance of the API. Calling it multiple times will create different, independent instances of the API (e.g. required to interact with multiple TED-Kit 2 devices at once).

Table 7. API Initialization Function

Programming Language Interface	Method Signature
C	`void* phcsApiInt_Alloc()`
C++	`phcs_TedKit2::ApiInt()`
C#	`phcs_TedKit2::API()`

9.1.2 Execution

To execute actual TED-Kit 2 functions, the following method must be used:

Table 8. API Execution Function

Programming Language Interface	Method Signature
C	`PHTEDKITSTATUS phcsApiInt_Run( void* instance, uint16_t comp, void* param)`
C++	`phcs_TedKit2::PHTEDKITSTATUS phcs_TedKit2::ApiInt.Run(uint16_t comp, void* param)`
C#	`phcs_TedKit2::API.Run(ushort comp, Object param)`

For C the first parameter is always the handle of the desired API/TED-Kit 2. For C++ and C#, the handle is an instance of the `ApiInt` class and its method `Run` can be called directly on this instance.

The parameter `comp` defines the layer to which the desired function belongs (see Table 11, page 35; Table 14, page 49 and Table 18, page 85).

The last parameter is a pointer (or reference in C#) to a structure (or class in C#). Depending on the API layer, one of two structures types needs to be used (see Table 11, page 35; Table 14, page 49 and Table 18, page 85).

All parameters for a TED-Kit 2 function and all return values created by the TED-Kit 2 function are passed through this structure.

9.1.3 Clean-Up

To clean-up all the resources used by the software, the following method shall be called. Once this is done, the handle cannot be used anymore!

Table 9. API Destruction Function

Programming Language Interface	Method Signature
C	`void phcsApiInt_Destroy(void* instance)`
C++	`phcs_TedKit2::~ApiInt()`
C#	`void phcs_TedKit2::API.dispose()`

9.2 Common Attributes

There are several attributes of the parameter structure which are shared between some or all functions. These attributes are explained in detail in the following sections.

9.2.1 Function ID

The attribute `Function` is required for each function call. It contains the function's ID intended to call. It must be one of the IDs shown in Table 10, page 35; Table 13, page 49 and Table 17, page 85.

9.2.2 Status Code

Each function of the API returns a status code stored in the attribute `StatusCode`. In case the function executed successfully, the return code will always be `PHTEDKITSTATUS_OK`. In case of a problem, the status code will be one of the codes listed in Table 28, page 122.

9.2.3 Trace Buffer

All functions of the Ext- (`PHTEDKITCOMPID_EXTAPI`) and the Base- (`PHTEDKITCOMPID_BASEAPI`) API are capable of returning the debug trace data eventually created by the firmware. This feature is not available for functions of the I/O layer (`PHTEDKITCOMPID_IO`).

The debug trace data are nothing but a NULL terminated, human readable ASCII character string. It contains text which allows analyzing the firmware's program flow. This text is usually available only if the debug version of the firmware is running at the TED-Kit 2's µController.

The trace is stored in the attribute `TraceBuf` of the structure `phTedKit_BaseData_t`. To print it, any character string printing function can be used, e.g.

```
std::cout << dataStructure.TraceBuf << std::endl;
```

Or

```
printf("%s\n", dataStructure.TraceBuf);
```

To determine whether some debug trace data are available, the size of the text string stored in TraceBuf shall be determined. If it's greater zero, debug trace data are available:

```
if (strlen(dataStructure.TraceBuf) > 0) {
  // debug trace data are available
  …
}
```

It is guaranteed that the TraceBuf attribute of the structure in use is NULL terminated. In case the debug trace information is empty, NULL will be the first and only content of the TraceBuf.

9.2.4 Timings

All timing parameters set or returned are specified as so called *ticks*. Each tick represents a certain amount of time which depends on the clock cycle of the TED-Kit 2's µController. This simplifies the time handling inside the firmware dramatically but creates some overhead for the API user.

To use the correct timing information, one has to determine the µControllers clock frequency by calling GetDeviceStatus (see section 11.8, page 59) and evaluating the 4th element of the returned RxBuf1:

```
long frequency;
baseData.Function = phcs_TedKit2::PHTEDKITBASEAPIFKT_GETDEVICESTATUS;
/* call the API's run(..) method */
phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &baseData);
if (evalStatus(baseData.StatusCode)) {
  frequency = (  (baseData.RxBuf1[6] << 24) | (baseData.RxBuf1[7] << 16)
               | (baseData.RxBuf1[8] << 8)  | baseData.RxBuf1[9]);
}
```

Now, the software can create the proper API timing values out of the natural timings (in seconds) given by the user, e.g.:

```
double headerSymbolDuration_s = 0.000128; /* 128 µs */
```

The user specifies e.g. the header symbol duration as 128 µs (0.000128 seconds). The tick time is than the natural time multiplied with the frequency in Hz:

```
int headerSymbolDuration_tick = headerSymbolDuration_s * frequency;
```

The result depends on the value of frequency, for a µController running at e.g. 48 MHz, the tick time value is 6144 and for a µController running at 24 MHz it is 3072. This value than shall be used during the API call of e.g. SetReceptionParams (see section 11.19, page 76) or GetReceptionParams (see section 11.11, page 63) .

All natural timing values need to be converted to the system specific tick time and vice versa. The user shall never see these tick times. In the user interface only timings in seconds (or the proper sub units like µs or ms) shall be used.

9.3 Common Example Code

All example code in the following sections is written in plain C and will run if compiled together with the code section shown below.

Example 9-1. Preamble for Example Code

```
#include "intfs\IphcsApiInt\inc\phIcsApiInt.h"
#include "types\phTedKitCommands.h"
#include "types\phTedKitStatus.h"

#include <stdio.h>

void* getTEDKit2API() {
  phTedKit_IoData_t data;

  void* api = phcsApiInt_Alloc();

  data.Function = PHTEDKITIOFKT_GET_DEVICE_NUMBER;
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  printf("FTDI devices found: %i\n", data.DeviceNum);

  data.Function = PHTEDKITIOFKT_OPEN;
  data.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
  data.Port = 0;
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);

  return api;
}

void copy(const uint32_t value, uint8_t* const dest, const int offset) {
  dest[offset + 0] = (uint8_t) ((value >> 24) & 0xFF);
  dest[offset + 1] = (uint8_t) ((value >> 16) & 0xFF);
  dest[offset + 2] = (uint8_t) ((value >>  8) & 0xFF);
  dest[offset + 3] = (uint8_t) ((value >>  0) & 0xFF);
}
```

10. API Reference - I/O Functions

The functions of the I/O library component provide access to the major functions of the FTDI driver library. These functions are mainly used for maintenance of the TED-Kit 2 and do not deliver any functionality regarding NXP's base station or transponder components.

The functions of this group do not interact with the TED-Kit 2 µC firmware. They only talk to either the device driver or to the FTDI hardware of the TED-Kit 2 system.

The I/O layer offers the following functions:

Table 10. Function Codes - I/O Layer

Note: All values are prefixed with `PHTEDKITIOFKT_`

Value	Description	Page
`CLOSE`	Close a FTDI/TED-Kit 2 device.	36
`EE_UAREAD`	Reads the TED-Kit 2's FTDI EEPROM.	37
`EE_UASIZE`	Returns the TED-Kit 2's FTDI EEPROM size.	39
`EE_UAWRITE`	Writes the TED-Kit 2's FTDI EEPROM.	40
`GET_API_VERSION`	Returns the TED-Kit 2 API library version.	41
`GET_DEVICE_INFODETAIL`	Returns details about a FTD/TED-Kit 2 device.	42
`GET_DEVICE_NUMBER`	Returns the number of FTDI devices on the USB.	44
`GET_DRIVER_VERSION`	Returns the FTDI/TED-Kit 2 device driver version.	45
`GET_LIBRARY_VERSION`	Returns the FTDI library version.	46
`OPEN`	Opens a FTDI/TED-Kit 2 device.	47

All functions explained in this section use the following API Run-method parameters:

Table 11. Parameters of Method `Run` – I/O layer

Parameter	Value
Component ID	`PHTEDKITCOMPID_IO`
Structure Type	`phTedKit_IoData_t`

10.1 Close

This function closes the connection to a previously opened TED-Kit 2 device.

See section 10.10, page 47 for the corresponding `Open` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_CLOSE`	
	Output	
`StatusCode`	see Table 28, page 122	The status code information about success or failure.

Example 10-1: Close

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_CLOSE;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

10.2 EEUARead

This function reads the content of the TED-Kit 2's FTDI IC EEPROM user area. The maximum size of that memory is determined by calling function `EEUASize` (section 10.3, page 39). If the input attribute `EESize` exceeds this number, the behavior is undefined.

See section 10.4, page 40 for the corresponding `EEUAWrite` command.

For the meaning of each of the `EEData`'s values (the memory layout), see Table 12, page 37.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_EE_UAREAD`	
`EESize`	unsigned 32 Bit	The number of bytes being read, maximum can be retrieved with call of `EEUASize`.
	Output	
`StatusCode`	see Table 28, page 122	The status code information about success or failure.
`EEData`	0...`EESize` of unsigned 8 bit	The bytes being read.

Table 12. FTDI EEPROM layout

Address	Value Description
0	The hardware version of the TED-Kit 2 main board. Currently, the values 0, 1 and 2 (for the board revisions 0, 1 and 2) are in use.
1..13	The TED-Kit 2's custom name. 12 ASCII characters in the range from 20_{hex} to $7E_{hex}$ (all printable US-ASCII characters).
14..23	Currently not used.

Example 10-2: EEUARead

```
int main() {
  phTedKit_IoData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* determine user area size */
  const uint32_t EEPROM_SIZE = 22; /* the default UAEE size */
  /* populate data structure */
  data.Function = PHTEDKITIOFKT_EE_UAREAD;
  data.EESize = EEPROM_SIZE;
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    /* print the user area content */
    for (i = 0; i < data.EESize; i++) {
      printf("%02X\n", (int) data.EEData[i]);
    }
  }
}
```

10.3 EEUASize

This function returns the number of bytes of the TED-Kit 2's FTDI IC EEPROM user area.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITIOFKT_EE_UASIZE	
	Output	
StatusCode	see Table 28, page 122	The status code information about success or failure.
EEEsize	unsigned 32 bit	The size of the user area in bytes.

Example 10-3: EEUASize

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_EE_UASIZE;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    /* print the user area content */
    printf("User EE size=%i bytes\n", data.EESize);
  }
}
```

10.4 EEUAWrite

This function writes the content of the TED-Kit 2's FTDI IC EEPROM user area. The maximum size of that memory is determined by calling function `EEUASize` (section 10.3, page 39). If the input attribute `EESize` exceeds this number, the behavior is undefined.

See section 10.2, page 37 for the corresponding `EEUARead` command.

For the meaning of each of the `EEData`'s values (the memory layout), see Table 12, page 37.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_EE_UAWRITE`	
`EESize`	0...`PHTEDKITIO_BUFSIZE` of unsigned 8 bit.	The number of bytes being written.
`EEData`	Array of unsigned 8 bit with `EESize` of valid data.	The bytes being written (always starts at EEPROM address 0).
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.

Example 10-4: EEUAWrite

```
int main() {
  phTedKit_IoData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_EE_UAWRITE;
  /* write ten bytes */
  data.EESize = 10;
  for (i = 0; i < 10; i++) {
    data.EEData[i] = i;
  }

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

10.5 GetAPIVersion

This function returns the version of the API library currently in use.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITIOFKT_GET_API_VERSION	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
ID	32 bit unsigned	The version information.
	Bits 0..7	The micro part of the version.
	Bits 8..15	The minor part of the version
	Bits 16..23	The major part of the version.

