Freescale USB Stack
OTG API Reference Manual

Document Number: USBOTGAPIRM
Rev. 2
03/2012

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: USBOTGAPIRM
Rev. 2
03/2012

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document.

Revision Number	Revision Date	Description of Changes
Rev. 0	01/2011	Initial release
Rev.1	07/2011	References updated
Rev. 2	03/2012	Replaced the term "Freescale USB Stack with PHDC" with "Freescale USB Stack"

Chapter 1
Before Beginning

Chapter 2
USB OTG API Overview

Chapter 3
USB OTG Layer New Types Definition

Chapter 4
USB OTG Layer API Function Listing

Chapter 1
Before Beginning

1.1 About this book

This book describes the Freescale USB stack OTG device and class API functions. It describes in detail the API functions that can be used to program the USB controller at various levels. Table 1-1 shows the summary of chapters included in this book.

Table 1-1. OTGUSBAPIRM summary

Chapter Title	Description
Before Beginning	This chapter provides the prerequisites of reading this book.
USB OTG Device API Overview	This chapter gives an overview of the API functions and how to use them for developing new class and applications.
USB OTG Layer New Type Definition	This chapter discusses the new type definitions in detail.
USB OTG Layer API Function Listing	This chapter discusses the USB OTG API functions in detail.

1.2 Reference material

Use this book in conjunction with:

- *Freescale USB Stack OTG Users Guide* (document USBOTGUG, Rev. 1)

For better understanding, refer to the following documents:

- USB Specification Revision 1.1
- USB Specification Revision 2.0
- S08 Core Reference
- ColdFire V1 Core Reference
- ColdFire V2 Core Reference
- Kinetis (ARM Cortex-M4) Core Reference
- CodeWarrior Help

1.3 Acronyms and abbreviations

API	Application Programming Interface
IDE	Integrated Development Environment
OTG	On-The-Go
PHDC	Personal Healthcare Device Class
USB	Universal Serial Bus

1.4 Function listing format

This is the general format of an entry for a function, compiler intrinsic, or macro.

function_name()

A short description of what **function_name()** does.

Synopsis

Provides a prototype for function **function_name()**.

```
<return_type> function_name(
  <type_1>  parameter_1,
  <type_2>  parameter_2,
  ...
  <type_n>  parameter_n)
```

Parameters

parameter_1 [in]—Pointer to x

parameter_2 [out]—Handle for y

parameter_n [in/out]—Pointer to z

Parameter passing is categorized as follows:

- *In*—Means the function uses one or more values in the parameter you give it without storing any changes.
- *Out*—Means the function saves one or more values in the parameter you give it. You can examine the saved values to find out useful information about your application.
- *In/out*—Means the function changes one or more values in the parameter you give it and saves the result. You can examine the saved values to find out useful information about your application.

Description

Describes the **function_name()**. This section also describes any special characteristics or restrictions that might apply:

- function blocks or might block under certain conditions
- function must be started as a task
- function creates a task
- function has pre-conditions that might not be obvious
- function has restrictions or special behavior

Return Value

Specifies any value or values returned by **function_name()**.

See Also

Lists other functions or data types related to **function_name()**.

Example

Provides an example (or a reference to an example) that illustrates the use of **function_name()**.

Chapter 2 USB OTG API Overview

2.1 Introduction

USB has traditionally consisted of a host-peripheral topology where the PC was the host and the peripheral was the device. The USB On-The-Go addition to the 2.0 standard defines a way for portable devices to connect to supported USB products in addition to the PC (through only one mini-connector).

The USB OTG new features are:

- A new standard for small form factor USB connectors and cables
- The addition of host capability to products that have been peripheral only
- The ability to be either host or peripheral (dual-role devices) and to dynamically switch between the two

2.2 USB OTG

The USB On-The-Go module is an addition to the current USB IP support to both Host and Device entities. It implements the On-The-