Example 10-5: GetAPIVersion

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_GET_API_VERSION;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("TED-Kit 2 API Version: %i.%i.%i\n",
      ((data.ID >> 16) & 255),
      ((data.ID >>  8) & 255),
      ((data.ID >>  0) & 255));
  }
}
```

10.6 GetDeviceInfoDetail

This function returns configuration details of an FTDI device (which might be a TED-Kit 2).

This function can be called without opening the device in advance. The function `GetDeviceNumber` (see section 10.7, page 44) must be called in advance to allow the device driver to properly determine the device details.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_GET_DEVICE_INFODETAIL`	
`Port`	0..126	The port (index) of the TED-Kit 2 box of interest.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`Flags`	unsigned 32 bit	Bit 0 indicates whether this port is open (1) or closed (0). All other bits are reserved and have no purpose at this time.
`Type`	unsigned 32 bit	unknown
`ID`	unsigned 32 bit	The hardware's product and vendor ID. For the TED-Kit 2, always `PHTEDKITUSB_VIDPID`.
`LocID`	unsigned 32 bit	unknown
`SerialNumber`	`NULL` terminated character string, length is 1..16 (incl. `NULL`)	The TED-Kit 2's unique serial number.
`Description`	`NULL` terminated character string, length 1..64 (incl. `NULL`)	The description of the TED-Kit 2 device from the USB point of view, always *TED-Kit 2*.

Example 10-6: GetDeviceInfoDetail

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_GET_DEVICE_INFODETAIL;
  data.Port = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("FTDI device details:\n");
    printf("TED-Kit 2    : %s\n",
      (data.ID == PHTEDKITUSB_VIDPID ? "yes" : "no"));
    printf("Port         : %d\n", data.Port);
    printf("Flags        : %d\n", data.Flags);
    printf("                -> opened=%s\n", ((data.Flags & 1) ? "yes" : "no"));
    printf("Type         : %d\n", data.Type);
    printf("DeviceID     : 0x%08X\n", data.ID);
    printf("LocId        : %d\n", data.LocId);
    printf("SerialNumber : %s\n", data.SerialNumber);
    printf("Description  : %s\n", data.Description);
  }
}
```

10.7 GetDeviceNumber

This function returns the number of FTDI devices found on the USB.

A FTDI device is not necessarily a TED-Kit 2 device (FTDI chips can be found in many products). To ensure an FTDI device is actually a TED-Kit 2 device, check for the vendor- and product code returned by `GetDeviceInfoDetail` (see section 10.6, page 42).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_GET_DEVICE_NUMBER`	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`DeviceNum`	0..126	The number of FTDI devices found on the USB.

Example 10-7: GetDeviceNumber

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_GET_DEVICE_NUMBER;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("FTDI devices found: %i\n", data.DeviceNum);
  }
}
```

10.8 GetDriverVersion

This function returns the version of the FTDI device driver currently in use.

In order to get a valid result, at least one FTDI device must be currently opened (see `Open`, section 10.10, page 47).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_GET_DRIVER_VERSION`	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`ID`	32 bit unsigned	The version information as. `major.minor.micro`.
	Bits 0..7	The micro part of the version.
	Bits 8..15	The minor part of the version
	Bits 16..23	The major part of the version.

Example 10-8: GetDriverVersion

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* OPEN AT LEAST ONE DEVICE … */

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_GET_DRIVER_VERSION;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("FTDI Driver Version: %i.%i.%i\n",
      ((data.ID >> 16) & 255),
      ((data.ID >>  8) & 255),
      ((data.ID >>  0) & 255));
  }
}
```

10.9 GetLibraryVersion

This function returns the version of the (FTDI-) library currently in use.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITIOFKT_GET_LIBRARY_VERSION	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
ID	32 bit unsigned	The version information as. major.minor.micro.
	Bits 0..7	The micro part of the version.
	Bits 8..15	The minor part of the version
	Bits 16..23	The major part of the version.

Example 10-9: GetLibraryVersion

```
int main() {
  phTedKit_IoData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITIOFKT_GET_LIBRARY_VERSION;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("FTDI Library Version: %i.%i.%i\n",
      ((data.ID >> 16) & 255),
      ((data.ID >>  8) & 255),
      ((data.ID >>  0) & 255));
  }
}
```

10.10 Open

This function opens a FTDI/TED-Kit 2 device. Calling this function is required to work with a device or to update its firmware (see input parameter).

If the device is not longer used, it shall be closed using the appropriate API function `Close` (see section 10.1, page 36).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITIOFKT_OPEN`	
`OpenMode`	`PHTEDKITIOFKT_NORMAL_OPERATION`	The TED-Kit 2 will operate in normal mode.
	`PHTEDKITIOFKT_FIRMWARE_UPDATE`	The TED-Kit 2 will be ready to load a new firmware (update).
`Port`	0..n	The device/port number of the TED-Kit 2 desired to be opened. This number is retrieved from the result of calling `GetDeviceInfoDetail`
`FileInfo`	String	The path and name of the HEX file containing the firmware to be uploaded. Used only if OpenMode is set to `PHTEDKITIOFKT_FIRMWARE_UPDATE`
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.

Example 10-10: Open (Normal Operation)

```
int main() {
  void* api = getTEDKit2API();

  // populate the data structure
  phTedKit_IoData_t data;
  data.Function = PHTEDKITIOFKT_OPEN;
  data.OpenMode = PHTEDKITIOFKT_NORMAL_OPERATION;
  data.Port = 0;
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

Example 10-11: Open (Firmware Update)

```
int main() {
  void* api = getTEDKit2API();

  /* populate the data structure */
  phTedKit_IoData_t data;
  data.Function = PHTEDKITIOFKT_OPEN;
  data.OpenMode = PHTEDKITIOFKT_FIRMWARE_UPDATE;
  data.Port = 0;
  data.FileInfo = "C:\\foo\\tedkit2.hex";
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_IO, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11. API Reference - Base Functions

The base API functions represent a direct access to the functionality provided by the TED-Kit 2 system (hardware with firmware running on the µC).

The base layer offers the following functions:

Table 13. Function Codes - Base Layer

*Note: All values are prefixed with PHTEDKITBASEFKT_*

Value	Description	Page
DELAY	Delays processing of the next TED-Kit 2 command.	50
DESELECTXSLOT	Deselects the given XSlot/XBoard.	51
DISABLECONTRECEPTION	Disables continuous data reception.	52
EDITGPIOPIN	Read/Write the TED-Kit 2's µC GPIO pins.	53
ENABLECONTRECEPTION	Enables continuous data reception.	55
GETBUTTONSTATES	Returns the status of the TED-Kit 2 buttons.	56
GETCONTRECEIVEDDATA	Returns the continuously received data.	57
GETDEVICESTATUS	Returns status information from the TED-Kit 2.	59
GETFWVERSION	Returns the firmware version running on a TED-Kit 2.	61
GETLEDSTATES	Returns the status (on/off) of the TED-Kit 2 LEDs.	62
GETRECEPTIONPARAMS	Returns the reception parameters for an XBoard.	63
GETTRANSMISSIONPARAMS	Returns the transmission parameters for an XBoard.	65
GETWORDSIZE	Returns the word size of the given XBoard.	67
GETXBOARDCONFIG	Returns the configuration of an XBoard.	68
GETXSLOTINFO	Returns information about the 4 XSlots of a TED-Kit 2.	70
RESETMAINBOARD	Resets the TED-Kit 2 main board (incl. µC)	73
RESETXBOARD	Resets the given XBoard.	74
SETLEDSTATES	Sets the state (on/off) of the TED-Kit 2's LEDs.	75
SETRECEPTIONPARAMS	Sets the reception parameters of an XBoard.	76
SETTRANSMISSIONPARAMS	Sets the transmission parameters of an XBoard.	78
SETWORDSIZE	Sets the word size for the given XBoards	80
SETXBOARDCONFIG	Sets the configuration of an XBoard.	81
TRANSMITRECEIVE	Transmits/Receives data as configured.	83

All functions explained in this section use the following API Run-method parameters:

Table 14. Parameters of Method Run – base layer

Parameter	Value
Component ID	PHTEDKITCOMPID_BASEAPI
Structure Type	phTedKit_BaseData_t

11.1 Delay

This function requests the TED-Kit 2 to delay processing of the next command for the given number of system ticks.

The duration of one tick is the time of one clock cycle of the TED-Kit 2 µController. To retrieve the µController's clock frequency, call GetDeviceStatus (see section 11.8, page 59).

Data Structure Attributes Used

Structure Attribute	Value(s)		Description
	Input		
Function	PHTEDKITBASEAPIFKT_DELAY		
TxTime1	unsigned 32 bit		The number of ticks any further processing is delayed.
	Output		
StatusCode	see Table 28, page 122		The status code informing about success or failure.
TraceBuf	0..PHTEDKITTRACE_BUFSIZE characters, NULL terminated.	ASCII	The human readable debug trace created by the firmware for each call.

Example 11-1: Delay

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_DELAY;
  /* delay all activities for 10 milliseconds */
  data.TxTime1 = 480000; /* 10ms @ 48 MHz TED-Kit 2 system clock */

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.2 DeselectXSlot

This function deselects the currently selected XSlot of the TED-Kit 2. An XSlot becomes selected automatically if used (by the appropriate commands).

This command shall be executed before calling EditGPIOPin (see section 11.4, page 53).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_DESELECTXSLOT	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-2: DeselectXSlot

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_DESELECTXSLOT;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.3 DisableContReception

This function stops the continuous data reception and discards all received data. See section 11.5, page 55 for the corresponding `EnableContReception` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_DISABLECONTRECEPTION`	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-3: DisableContReception

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_DISABLECONTRECEPTION;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.4 EditGPIOPin

Sets the directions and values of the GPIO pins of the µController on the TED-Kit 2 main board and returns immediately the new pin directions and values.

In order to make this command work properly, the currently selected XBoard needs to be deselected (see command `DeselectXSlot`, section 11.2, page 51).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
		Input
`Function`	`PHTEDKITBASEAPIFKT_EDITGPIOPIN`	
`TxData1`	PinDirection, 0..65535	Bits 0…15 correspond to the GPIO pins 0…15. 0 sets the corresponding GPIO pin to input and 1 to output.
`TxData2`	PinDirectionMask, 0..65535	Bits 0 to 15 correspond to the GPIO pins 0 to 15. When a bit is “1”, the direction of the corresponding GPIO pin will be set according to the corresponding bit of PinDirection; when a bit is “0”, the direction of the corresponding GPIO pin remains unchanged.
`TxData3`	PinValue, 0..65535	Bits 0 to 15 correspond to the GPIO pins 0 to 15. Output GPIO pins will be set accordingly if the corresponding bits of PinValMask are “1”.
`TxData4`	PinValueMask, 0..65535	Bits 0 to 15 correspond to the GPIO pins 0 to 15. The bits that correspond to input GPIO pins are ignored. For the rest bits, when it is “1”, the corresponding output GPIO pin will be set according to the corresponding bit of PinValue; when a bit is “0”, the corresponding output GPIO pin remains unchanged.
		Output
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	PinDirection, 0..65535	Bits 0 to 15 correspond to the GPIO pins 0 to 15. If a bit is “0”, the corresponding GPIO pin is currently set as input; otherwise, the pin is set as output.
`RxData2`	PinValue, 0..65535	Bits 0 to 15 correspond to the values of GPIO pins 0 to 15.

Example 11-4: EditGPIOPin

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_EDITGPIOPIN;
  /* set pins to reading (input) */
  data.TxData1 = 0;
  /* pin direction valid only for pins 0 and 1 */
  data.TxData2 = 0x03;
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    /* do something with the data retreived. */
    printf("%X\n", (data.RxData2 & data.RxData1));
  }
}
```

11.5 EnableContReception

This function starts continuous data reception using the XBoard in the designated XSlot. Only one XBoard can receive continuous data at a time.

To actually get the data received, repetitive calls to function `GetContReceivedData` (see section 11.7, page 57) are necessary.

To stop continuous reception of data, use function `DisableContReception` (see section 11.3, page 52).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_ENABLECONTRECEPTION`	
`TxData1`	See Table 29, page 124.	Specifies the target XSlot to be selected.
`TxData2`	RxMaxLength, 0..65535	The maximum data frame size to be received, in words.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-5: EnableContReception

```
int main() {
  phTedKit_BaseData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_ENABLECONTRECEPTION;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = 500;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.6 GetButtonStates

This function returns the state of the 2 buttons at the TED-Kit 2 box. These buttons can be freely used to interact with the user.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_GETBUTTONSTATES	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxData1	Button states, only bits 0 and 1 relevant.	The button state (bit 0 for button 1, bit 1 for button 2), 1 indicates pressed, 0 indicates released.

Example 11-6: GetButtonStates

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETBUTTONSTATES;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("Button #1: %s\n",
      (data.RxData1 & 1 ? "pressed" : "released"));
    printf("Button #1: %s\n",
      (data.RxData1 & 2 ? "pressed" : "released"));
  }
}
```

11.7 GetContReceivedData

This function is used to retrieve the next available chunk of data during continuous data reception.

Continuous data reception has to be enabled first using function `EnableContReception` (see section 11.5, page 55).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETCONTRECEIVEDDATA`	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	RxLength, 16 Bit	The length of received response data frame in word.
`RxData2`	Overrun, 8 Bit, true/false information	Reception of frames is contiguous.
`RxBuf1`	RxData	The received data frame.
`RxBuf2`	Timestamp, 8 Byte	The timestamp of the received data frame

Example 11-7: GetContReceivedData

```
int main() {
  phTedKit_BaseData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* CONFIGURE AND ENABLE CONTINOUS RECEPTION… */

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETCONTRECEIVEDDATA;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    if (data.RxData1 != 0) {
      printf("data loss\n");
    } else {
      printf("received %i bytes:\n", data.RxData1);
      for (i = 0; i < data.RxData1; i++) {
        printf("%02X\n", data.RxBuf1[0]);
      }
    }
  }
}
```

11.8 GetDeviceStatus

This function returns status information of the TED-Kit 2 device.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_GETDEVICESTATUS	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	Byte array with 10 entries representing 4 numbers, each number is stored in big endian (highest byte first)	
	[0..1]	The number of bytes available in the µController's memory.
	[2..3]	The number of bytes of the largest continuous block in the µController's memory.
	[4..5]	The supply voltage in multiples of 0.03255V.
	[6..9]	The frequency of the µController clock in Hz.

Example 11-8: GetDeviceStatus

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETDEVICESTATUS;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("µC memory free              : %i bytes\n",
      ((data.RxBuf1[0] << 8) | data.RxBuf1[1]));
    printf("µc memory largest block free: %i bytes\n",
      ((data.RxBuf1[2] << 8) | data.RxBuf1[3]));
    printf("Supply Voltage              : %f V\n",
      ((data.RxBuf1[4] << 8) | data.RxBuf1[5]) * 0.03255f);
    printf("µC clock frequency          : %i Hz\n",
      (  (data.RxBuf1[6] << 24) | (data.RxBuf1[7] << 16)
       | (data.RxBuf1[8] << 8)  |  data.RxBuf1[9]));
  }
}
```

11.9 GetFWVersion

This function returns the code and version of the firmware currently running at the TED-Kit 2 board.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_GETFWVERSION	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxData1	unsigned 16 bit	Firmware code number. For TED-Kit 2 always 0.
RxData2	unsigned 16 bit	The major version number.
RxData3	unsigned 16 bit	The minor version number.

Example 11-9: GetFWVersion

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETFWVERSION;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("TED-Kit 2 Firmware Version: %i.%i\n",
      data.RxData2, data.RxData3);
  }
}
```

11.10 GetLEDStates

This function returns the status (on/off) of the 4 LEDs at the front of the TED-Kit 2 box.

See section 11.18 , page 75 for the corresponding `SetLEDStates` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETLEDSTATES`	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	Bits 0..3 of `RxData1`.	Each bit indicates the status of one LED (0 off, 1 on).

Example 11-10: GetLEDStates

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETLEDSTATES;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("LED #1:%s, #2:%s, #3:%s, #4:%s\n",
      (data.RxData1 & 1 ? "on" : "off"),
      (data.RxData1 & 2 ? "on" : "off"));
      (data.RxData1 & 4 ? "on" : "off"),
      (data.RxData1 & 8 ? "on" : "off"));
  }
}
```

11.11 GetReceptionParams

This function returns the reception parameters for the XBoard in the given XSlot.

See section 11.19, page 76 for the corresponding `SetReceptionParams` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETRECEPTIONPARAMS`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	Global Inversion, only bit 0 relevant.	Indicates whether the electrical voltage levels of the signals sent by the transponder are plain (0) or inverted (1).
`RxData2`	Synchronization mode, only bit 0 relevant: 0 – pattern based 1 – time based	Always 1, indicates time-based synchronization of the communication between transponder and base station.
`RxData3`	HeaderLength; 10 – standard, 14 - extended	The number of the half-bits the signal header contains of.
`RxData4`	DataWordSize	See Table 31, page 125.
`RxData5`	BodyDataCoding	See Table 31, page 125.
`RxTime1`	SyncTime; depends on the transponder type, refer to transponder documentation [3], [4] and [5] – look for $t_{WAIT,Tr}$.	Time in ticks between end of transmit from base station and start of receiving data from transponder.
`RxTime2`	HeaderSymbolDuration; depends on the transponder type, refer to transponder documentation [3], [4] and [5] – look for T_{unit}.	Duration in ticks for one symbol of the header.
`RxBuf1`	DataCodingParams; one value, encoded as big endian. The value depends on the transponder type, refer to transponder documentation [3], [4] and [5]	Timing information
	[0..3]	T_{unit}
`RxBuf2`	HeaderPattern; 10 or 14 Bits	The coded header bits, value depends on DataCodingType and HeaderLength.

Example 11-11: GetReceptionParams

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETRECEPTIONPARAMS;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("Global Inversion Flag : %s\n",
      (data.RxData1 ? "on" : "off"));
    printf("Synchronization Mode  : %s based\n",
      (data.RxData2 ? "time" : "pattern"));
    printf("Header Length         : %i bits\n", data.RxData3);
    printf("Word Size             : %i bits\n", data.RxData4);
    printf("Body Data Coding      : %s\n",
      (data.RxData5 ? "CDP" : "Manchester"));
    printf("Synchronizaton Time   : %i system ticks\n", data.RxTime1);
    printf("Header Symbol Duration: %i system ticks\n", data.RxTime2);
    printf("Data Coding Params    : 0x%0X, 0x%0X, 0x%0x, ...\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2]);
    printf("Header Pattern        : 0x%0X%0X\n",
      data.RxBuf2[0], data.RxBuf2[1]);
  }
}
```

11.12 GetTransmissionParams

Returns the transmission parameters set for the XBoard in the given XSlot.

See section 11.20, page 78 for the corresponding `SetTransmissionParams` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETTRANSMISSIONPARAMS`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	Invert, bit 0	Indicates whether the electrical voltage levels of the signals sent by the base station are plain (0) or inverted (1).
`RxData2`	PreambleLength	
`RxData3`	HeaderLength	
`RxData4`	TrailerLength	
`RxData5`	IdleLevel	The electrical level of energy (0 or 1) between base station and transponder if no communication happens (to ensure the transponder is still provided with energy).
`RxData6`	DataWordSize	See Table 31, page 125.
`RxData7`	BodyDataCoding	See Table 31, page 125.
`RxTime1`	PreambleSymbolDuration	
`RxTime2`	HeaderSymbolDuration	
`RxTime3`	TrailerSymbolDuration	
`RxBuf1`	DataCodingParams, array of 16 bytes, each value encoded as big endian. The values depend on the transponder type, refer to transponder documentation [3], [4] and [5]	Timing information
	[0..3]	T_{pulse} (in ticks)
	[4..7]	T_0 (in ticks)
	[8..11]	T_1 (in ticks)
	[12..15]	T_{STOP} (in ticks)
`RxBuf2`	HeaderPattern	not used
`RxBuf3`	TrailerPattern	not used

Example 11-12: GetTransmissionParams

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETTRANSMISSIONPARAMS;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("Global Inversion Flag : %s\n",
      (data.RxData1 ? "on" : "off"));
    printf("Preamble Length         : %i bits\n", data.RxData2);
    printf("Header Length           : %i bits\n", data.RxData3);
    printf("Trailer Length          : %i bits\n", data.RxData4);
    printf("Idle Level              : %s\n",
      (data.RxData5 ? "high" : "low"));
    printf("Word Size               : %i bits\n", data.RxData6);
    printf("Body Data Coding        : %i\n", data.RxData7);
    printf("Preamble Symbol Duration: %i system ticks\n", data.RxTime1);
    printf("Header Symbol Duration  : %i system ticks\n", data.RxTime2);
    printf("Trailer Symbol Duration : %i system ticks\n", data.RxTime3);
    printf("Data Coding Params      : 0x%0X, 0x%0X, 0x%0x, ...\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2]);
  }
}
```

11.13 GetWordSize

This function gets the word size for the different coding types. The word size is directly stored in the API.

One word represents the number of bits carried per symbol of the selected coding scheme. See Table 31, page 125 for an overview of value combinations.

See section 11.21, page 80 for the corresponding `SetWordSize` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETWORDSIZE`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest
`TxData2`	DataDirection, 0=Transmission, 1=Reception	Transmit or Receive direction
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	Word size	See Table 31, page 125.

Example 11-13: GetWordSize

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETWORDSIZE;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = 0; /* transmission */

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("Current word size of XSlot #%i (%s): %i bit(s)\n",
      data.TxData1, (data.TxData2 ? "Rx" : "Tx"), data.RxData1);
  }
}
```

11.14 GetXBoardConfig

This function returns the configuration of the XBoard in the given XSlot. The configuration data are device (XBoard) specific. The stream of bytes received needs to be interpreted according to the specification of the XBoard device (e.g. ABIC1 or LoPSTer).

See section 11.22, page 81 for the corresponding `SetXBoardConfig` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_GETXBOARDCONFIG`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
`TxData2`	Offset	The offset in the ConfigData array.
`TxData3`	Length	The length of the ConfigData array.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.
`RxBuf1`	alternative 1: ConfigData for ABIC1 Xboard see [UM10278_1]	The configuration data for ABIC1 being set.
	alternative 2: ConfigData for LoPSTer Xboard see [UM10278_1]	The configuration data for LoPSTer being set.

Example 11-14: GetXBoardConfig

```
int main() {
  phTedKit_BaseData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure to receive the configuraton
     for an ABIC1 XBoard. */
  data.Function = PHTEDKITBASEAPIFKT_GETXBOARDCONFIG;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  /* request all data starting at configuration 0 */
  data.TxData2 = 0;
  /* request all data (13 bytes) */
  data.TxData3 = 13;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("Configuration for ABIC1 in XSlot #%i:\n", data.TxData1);
    for (i = 0; i < data.TxData3; i++) {
      printf("0x%02X ", data.RxBuf1[i]);
    }
    printf("\n");
  }
}
```

11.15 GetXSlotInfo

This function returns information about the 4 XSlots of a TED-Kit 2 box.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_GETXSLOTINFO	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxData1	See Table 15, page 70.	Type of XBoard in XSlot 0.
RxData2	See Table 15, page 70.	Type of XBoard in XSlot 1.
RxData3	See Table 15, page 70.	Type of XBoard in XSlot 2.
RxData4	See Table 15, page 70.	Type of XBoard in XSlot 3.
RxData5	See Table 16, page 70.	Features of XBoard in XSlot 0.
RxData6	See Table 16, page 70.	Features of XBoard in XSlot 1.
RXdata7	See Table 16, page 70.	Features of XBoard in XSlot 2.
RxData8	See Table 16, page 70.	Features of XBoard in XSlot 3.

Table 15. XBoard Type Code Values

Value	Description
PHTEDKITXBOARD_SELFTESTING	future use
PHTEDKITXBOARD_EXPERIMENT	Indicates an experimental XBoard.
PHTEDKITXBOARD_UAA3220	future use
PHTEDKITXBOARD_CRYPTO	future use
PHTEDKITXBOARD_LOBSTER	Indicates a LoPSTer XBoard.
PHTEDKITXBOARD_ABIC1	Indicates an ABIC-1 XBoard.
PHTEDKITXBOARD_ABIC2	Indicates an ABIC-2 XBoard
PHTEDKITXBOARD_NONE	Indicates no XBoard present.

Table 16. XBoard Feature Code Values

Value	Description
PHTEDKITXBOARD_FEAT_LF	XBoard has LF antenna.
PHTEDKITXBOARD_FEAT_UHF	XBoard has UHF antenna.
PHTEDKITXBOARD_FEAT_SPI	XBoard communicates via SPI interface.
PHTEDKITXBOARD_FEAT_I2C	XBoard communicates via I2C interface.
PHTEDKITXBOARD_FEAT_GPIO	XBoard communicates via GPIO interface.
PHTEDKITXBOARD_FEAT_LIN	XBoard communicates via LIN.
PHTEDKITXBOARD_FEAT_INT	XBoard has Interrupt output.
PHTEDKITXBOARD_FEAT_AIN	XBoard has analog input.

Value	Description
PHTEDKITXBOARD_FEAT_AOUT	XBoard has analog output.
PHTEDKITXBOARD_FEAT_TX	XBoard is able to transmit data.
PHTEDKITXBOARD_FEAT_RX	XBoard is able to receive data.
PHTEDKITXBOARD_FEAT_CONFIGTXRX	XBoard can be configured while transmitting or receiving data.
PHTEDKITXBOARD_FEAT_DISABLED	XBoard is disabled due to a conflicting XBoard configuration.

Example 11-15: GetXSlotInfo

```
void printXSlotInfo(
    const int slot, const uint16_t type, const uint16_t features) {
  printf("XBoard in XSlot #%i:\n", slot);
  printf("  Type: ");
  switch (type) {
  case PHTEDKITXBOARD_SELFTESTING:
    printf("Selftesting\n"); break;
  case PHTEDKITXBOARD_EXPERIMENT:
    printf("Experiment\n"); break;
  case PHTEDKITXBOARD_UAA3220:
    printf("UAA 3220\n"); break;
  case PHTEDKITXBOARD_CRYPTO:
    printf("Crypto\n"); break;
  case PHTEDKITXBOARD_LOPSTER:
    printf("LoPSTer\n"); break;
  case PHTEDKITXBOARD_ABIC1:
    printf("ABIC-1\n"); break;
  case PHTEDKITXBOARD_ABIC2:
    printf("ABIC-2\n"); break;
  case PHTEDKITXBOARD_NONE:
    printf("None\n"); return;
  default:
    printf("Unknown (%02X)\n", type); return;
  }

  printf("  Features: LF(%s), UHF(%s), SPI(%s), I²C(%s), GPIO(%s), LIN(%s),\n",
    (features & PHTEDKITXBOARD_FEAT_LF ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_UHF ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_SPI ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_I2C ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_GPIO ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_LIN ? "yes" : "no"));
  printf("            Interrupt(%s), Analog In(%s), Analog Out(%s), Tx(%s),\n",
    (features & PHTEDKITXBOARD_FEAT_INT ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_AIN ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_AOUT ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_TX ? "yes" : "no"));
  printf("            Rx(%s), Configure Tx/Rx(%s), Disabled(%s)\n",
    (features & PHTEDKITXBOARD_FEAT_RX ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_CONFIGTXRX ? "yes" : "no"),
    (features & PHTEDKITXBOARD_FEAT_DISABLED ? "yes" : "no"));
}

int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_GETXSLOTINFO;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printXSlotInfo(PHTEDKITXBOARD_XSLOT_0, data.RxData1, data.RxData5);
    printXSlotInfo(PHTEDKITXBOARD_XSLOT_1, data.RxData2, data.RxData6);
    printXSlotInfo(PHTEDKITXBOARD_XSLOT_2, data.RxData3, data.RxData7);
    printXSlotInfo(PHTEDKITXBOARD_XSLOT_3, data.RxData4, data.RxData8);
  }
}
```

11.16 ResetMainBoard

This function resets the main board. After the reset, all four LEDs on the TED-Kit 2 main board are switched off and all GPIO pins are set to input pins.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_RESETMAINBOARD	
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated	The human readable debug trace created by the firmware for each call.

Example 11-16: ResetMainBoard

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_RESETMAINBOARD;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.17 ResetXBoard

This function resets the XBoard in the given XSlot. The state will be the same as after the device initialization, i.e. all parameters of the XBoard are back to default.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITBASEAPIFKT_RESETXBOARD	
TxData1	See Table 29, page 124.	The XSlot of the XBoard being reset.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace eventually created by the firmware for each call.

Example 11-17: ResetXBoard

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_RESETXBOARD;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.18 SetLEDStates

This function turns the 4 status LEDs at the TED-Kit 2 box on or off.

To retrieve the status of the LEDs, use command `GetLEDStates` (see section 11.10, page 62).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_SETLEDSTATES`	
`TxData1`	States, 0..15	The LED states, 0 – off, 1 – on. The bits 0 to 3 correspond with the LEDs 1 to 4. To leave a state unchanged, use the second argument (mask).
`TxData2`	Mask, 0..15	Masks the LED modifications, 0 – ignore modification, 1 – apply modification. The bits 0 to 3 correspond with the LEDs 1 to 4.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-18: SetLEDStates

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_SETLEDSTATES;
  /* set all to "on" */
  data.TxData1 = 0xFF;
  /* mask out all but LED 2 */
  data.TxData2 = 0x02;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.19 SetReceptionParams

This function sets the reception parameters for the XBoard in the given XSlot.

See section 11.11, page 63 for the corresponding `GetReceptionParams` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_SETRECEPTIONPARAMS`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
`TxData2`	SyncMode	Always 1, indicates time-based synchronization of the communication between transponder and base station.
`TxData3`	Invert	Indicates whether the electrical voltage levels of the signals sent by the transponder are plain (0) or inverted (1).
`TxData4`	HeaderLength; 10 – standard, 14 - extended	The number of the half-bits the signal header contains of.
`TxData5`	BodyDataCoding	See Table 31, page 125.
`TxTime1`	SyncTime; depends on the transponder type, refer to transponder documentation [3], [4] and [5] – look for $t_{WAIT,Tr}$.	Time in ticks between end of transmit from base station and start of receiving data from transponder.
`TxTime2`	HeaderSymbolDuration; depends on the transponder type, refer to transponder documentation [3], [4] and [5] – look for T_{unit}.	Duration in ticks for one symbol of the header.
`TxBuf1`	DataCodingParams; one value, encoded as big endian. The value depends on the transponder type, refer to transponder documentation [3], [4] and [5]	Timing information
	[0..3]	T_{unit}
`TxBuf2`	HeaderPattern; 10 or 14 Bits	The coded header bits, value depends on DataCodingType and HeaderLength.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-19: SetReceptionParams

```
int main() {
  phTedKit_BaseData_t data;
  /* tick time for the default µC of 48 MHz */
  float tickTime = 0.0208333;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure to receive data from a HITAG2 via XSlot #0 */
  data.Function = PHTEDKITBASEAPIFKT_SETRECEPTIONPARAMS;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = 1; /* time based synchronization */
  data.TxTime1 = (uint32_t) (1330 / tickTime); /* SyncDelay 1330µs */
  data.TxData3 = 0; /* global inversion off */
  data.TxData4 = 10; /* Header Length 10 bit */
  data.TxTime2 = (uint32_t) (128 / tickTime); /* Header Symbol Duration  128µs
*/
  data.TxData5 = PHTEDKITCODING_MANCHESTER;

  /* populate the data coding params (for Machester encoding): */
  /* Tunit = 128µs */
  copy ((uint32_t) (128 / tickTime), data.TxBuf1, 0);

  /* Header Pattern for Manchester/EQ -> 5540hex */
  data.TxBuf2[0] = 0x55;
  data.TxBuf2[1] = 0x40;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.20 SetTransmissionParams

This function sets the transmission parameters for the XBoard in the given XSlot.

See section 11.12, page 65 for the corresponding `GetTransmissionParams` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_SETTRANSMISSIONPARAMS`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
`TxData2`	Invert, bit 0	Indicates whether the electrical voltage levels of the signals sent by the base station are plain (0) or inverted (1).
`TxData3`	PreambleLength	Specifies the preamble pattern length in symbols.
`TxData4`	HeaderLength	Specifies the header pattern length in plain bit.
`TxData5`	TrailerLength	Specifies the trailer pattern length in plain bit.
`TxData6`	IdleLevel	The electrical level of energy (0 or 1) between base station and transponder if no communication happens (to ensure the transponder is still provided with energy).
`TxData7`	BodyDataCoding	See Table 31, page 125.
`TxTime1`	PreambleSymbolDuration	Duration in ticks for one symbol of the preamble.
`TxTime2`	HeaderSymbolDuration	Duration in ticks for one symbol of the header.
`TxTime3`	TrailerSymbolDuration	Duration in ticks for one symbol of the trailer
`TxBuf1`	DataCodingParams, array of 16 bytes, each value encoded as big endian. The values depend on the transponder type, refer to transponder documentation [3], [4] and [5]	Timing information
	[0..3]	T_{pulse} (in ticks)
	[4..7]	T_0 (in ticks)
	[8..11]	T_1 (in ticks)
	[12..15]	T_{STOP} (in ticks)
`TxBuf2`	HeaderPattern	Specifies the header pattern in plain bits.
`TxBuf3`	TrailerPattern	Specifies the trailer pattern in plain bits.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.

Example 11-20: GetTransmissionParams

```
int main() {
  phTedKit_BaseData_t data;
  /* tick time for the default µC of 48 MHz */
  float tickTime = 0.0208333;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure to transmit to a HITAG2 transponder via XSlot 0*/
  data.Function = PHTEDKITBASEAPIFKT_SETTRANSMISSIONPARAMS;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = 0; /* Invert -> no */
  data.TxData3 = 0; /* PreambleLength */
  data.TxTime1 = 0; /* PreambleSymbolDuration */
  data.TxData4 = 0; /* HeaderLength */
  data.TxTime2 = 0; /* HeaderSymbolDuration */
  data.TxData5 = 0; /* TrailerLength */
  data.TxTime3 = 0; /* TrailerSymbolDuration */
  data.TxData6 = 0; /* IdleLevel */
  data.TxData7 = PHTEDKITCODING_BPLM; /* Data Coding Type -> BPLM */

  /* populate the data coding params (for Machester encoding): */
  /* T_Pulse -> 48µs */
  copy((uint32_t) (48 / tickTime), data.TxBuf1, 0);
  /* T_Log0 -> 160µs */
  copy((uint32_t) (160 / tickTime), data.TxBuf1, 4);
  /* T_Log1 -> 224µs */
  copy((uint32_t) (224 / tickTime), data.TxBuf1, 8);
  /* T_Stop -> 288µs  */
  copy((uint32_t) (288 / tickTime), data.TxBuf1, 12);

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.21 SetWordSize

This function sets the word size for the different coding types. The word size is directly stored in the API.

One word represents the number of bits carried per symbol of the selected coding scheme. See Table 31, page 125 for an overview of value combinations.

See section 11.13, page 67 for the corresponding `GetWordSize` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_SETWORDSIZE`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest
`TxData2`	DataDirection, 0=Transmission, 1=Reception	Transmit or Receive direction
`TxData3`	Word size	See Table 31, page 125.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.

Example 11-21: SetWordSize

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Function = PHTEDKITBASEAPIFKT_SETWORDSIZE;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = 0; /* transmission */
  data.TxData3 = 1; /* 1 bit per word. */

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.22 SetXBoardConfig

This function sets the configuration of the XBoard in the given XSlot. The configuration data are device (XBoard) specific. The stream of bytes sent needs to be set-up according to the specification of the XBoard device (e.g. ABIC1 or LoPSTer).

To only set some configuration bytes, an offset and a length (smaller than the maximum number of configuration bytes) can be configured.

See section11.14, page 68 for the corresponding `GetXBoardConfig` command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_SETXBOARDCONFIG`	
`Device`	See Table 30, page 124.	The device type of the XBoard being configured.
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard of interest.
`TxData2`	Offset: ABIC1 0-12 LoPSTer 0-207 Note: The data being set have to be placed at the correct, original index in TxBuf1. Example: offset = 20, data being transferred have be placed at TxBug1[20] and beyond.	The offset within the ConfigData array.
`TxData3`	Length: ABIC1 1-13 LoPSTer 1-208	The number of bytes being transmitted out of ConfigData.
`TxBuf1`	alternative 1: ConfigData for ABIC1 Xboard see docu TED-Kit FW Host Interface Spec	The configuration data for ABIC1 being set.
	alternative 2: ConfigData for LoPSTer Xboard see docu TED-Kit FW Host Interface Spec	The configuration data for LoPSTer being set.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 11-22: SetXBoardConfig

```
int main() {
  phTedKit_BaseData_t data;
  /* tick time for the default µC of 48 MHz */
  float tickTime = 0.0208333;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure to configure an ABIC1 in XSlot #0 to work
     as base station for a HITAG2. */
  data.Function = PHTEDKITBASEAPIFKT_SETXBOARDCONFIG;
  data.Device = PHTEDKITEXTAPIDEVICE_ABIC1;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  /* offset = 0 -> set all configuration data at once */
  data.TxData2 = 0;
  /* length = 13 -> set all configuration data at once */
  data.TxData3 = 13;
  /* 1 byte "interface & mode" -> set to 0 -> "non-filtered" */
  data.TxBuf1[0] = 0;
  /* set the data rate to 100 kHz */
  copy((uint32_t) (10/tickTime), data.TxBuf1, 1);
  /* demodulator sampling phase */
  data.TxBuf1[5] = 0x2c;
  /* ignore, set to zero. */
  data.TxBuf1[6] = 0;
  /* ignore, set to zero. */
  data.TxBuf1[7] = 0;
  /* set ABIC1 configuration register 0. */
  data.TxBuf1[8] = 7;
  /* set ABIC1 configuration register 1. */
  data.TxBuf1[9] = 0;
  /* set ABIC1 configuration register 2. */
  data.TxBuf1[10] = 0;
  /* set ABIC1 configuration register 3. */
  data.TxBuf1[11] = 0;
  /* no test mode */
  data.TxBuf1[12] = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

11.23 TransmitReceive

Transmit and receives data to and from the transponder in range of the base station of the given XSlot.

`TransmitReceive` of the BaseApi is a general purpose T_x/R_x function. For a more transponder/immobilizer specific version, refer to the ExtApi TransmitReceive (see section 12.2.1, page 98).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_TRANSMITRECEIVE`	
`TxData1`	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
`TxData2`	See Table 29, page 124.	The XSlot of the XBoard used for reception.
`TxData3`	TxLength	The number of bits to send.
`TxData4`	RxLength	The number of bits expected to be received.
`TxBuf2`	TxData, array of unsigned 8 bit. data organization is like this:	The data bits send to the transponder. Bit 7 of byte 0 is send first followed by bit 6 and so on.
	Byte 0 1 …	
	Bit 7 6 5 4 3 2 1 0 7 6 …	
	Order 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. …	
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1...`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	RxDataLength in bits	The number of bits being received.
`RxBuf1`	RxData, array of unsigned 8 bit. data organization is like this:	The data bits received from the transponder. Bit 7 of byte 0 is received first followed by bit 6 and so on.
	Byte 0 1 …	
	Bit 7 6 5 4 3 2 1 0 7 6 …	
	Order 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. …	

Example 11-23: TransmitReceive (base-layer)

```
int main() {
  phTedKit_BaseData_t data;
  /* tick time for the default µC of 48 MHz */
  float tickTime = 0.0208333;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* SET THE TX PARAMETERS … */

  /* SET THE RX PARAMETERS … */

  /* SET THE TX WORD SIZE TO 1 … */

  /* SET THE RX WORD SIZE TO 1 … */

  /* populate data structure to execute first step
     of authentication of a HITAG2 via XSlot #0 */
  data.Function = PHTEDKITBASEAPIFKT_TRANSMITRECEIVE;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = PHTEDKITXBOARD_XSLOT_0;
  /* send 5 bits (5 bits for start authent) */
  data.TxData3 = 5;
  /* expect 32 bits returned from the transponder (IDE) */
  data.TxData4 = 32;
  /* reset duration -> 5ms */
  data.TxTime1 = (uint32_t) (5000 / tickTime);
  /* reset delay -> 10ms */
  data.TxTime2 = (uint32_t) (10000 / tickTime);
  /* start auth := 11000 := 1100 0000 (byte) := 0xC0*/
  data.TxBuf2[0] = 0xC0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_BASEAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    /* Print IDE */
    printf("%i bit(s) received:\n", data.RxData1);
    printf("IDE: %02X%02X%02X%02X\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2], data.RxBuf1[3]);
  }
}
```

For examples how to set the T_x and R_x parameters, refer to SetTransmissionParams (see section 11.20, page 78) or SetReceptionParams (see section 11.19, page 76) respectively.

For an example how to set the word size, refer to SetWordSize (see section 11.21, page 80).

12. API Reference - Extended Functions

The extended API functions provide functionality on top of the base API functions. They do not have a direct counterpart in the TED-Kit 2 system (in contrast to the base API functions).

The extended layer offers the following functions:

Table 17. Function Codes - Extended Layer

*Note: All values are prefixed with PHTEDKITEXTFKT_*

Value	Description	Page
	Cryptographic Engines	
AESCRYPTOINIT	Initializes the library's AES crypto engine.	87
AESCRYPTOOPERATION	Encrypts/decrypts data with the AES engine.	88
HITAG2CRYPTOINIT	Initializes the library's HITAG2 crypto engine.	89
HITAG2CRYPTOOPERATION	Encrypts/decrypts data with the HITAG2 engine.	91
HITAG3CRYPTOINIT	Initializes the library's HITAG2 crypto engine.	93
HITAG3CRYPTOOPERATION	Encrypts/decrypts data with the HITAG2 engine.	95
	Immobilizer Communication	
TRANSMITRECEIVE	Communicate with an immobilizer transponder.	98
	Passive Keyless Entry	
PKEAUTHENT	Performs PKE authentication.	104
PKEPOLLENABLE	Prepares polling of the IDE of PKE transponders in range.	106
PKEPOLLIDE	Polls the IDE from the PKE transponders in range.	107
PKEPOLLMUTE	Prevents a given PKE transponder from IDE polling.	109
PKEREADEEPROM	Reads the EEPROM of a PKE transponder.	111
PKEREADVBAT	Reads battery voltage of a PKE transponder.	113
PKERSSIALL	Returns the RSSI for all axes of a PKE transponder.	115
PKERSSISINGLE	Returns the RSSI for each axis of a PKE transponder.	117
PKEWRITEEEPROM	Writes the EEPROM of a PKE transponder.	119

All functions explained in this section use the following Run-method parameters:

Table 18. Parameters of Method Run – extended layer

Parameter	Value
Component ID	PHTEDKITCOMPID_EXTAPI
Structure Type	phTedKit_BaseData_t

12.1 Cryptography

The following functions can be used to access the cryptographic engines of the TED-Kit 2 library.

12.1.1 AESCyrptoInit

This function prepares the API's AES crypto-unit with the given ingredients for later use.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_AESCRYPTOINIT	
Device	PHTEDKITEXTAPIDEVICE_AESCRYPTO	See also Table 30, page 124.
TxBuf1	Secret Key 128 Bit, Bytes [0..15]	The AES secret key
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 12-1: AESCryptoInit

```
int main() {
  phTedKit_BaseData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Device = PHTEDKITEXTAPIDEVICE_AESCRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_AESCRYPTOINIT;
  for (i = 0; i < 15; i++) {
    data.TxBuf1[i] = i; /* secret key byte i */
  }

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %4X\n", data.StatusCode);
  }
}
```

12.1.2 AESCryptoOperation

Encrypts or decrypts given data using the 128 bit AES algorithm. The AES crypto-unit has to be initialized using function `AESCyrptoInit` (see section 12.1.1, page 87) first.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITEXTAPIFKT_AESCRYPTOOPERATION`	
`Device`	`PHTEDKITEXTAPIDEVICE_AESCRYPTO`	See also Table 30, page 124.
`TxBuf1`	Data to be encrypted or decrypted, bytes [0..15]	The data to be encrypted or decrypted.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1...`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.
`RxBuf1`	Encrypted or decrypted data, Bytes [0..15]	The decrypted or encrypted data.

Example 12-2: AESCryptoOperation

```
int main() {
  phTedKit_BaseData_t data;
  int i;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate data structure */
  data.Device = PHTEDKITEXTAPIDEVICE_AESCRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_AESCRYPTOOPERATION;
  for (i = 0; i < 15; i++) {
    data.TxBuf1[i] = i; /* payload byte i */
  }

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %4X\n", data.StatusCode);
  }
}
```

12.1.3 HITAG2CryptoInit

This function prepares the API's HITAG2 crypto-unit with the given ingredients for later use.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_HITAG2CRYPTOINIT	
Device	PHTEDKITEXTAPIDEVICE_HITAG2CRYPTO	See also Table 30, page 124.
TxBuf1	Secret Key 48 Bit, Bytes [0-5] IDE, Bytes [6-9] Random Number, Bytes [10-13]	The Secret Key, the Tags IDE and a Random Number.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 12-3: HITAG2CryptoInit

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate the data structure with the ingredients for the
     HITAG 2 crypto engine initialization. */
  data.Device = PHTEDKITEXTAPIDEVICE_HITAG2CRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_HITAG2CRYPTOINIT;
  /* set the (defualt HITAG2 immobilizer) secret key */
  data.TxBuf1[0] = 'M';
  data.TxBuf1[1] = 'I';
  data.TxBuf1[2] = 'K';
  data.TxBuf1[3] = 'R';
  data.TxBuf1[4] = 'O';
  data.TxBuf1[5] = 'N';
  /* set the IDE of a HITAG2 transponder */
  data.TxBuf1[6] = 0xf1;
  data.TxBuf1[7] = 0x2c;
  data.TxBuf1[8] = 0x7a;
  data.TxBuf1[9] = 0xb2;
  /* set some random number */
  data.TxBuf1[10] = 0;
  data.TxBuf1[11] = 0;
  data.TxBuf1[12] = 0;
  data.TxBuf1[13] = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

12.1.4 HITAG2CryptoOperation

Encrypts or decrypts given data using the HITAG2 crypto algorithm. The HITAG2 crypto-unit has to be initialized using function `HITAG2CryptoInit` (see section 12.1.3, page 89) first.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITEXTAPIFKT_HITAG2CRYPTOOPERATION`	
`Device`	`PHTEDKITEXTAPIDEVICE_HITAG2CRYPTO`	See also Table 30, page 124.
`TxData1`	Number of bits to process.	
`TxBuf1`	Data to be encrypted or decrypted, Bytes [0-x]	The data to be encrypted or decrypted.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1...`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxBuf1`	Encrypted or Decrypted data, Bytes [0-x]	The decrypted or encrypted data.

Example 12-4: HITAG2CryptoOperation

```
int main() {
  phTedKit_BaseData_t data;

    /* INITIALIZE THE HITAG2 CRYPTO UNIT … */

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* simulation of a start authent command (2nd, ciphered step) */
  data.Device = PHTEDKITEXTAPIDEVICE_HITAG2CRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_HITAG2CRYPTOOPERATION;
  data.TxData1 = 32;
  /* the base station password has to be send encrypted */
  data.TxBuf1[0] = 0xff;
  data.TxBuf1[1] = 0xff;
  data.TxBuf1[2] = 0xff;
  data.TxBuf1[3] = 0xff;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("send (the encrypted part, should be BDEA3E86): ");
    printf("%02X %02X %02X %02X\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2], data.RxBuf1[3]);
  }

  /* we received the encrypted default transponder password */
  /* encrypted: 94 b9 fa 0b , plain: xx AA 48 54 */
  data.TxData1 = 32;
  data.TxBuf1[0] = 0x94;
  data.TxBuf1[1] = 0xb9;
  data.TxBuf1[2] = 0xfa;
  data.TxBuf1[3] = 0x0b;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("received (should be xxAA4854):");
    printf("%02X %02X %02X %02X\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2], data.RxBuf1[3]);
  }
}
```

12.1.5 HITAG3CryptoInit

This function prepares the API's HITAG3 crypto-unit with the given ingredients for later use.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITEXTAPIFKT_HITAG3CRYPTOINIT`	
`Device`	`PHTEDKITEXTAPIDEVICE_HITAG3CRYPTO`	See also Table 30, page 124.
`TxBuf1`	IDE, Bytes [0-3] Challenge, Bytes [4-11] Secret Key 48 Bit, Bytes [12-23]	The secret key, the tag's IDE and a challenge
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1...`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 12-5: HITAG3CryptoInit

```
#include <string.h>

int main() {
  phTedKit_BaseData_t data;
  /* HITAG3 default secret key */
  uint8_t sk[] = {
    0x11, 0x11, 0x22, 0x22, 0x33, 0x33,
    0x44, 0x44, 0x55, 0x55, 0x66, 0x66 };

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* populate the data structure with the ingredients for the
     HITAG 2 crypto engine initialization. */
  data.Device = PHTEDKITEXTAPIDEVICE_HITAG3CRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_HITAG3CRYPTOINIT;
  /* set the IDE of the transponder 8B4C8010*/
  data.TxBuf1[0] = 0x8b;
  data.TxBuf1[1] = 0x4c;
  data.TxBuf1[2] = 0x80;
  data.TxBuf1[3] = 0x10;
  /* set some random number */
  memset(data.TxBuf1 + 4, 0, 8);
  /* set the default secret key */
  memcpy(data.TxBuf1 + 12, sk, 12);

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  }
}
```

12.1.6 HITAG3CryptoOperation

Encrypts or decrypts given data using the HITAG3 crypto algorithm. The HITAG3 crypto-unit has to be initialized using function `HITAG3CryptoInit` (see section 12.1.5, page 93) first.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITEXTAPIFKT_HITAG3CRYPTOOPERATION`	
`Device`	`PHTEDKITEXTAPIDEVICE_HITAG3CRYPTO`	See also Table 30, page 124.
`TxData1`	Number of bits to process.	
`TxBuf1`	Data to be encrypted or decrypted, Bytes [0-x]	The data to be encrypted or decrypted.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1...`PHTEDKITTRACE_BUFSIZE` ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
`RxBuf1`	Encrypted or Decrypted data, Bytes [0-x]	The decrypted or encrypted data.

Example 12-6: HITAG3CryptoOperation

```
int main() {
  phTedKit_BaseData_t data;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* INITIALIZE THE HITAG3 CRYPTO UNIT … */

  /* simulation of a ciphered read page #0 command */
  data.Device = PHTEDKITEXTAPIDEVICE_HITAG3CRYPTO;
  data.Function = PHTEDKITEXTAPIFKT_HITAG3CRYPTOOPERATION;
  data.TxData1 = 10;

  /* send the read page 0 := 11000 00111*/
  data.TxBuf1[0] = 0xc1;
  data.TxBuf1[1] = 0xc0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("send (the encrypted part, should be 0440): ");
    printf("%02X %02X\n", data.RxBuf1[0], data.RxBuf1[1]);
  }

  /* we received the encrypted page content 32 bit */
  /* we read page 0 -> the IDE; encrypted: 9FF0 8717, plain 8b4c 8010 */
  data.TxData1 = 32;
  data.TxBuf1[0] = 0x9f;
  data.TxBuf1[1] = 0xf0;
  data.TxBuf1[2] = 0x87;
  data.TxBuf1[3] = 0x17;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    printf("received (should be 8b4c 8010):");
    printf("%02X %02X %02X %02X\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2], data.RxBuf1[3]);
  }
}
```

12.2 Immobilizer

The following functions can be used to interact with NXP proprietary car immobilizer transponders.

12.2.1 TransmitReceive

Transmit and receives data to and from the immobilizer transponder in range of the base station of the given XSlot.

Using `TransmitReceive` via the ExtApi interface ensures a correct data setup according to transponder-type, transponder-state and transponder-command. In addition, data encryption and decryption is also handled by the API (if necessary).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
`Function`	`PHTEDKITBASEAPIFKT_TRANSMITRECEIVE`	
`Device`	See Table 30, page 124.	The immobilizer transponder used as communication counterpart.
`State`	See Table 19, page 99.	The state in which the transponder is expected to (for evaluation purposes only).
`Command`	See Table 20, page 99.	The command to be executed by the transponder.
`TxData1`	See Table 29, page 124.	XSlot/ XBoard for transmission.
`TxData2`	See Table 29, page 124.	XSlot/ XBoard for reception.
`TxTime1`	ResetDuration, set to 0 to prevent field reset.	The duration of the field turned off in TED-Kit 2 ticks.
`TxTime2`	ResetDelay, set to 0 to prevent field reset.	The time to be waited after a reset until a new transponder communication can be started in ticks.
`TxBuf1`	TxData, array of unsigned 8 bit. data organization is like this: Byte 0 1 … Bit 7 6 5 4 3 2 1 0 7 6 … Order 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. …	The data bits send to the transponder. Bit 7 of byte 0 is send first followed by bit 6 and so on. See [1] for details.
	Output	
`StatusCode`	see Table 28, page 122	The status code informing about success or failure.
`TraceBuf`	1..`PHTEDKITTRACE_BUFSIZE` ASCII characters, `NULL` terminated.	The human readable debug trace created by the firmware for each call.
`RxData1`	RxDataLength in bits	The number of bits being received.
`RxBuf1`	RxData, array of unsigned 8 bit. data organization is like this: Byte 0 1 … Bit 7 6 5 4 3 2 1 0 7 6 … Order 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. …	The data bits received from the transponder. Bit 7 of byte 0 is received first followed by bit 6 and so on. See [1] for details.

Table 19. Immobilizer State(-machine) Codes

Value	Description
HITAG2, HITAG2+	
PHTEDKITHITAG2STATE_WAIT	The transponder's immobilizer WAIT state.
PHTEDKITHITAG2STATE_AUTHORIZED	The transponder's immobilizer AUTHORIZED state
PHTEDKITHITAG2STATE_HALT	The transponder's immobilizer HALT state.
HITAG2+EE (in addition to HITAG2)	
PHTEDKITHITAG2STATE_XMA	The transponder's immobilizer XMA state.
HITAG2-Extended (in addition to HITAG2+EE)	
PHTEDKITHITAG2STATE_XMACFG	The transponder's immobilizer XMA config state.
PHTEDKITHITAG2STATE_TEST	The transponder's immobilizer TEST state.
~~PHTEDKITHITAG2STATE_HALT~~	Not supported.
HITAG3, HITAG-AES (in addition to HITAG2-Extended)	
PHTEDKITHITAG2STATE_USER	
HITAG-Pro	
PHTEDKITHITAGPROSTATE_WAIT	The transponder's immobilizer WAIT state.
PHTEDKITHITAGPROSTATE_AUTHENT	The transponder's immobilizer AUTHENT state.
PHTEDKITHITAGPROSTATE_CIPHER	The transponder's immobilizer CIPHER state.
PHTEDKITHITAGPROSTATE_PLAIN	The transponder's immobilizer PLAIN state.
PHTEDKITHITAGPROSTATE_CFG	The transponder's immobilizer CFG (config) state.
PHTEDKITHITAGPROSTATE_TEST	The transponder's immobilizer TEST state.
PHTEDKITHITAGPROSTATE_USER	The transponder's immobilizer USER state.

Table 20. Immobilizer Command Codes

Value	Description
HITAG2	
PHTEDKITHITAG2CMD_STARTAUTH	
PHTEDKITHITAG2CMD_GETIDE64	
PHTEDKITHITAG2CMD_READPAGE	
PHTEDKITHITAG2CMD_READPAGEINV	
PHTEDKITHITAG2CMD_WRITEPAGE	
PHTEDKITHITAG2CMD_HALT	
HITAG2+ (in addition to HITAG2)	
PHTEDKITHITAG2CMD_BATTTEST	
PHTEDKITHITAG2CMD_BUTTONTEST	
PHTEDKITHITAG2CMD_SETXON	
PHTEDKITHITAG2CMD_SETDOUT	
PHTEDKITHITAG2CMD_SETLED	
PHTEDKITHITAG2CMD_RESETALL	
~~PHTEDKITHITAG2CMD_GETIDE64~~	Not supported
HITAG2+EE (in addition to HITAG2+)	
PHTEDKITHITAG2CMD_XMAPLUS	
PHTEDKITHITAG2CMD_INCBLKPTR	

Value	Description
PHTEDKITHITAG2CMD_DECBLKPTR	
PHTEDKITHITAG2CMD_INITBLKPTR	
HITAG2-Extended (in addition to HITAG2)	
PHTEDKITHITAG2CMD_SOFTRESET	
PHTEDKITHITAG2CMD_REFRESH	
PHTEDKITHITAG2CMD_XMAWAIT	
PHTEDKITHITAG2CMD_XMACFG	
PHTEDKITHITAG2CMD_TEST	
PHTEDKITHITAG2CMD_XMA	
PHTEDKITHITAG2CMD_SELBLOCK	
PHTEDKITHITAG2CMD_WRITECFGS	
PHTEDKITHITAG2CMD_WRITECFGM	
PHTEDKITHITAG2CMD_READCFG	
PHTEDKITHITAG2CMD_THASHINIT	
PHTEDKITHITAG2CMD_THASHGET	
PHTEDKITHITAG2CMD_TWRITEPAGE	
PHTEDKITHITAG2CMD_TREADPAGE	
PHTEDKITHITAG2CMD_TGETCFG	
HITAG3 (in addition to HITAG2-Extended)	
PHTEDKITHITAG3CMD_USER	
PHTEDKITHITAG3CMD_XMAWAIT	
PHTEDKITHITAG3CMD_XMACFG	
PHTEDKITHITAG3CMD_TEST	
PHTEDKITHITAG3CMD_STARTAUTH	
PHTEDKITHITAG3CMD_GETIDE64	
~~PHTEDKITHITAG2CMD_XMAWAIT~~	Not supported
~~PHTEDKITHITAG2CMD_XMACFG~~	Not supported
~~PHTEDKITHITAG2CMD_TEST~~	Not supported
~~PHTEDKITHITAG2CMD_STARTAUTH~~	Not supported
~~PHTEDKITHITAG2CMD_GETIDE64~~	Not supported
HITAG-AES (in addition to HITAG2-Extended)	
PHTEDKITHITAGAESCMD_STARTAUTH250	
PHTEDKITHITAGAESCMD_STARTAUTH500	
PHTEDKITHITAGAESCMD_TAESINIT	
PHTEDKITHITAGAESCMD_TAESGET	
PHTEDKITHITAGAESCMD_GETIDE64	
~~PHTEDKITHITAG2CMD_STARTAUTH~~	Not supported
~~PHTEDKITHITAG2CMD_GETIDE64~~	Not supported
HITAG-Pro	
PHTEDKITHITAGPROCMD_SOFTRESET	
PHTEDKITHITAGPROCMD_REFRESH	

Value	Description
PHTEDKITHITAGPROCMD_GETIDE	
PHTEDKITHITAGPROCMD_AUTHENT	
PHTEDKITHITAGPROCMD_PLAIN	
PHTEDKITHITAGPROCMD_CFG	
PHTEDKITHITAGPROCMD_SEQINC	
PHTEDKITHITAGPROCMD_SELBLOCK	
PHTEDKITHITAGPROCMD_READPAGE	
PHTEDKITHITAGPROCMD_WRITEPAGE	
PHTEDKITHITAGPROCMD_READBYTE	
PHTEDKITHITAGPROCMD_WRITEBYTE	
PHTEDKITHITAGPROCMD_TAESINIT	
PHTEDKITHITAGPROCMD_TAESGET	
PHTEDKITHITAGPROCMD_TREADBYTE	
PHTEDKITHITAGPROCMD_TWRITEBYTE	
PHTEDKITHITAGPROCMD_THASHINIT	
PHTEDKITHITAGPROCMD_THASHGET	
PHTEDKITHITAGPROCMD_TGETCFG	
HITAG-Pro 2 (in addition to HITAG-Pro)	
PHTEDKITHITAGPRO2CMD_WRITEDIST	
PHTEDKITHITAGPRO2CMD_READDISTN	
PHTEDKITHITAGPRO2CMD_READMPAGE	
PHTEDKITHITAGPRO2CMD_SEQINC32	
PHTEDKITHITAGPRO2CMD_GETIDE64	

Example 12-7: TransmitReceive (extension-layer)

```
int main() {
  phTedKit_BaseData_t data;
  /* tick time for the default µC of 48 MHz */
  float tickTime = 0.0208333;

  /* get the API of the TED-Kit 2 being used. */
  void* api = getTEDKit2API();

  /* SET THE TX PARAMETERS … */

  /* SET THE RX PARAMETERS … */

  /* SET THE TX WORD SIZE TO 1 … */

  /* SET THE RX WORD SIZE TO 1 … */

  /* populate data structure to execute first step
     of authentication of a HITAG2 via XSlot #0 */
  data.Function = PHTEDKITBASEAPIFKT_TRANSMITRECEIVE;
  data.Device = PHTEDKITEXTAPIDEVICE_HITAG2;
  data.State = PHTEDKITHITAG2STATE_WAIT;
  data.Command = PHTEDKITHITAG2CMD_STARTAUTH;
  data.TxData1 = PHTEDKITXBOARD_XSLOT_0;
  data.TxData2 = PHTEDKITXBOARD_XSLOT_0;
  /* 2 repetitions of the command */
  data.TxData5 = 2;
  /* reset duration -> 5ms */
  data.TxTime1 = (uint32_t) (5000 / tickTime);
  /* reset delay -> 10ms */
  data.TxTime2 = (uint32_t) (15000 / tickTime);
  /* first step of the authentication sequence */
  data.TxBuf1[0] = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &data);
  /* evaluate status code returned */
  if (data.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure %04X\n", data.StatusCode);
  } else {
    /* Print IDE */
    printf("IDE: %02X%02X%02X%02X\n",
      data.RxBuf1[0], data.RxBuf1[1], data.RxBuf1[2], data.RxBuf1[3]);
  }
}
```

For examples how to set the T_x and R_x parameters, refer to SetTransmissionParams (see section 11.20, page 78) or SetReceptionParams (see section 11.19, page 76) respectively.

For an example how to set the word size, refer to `SetWordSize` (see section 11.21, page 80).

12.3 Passive Keyless Entry

The following functions can be used to interact with a NXP proprietary passive keyless entry (PKE) system.

In order to execute the example code, several setups and helper functions are necessary. A stub file containing all this is provided here:

[TED-Kit 2 installation]\Development\API\doc\examples\pke-example-stub.c

12.3.1 PkeAuthent

The function performs a PKE authentication using the HITAG2 crypto algorithm.

The crypto engine is fed with the tag's IDE, the immobilizer secret key and a random number. The value Signature1 is made from the first 16 output bits from the crypto engine, and verified by the tag. After successful verification, the tag responds with another 48 bit value from the crypto engine, called Signature2.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEAUTHENT	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxTime1	ResetDuration; depends on the transponder type, refer to transponder documentation [3], [4] and [5]	The duration of the reset sequence of the transponder in ticks.
TxTime2	ResetDelay; depends on the transponder type, refer to transponder documentation [3], [4] and [5]	The time to be waited after a reset until a new transponder communication can be started in ticks.
TxBuf1	IDE, Bytes [0-3] Random Number, Bytes [4-7] encrypted Signature1, Bytes [8-9]	The Tags IDE, a Random Number and an encrypted local Signature1.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	encrypted Signature2, Bytes [0-5]	The encrypted remote Signature2

Example 12-8: PkeAuthent

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEAUTHENT;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* set random number */
  longToBytes(0, baseData.TxBuf1, 0);
  /* signature 1 */
  baseData.TxBuf1[8] = 0;
  baseData.TxBuf1[9] = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("transponder signature := %02X %02X %02X %02X %02X %02X\n",
      baseData.RxBuf1[0], baseData.RxBuf1[1], baseData.RxBuf1[2],
      baseData.RxBuf1[3], baseData.RxBuf1[4], baseData.RxBuf1[5]);

  return shutdown();
}
```

12.3.2 PkePollEnable

This function enables the tag for reception of the following PkePollIde (see section 12.3.3, page 107) command.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEPOLLENABLE	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 12-9: PkePollEnable

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEPOLLENABLE;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  return shutdown();
}
```

12.3.3 PkePollIde

This function scans available tags in range and receives their IDE (one at a time).

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEPOLLIDE	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	Random Number, Bytes [0-1]	Random number to generate a timeslot for the tag response.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	IDE, Bytes [0-3]	The PKE tag's IDE

Example 12-10: PkePollIde

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEPOLLIDE;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* random number == 0 */
  baseData.TxBuf1[0] = 0;
  baseData.TxBuf1[1] = 0;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  } else {
    ide = bytesToLong(baseData.RxBuf1, 0);
    printf("IDE 0x%08X\n", ide);
  }

  return shutdown();
}
```

12.3.4 PkePollMute

This function "mutes" the one tag that is addressed by its own IDE, with respect to tag scanning. This tag will not respond to PkePollIde (see section 12.3.3, page 107) anymore, until another PkePollEnable (see section 12.3.2, page 106) command is issued.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEPOLLMUTE	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3]	The IDE of the Tag which has to be muted.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.

Example 12-11: PkePollMute

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEPOLLMUTE;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  longToBytes(ide, baseData.TxBuf1, 0);

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  return shutdown();
}
```

12.3.5 PkeReadEeprom

Reads an EEPROM page of 32 bits.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEREADEEPROM	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3] Page, Byte [4]	The Tags IDE and the selected page.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	Page value, Bytes [0-3]	The 32 Bit page value.

Example 12-12: PkeReadEeprom

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* EXECUTE PKE AUTHENT … */

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEREADEEPROM;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* set page to read := 5 */
  baseData.TxBuf1[4] = 5;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("page #5 := %08X\n", bytesToLong(baseData.RxBuf1));

  return shutdown();
}
```

12.3.6 PkeReadVbat

Measures and returns the tag battery voltage index. The index is used together with a transponder specific look-up table to get the actual voltage value.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEREADVBAT	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3] LOAD SELECTION, Byte [4] (1 – UHF, 2 – LED1, 4 – LED2)	The Tags IDE and the Load selection.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1...PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	Battery voltage, Byte [0]	The coded tag battery voltage.

Example 12-13: PkeReadVbat

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;
  /* The VBat lookup table applicable for a PCF7953 */
  const double vbat[] = {
    1.83, 1.92, 2.02, 2.12, 2.21, 2.31, 2.41, 2.50,
    2.60, 2.70, 2.80, 2.89, 2.99, 3.09, 3.18, 3.28 };

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEREADVBAT;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* set load selection: LED1 + LED2 */
  baseData.TxBuf1[4] = 2 | 4;

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("VBat index := %i, value := %fV\n",
      (baseData.RxBuf1[0] & 0x0F), vbat[baseData.RxBuf1[0] & 0x0F]);

  return shutdown();
}
```

12.3.7 PkeRssiAll

Performs RSSI measurements of all three axes, and calculates the squared vector length (squared geometric mean) $V^2 = X^2 + Y^2 + Z^2$.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKERSSIALL	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3] ADC Resolution, Byte [4] 0…5 ➔ 5 bit...10 bit	The tag's IDE and the ADC resolution.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	Sum, Bytes [0-2]	The floating point value for the sum of the 3 axis.

Example 12-14: PkeRssiAll

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKERSSIALL;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* ADC resolution 10 bit */
  baseData.TxBuf1[4] = 5;
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("RSSI := %f\n", bytesToDouble(baseData.RxBuf1));

  return shutdown();
}
```

12.3.8 PkeRssiSingle

Performs RSSI measurements of all three axes, and returns the three vector components X, Y, Z.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKERSSISINGLE	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3] ADC Resolution, Byte [4] 0…5 ➔ 5 bit...10 bit	The tag's IDE and the ADC resolution.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	X-Axis, Bytes [0-2] Y-Axis, Bytes [3-5] Z-Axis, Bytes [6-8]	The floating point values for the 3 axis.

Example 12-15: PkeRssiSingle

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKERSSISINGLE;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* ADC resolution 10 bit */
  baseData.TxBuf1[4] = 5;
  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("RSSI X:%f\n", bytesToDouble(baseData.RxBuf1 + 0));
  printf("RSSI Y:%f\n", bytesToDouble(baseData.RxBuf1 + 3));
  printf("RSSI Z:%f\n", bytesToDouble(baseData.RxBuf1 + 6));

  return shutdown();
}
```

12.3.9 PkeWriteEeprom

Writes an EEPROM page of 32 bits.

Data Structure Attributes Used

Structure Attribute	Value(s)	Description
	Input	
Function	PHTEDKITEXTAPIFKT_PKEWRITEEEPROM	
Device	PHTEDKITEXTAPIDEVICE_PKE	See also Table 30, page 124.
TxData1	See Table 29, page 124.	The XSlot of the XBoard used for transmission.
TxData2	See Table 29, page 124.	The XSlot of the XBoard used for reception.
TxBuf1	IDE, Bytes [0-3] Page, Byte [4] Page value [5-8]	The tag's IDE, the selected page and the Page value.
	Output	
StatusCode	see Table 28, page 122	The status code informing about success or failure.
TraceBuf	1..PHTEDKITTRACE_BUFSIZE ASCII characters, NULL terminated.	The human readable debug trace created by the firmware for each call.
RxBuf1	Page value, Bytes [0-3]	The 32 Bit page value confirmation.

Example 12-16: PkeWriteEeprom

```
int main(void) {
  /* An instance of the base data structure used to exchange data with
     the API Library. */
  phTedKit_BaseData_t baseData;
  /* The transponder's IDE */
  uint32_t ide;

  /* initialize the TED-Kit 2 and its components for PKE. */
  if (setup() == EXIT_FAILURE) {
    return EXIT_FAILURE;
  }

  /* EXECUTE PKE POLL ENABLE … */

  /* EXECUTE PKE POLL IDE … */
  ide = …;

  /* EXECUTE PKE AUTHENT … */

  /* populate data structure */
  baseData.Device = PHTEDKITEXTAPIDEVICE_PKE;
  baseData.Function = PHTEDKITEXTAPIFKT_PKEWRITEEEPROM;
  baseData.TxData1 = xSlotABICPort;
  baseData.TxData2 = xSlotLoPSTerPort;
  /* set IDE */
  longToBytes(ide, baseData.TxBuf1, 0);
  /* set page to write := 5 */
  baseData.TxBuf1[4] = 5;
  /* set value to write */
  longToBytes(0x12345678, baseData.TxBuf1, 5);

  /* call the API's run(..) method */
  phcsApiInt_Run(api, PHTEDKITCOMPID_EXTAPI, &baseData);
  /* evaluate status code returned */
  if (baseData.StatusCode != PHTEDKITSTATUS_OK) {
    /* failure, handle error */
    printf("failure 0x%04X\n", baseData.StatusCode);
    return EXIT_FAILURE;
  }

  printf("page #5 written, transponder answered:= %08X\n",
      bytesToLong(baseData.RxBuf1));

  return shutdown();
}
```

13. API Reference – Common Constants

13.1 Places of Definition

This section lists the locations of all the elements of the TED-Kit 2 API required and used throughout this manual. All locations given refer to the installation location of the TED-Kit 2 software.

13.1.1 Function Declarations

The three functions to initialize, execute and shut down a TED-Kit 2 API are defined in the following files:

Table 21. API Function Declarations

Programming Language Interface	Declaration in File
C	intfs\IphcsApiInt\inc\phIcsApiInt.h
C++	comps\phcsApiInt\inc\phcsApiInt.hpp
C#	TED-Kit 2 API.cs

13.1.2 Function Codes

The function codes for the three layers of the TED-Kit 2 API can be found here:

Table 22. API Function Codes Definitions

Programming Language Interface	Declaration in File	Item
C	types\phTedKitCommands.h	`#define …`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::Functions`

13.1.3 Data Structures

The data structures needed to exchange data with the TED-Kit 2 API are defined here:

Table 23. API Data Structure Declaration

Programming Language Interface	Declaration in File	Item
C	types\phTedKitTypeDefs.h	`phTedKit_IoData_t` `phTedKit_BaseData_t`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::IOData` `phcs_TedKit2::BaseData`

13.1.4 Status Codes

The status codes returned by the TED-Kit 2 API are defined here:

Table 24. API Status Code Definitions

Programming Language Interface	Declaration in File	Item
C	types\phTedKitStatus.h	`#define …`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::ReturnCode`

13.1.5 API Layer IDs

The IDs for the different API layers are defined here:

Table 25. API Layer IDs

Programming Language Interface	Declaration in File	Item
C	types\phTedKitStatus.h	`#define …`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::Layer`

13.1.6 Transponder Specifics

All transponder (NXP immobilizer) commands are defined here:

Table 26. Transponder Command Definitions

Programming Language Interface	Declaration in File	Item
C	types\phTedKitCommands.h	`#define …`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::Command`

All transponder (NXP immobilizer) states are defined here:

Table 27. Transponder State Definitions

Programming Language Interface	Declaration in File	Item
C	types\phTedKitCommands.h	`enum { … }`
C++		
C#	TED-Kit 2 API.cs;	`phcs_TedKit2::State`

13.2 Status Codes

The status codes are returned by API each time a function is called. It tells whether the function call was successful or not as well as what went wrong. The following values are defined:

Table 28. Status Code Values

Value	Description
TED-Kit 2 Status Codes (prefix `PHTEDKITSTATUS_`)	
`OK`	Function was successful.
`MSG_RX_TIMEOUT`	The base station was unable to receive anything within a timeout period.
`MSG_RX_ERROR`	The base station was able to receive data but they are corrupted.
`MSG_RX_FRAMESIZE`	The base station was able to receive something from the transponder but the wrong number of bits.
`ERR_INVALID_FUNCTION`	The run method was called with an undefined function ID.

Value	Description
ERR_INVALID_CHECKSUM	The communication from the API to the firmware is corrupted; the checksum of the data exchanged does not match.
ERR_MEM_ALLOC_COMMAND_RX	The firmware is unable to allocate enough memory for data received from the host/API.
ERR_INVALID_PARAM_RANGE	One or more parameters used in the API call are out of their specified range.
ERR_INVALID_PARAM_LENGTH	The length of an (array) parameter does not match.
ERR_NO_XBOARD_IN_XSLOT	Accessing an XBoard which is not available (broken are not existing) in the given XSlot.
ERR_FUNCTION_NOT_SUPPORTED	The function called is currently not supported.
ERR_XBOARD_COMM_FAILURE	The communication with the given XBoard failed.
ERR_MEM_ALLOC_COMMAND_EXEC	The memory allocation failed.
IO_READTIMEOUT	Reading data via the I/O layer timed out.
IO_INVALID_FUNCTION	An invalid I/O layer function was specified.
IO_DOWNLOADFILE_OPEN_ERROR	The download-file could not be opened.
IO_DOWNLOADFILE_DOWNLOAD_ERROR	The download-file could not be downloaded.
BASEAPI_INVALID_FUNCTION	An invalid Base layer function was specified.
BASEAPI_RX_FRAMESIZE_INVALID	The number of data received by the API from the firmware is wrong.
BASEAPI_RX_CHECKSUM_INVALID	The communication from the Firmware to the API is corrupted; the checksum of the data received by the API from the firmware does not match.
BASEAPI_INVALID_BOUNDARY	Wrong usage of array size
EXTAPI_INVALID_DEVICE	The given base station or transponder type during an API call is invalid.
EXTAPI_INVALID_FUNCTION	The run method of the EXT API Layer was called with an undefined function ID (for that layer).
EXTAPI_INVALID_TRANSPONDER_CMD	A bad transponder command was specified.
EXTAPI_INVALID_TRANSPONDER_CHK	HITAG Pro Transponder checksum is not OK.
EXTAPI_INVALID_RX_LENGTH	HITAG Pro Transponder length is not OK.
EXTAPI_INVALID_STATE	Not existing Transponder state.
EXTAPI_INVALID_COMMAND_USE	Not allowed Transponder command in actual Transponder state.
NO_TRANSPONDER_ANSWER	The transponder did not answered on a communication attempt made by the base station.
WRONG_API_LAYER	Not existing API layer
FTDI Error Codes (prefix PHTEDKITSTATUS_FT_)	
INVALID_HANDLE	
DEVICE_NOT_FOUND	
DEVICE_NOT_OPENED	
IO_ERROR	
INSUFFICIENT_RESOURCES	
INVALID_PARAMETER	

Value	Description
INVALID_BAUD_RATE	
DEVICE_NOT_OPENED_FOR_ERASE	
DEVICE_NOT_OPENED_FOR_WRITE	
FAILED_TO_WRITE_DEVICE	
EEPROM_READ_FAILED	
EEPROM_WRITE_FAILED	
EEPROM_ERASE_FAILED	
EEPROM_NOT_PRESENT	
EEPROM_NOT_PROGRAMMED	
INVALID_ARGS	
NOT_SUPPORTED	
OTHER_ERROR	

13.3 XSlot Codes

To access the XSlot (the XBoard actually), a set of constants has been defined. The definition locations are shown in the first table below:

Table 29. XSlot/XBoard Codes

Value	Description
PHTEDKITXBOARD_XSLOT_0	Selects XBoard in XSlot 0.
PHTEDKITXBOARD_XSLOT_1	Selects XBoard in XSlot 1.
PHTEDKITXBOARD_XSLOT_2	Selects XBoard in XSlot 2.
PHTEDKITXBOARD_XSLOT_3	Selects XBoard in XSlot 3.

13.4 API Device Codes

The base and the extension layer of the TED-Kit 2 API use a device code to correctly process and direct the data given via the `phTedKit_BaseData_t` data structure. The following API devices are defined:

Table 30. API Device Codes

Value	Description
Immobilizer Transponders	
PHTEDKITEXTAPIDEVICE_HITAG2	Addresses HITAG2 and HITAG2+.
PHTEDKITEXTAPIDEVICE_HITAG2EXT	Addresses HITAG 2 Extended.
PHTEDKITEXTAPIDEVICE_HITAGPRO	Addresses HITAG Pro and HITAG-Pro 2.
PHTEDKITEXTAPIDEVICE_HITAG3	Addresses HITAG3.
PHTEDKITEXTAPIDEVICE_HITAGAES	Addresses HITAG-AES.
PHTEDKITEXTAPIDEVICE_HITAG2PLUSEE	Addresses HITAG2+EE.
Virtual Devices	
PHTEDKITEXTAPIDEVICE_PKE	Addresses the PKE feature block
PHTEDKITEXTAPIDEVICE_HITAG2CRYPTO	Addresses the HITAG2 crypto unit.
PHTEDKITEXTAPIDEVICE_HITAG3CRYPTO	Addresses the HITAG3 crypto unit.
PHTEDKITEXTAPIDEVICE_AESCRYPTO	Addresses the AES crypto unit.

Value	Description
Base Stations	
PHTEDKITEXTAPIDEVICE_ABIC1	Addresses ABIC1.
PHTEDKITEXTAPIDEVICE_ABIC2	Addresses ABIC2.
PHTEDKITEXTAPIDEVICE_LOPSTER	Addresses LoPSTer.

13.5 Coding Schemes and Word Size

The µC of the TED-Kit 2 supports several coding schemes for data encoding and decoding. The table below lists the coding scheme codes and corresponding word sizes supported:

Table 31. Coding Scheme Codes and Word Size

Value	Word Size (Bits per Value)	Description
PHTEDKITCODING_MANCHESTER	1	Manchester.
PHTEDKITCODING_CDP	1	Conditional De-Phase.
PHTEDKITCODING_BPLM	1	Binary coded Pulse Length Modulation.
PHTEDKITCODING_FREEWAVE	8	Arbitrary coded waveform (high and low level alternate).
PHTEDKITCODING_ANALOG	8/16	Analogue signal with either 8 or 10 bit resolution
PHTEDKITCODING_PLAIN	1	Free wave form signal (arbitrary order of carrier on/off).
PHTEDKITCODING_GPIO	16	16 bits at a time (parallel) via GPIO.

14. Document Management

14.1 Abbreviations and Terminology

The following abbreviations and terminology is used throughout this document:

Table 32. Abbreviations and Terminology

Abbreviation	Description
µC	***M***icro ***C***ontroller
ABIC{1|2}	***A***dvanced ***B***ase station ***IC***{1|2}
AES	***A***dvanced ***E***ncryption ***S***tandard
API	***A***pplication ***P***rogramming ***I***nterface
ASCII	American ***S***tandard ***C***ode for ***I***nformation ***I***nterchange
BPLM	***B***inary coded ***P***ulse ***L***ength ***M***odulation
CDP	***C***onditional ***D***e***p***hase Encoding
DLL	***D***ynamic ***L***ink ***L***ibrary
EEPROM	***E***lectrically ***E***rasable ***P***rogrammable ***R***ead-***O***nly ***M***emory
FTDI	Future Technology Devices International Ltd., http://www.ftdichip.com, provider of the UART/USB converter soft- and hardware.
GPIO	***G***eneral ***P***urpose ***I***nput/***O***utput
HW	***H***ard***w***are
I/O	***I***nput/***O***utput
I^2C	***I***nter-***I***ntegrated ***C***ircuit; a multi-master serial single-ended computer bus invented by Philips
IC	***I***ntegrated ***C***ircuit
ID	***Id***entifier
IDE	***Id****e*ntifier
LED	***L***ight-***E***mitting ***D***iode
LF	***L***ow ***F***requency
LIN	***L***ocal ***I***nterconnect ***N***etwork
LoPSTer	***Lo***w ***P***ower, ***S***ingle-chip ***T***ransce***iver***
PKE	***P***assive ***K***eyless ***E***ntry
RSSI	***R***eceived ***S***ignal ***S***trength ***I***ndicator
R_x	***R***eception
SPI	***S****erial* ***P***eripheral ***I***nterface
SW	***S***oft***w***are
TED-Kit 2	***T***ransponder ***E***valuation and ***D***emonstration***-Kit 2***
T_x	***T***ransmission
UHF	***U***ltra ***H***igh ***F***requency
USB	***U***niversal ***S***erial ***B***us
XBoard	E***x***tension ***Board***
XMA	E***x***tended ***M***emory ***A***ccess
XSlot	E***x***tension ***Slot***

14.2 Referenced Documents

The following documents are referenced throughout this document:

Table 33. Referenced Documents

ID	Title	Version	Issue Date
UM10278_1	TED-Kit 2 Firmware-to-Host Interface Specification	4.04	October 2nd, 2008
1	Run-Method Overview.xls	n/a	n/a

15. Legal Information

15.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

15.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

15.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

15.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

15.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP B.V.

16. List of Examples

17. List of Tables

18. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 June 2011
Document identifier: DOC-091497

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 June 2011
Document identifier: DOC-091497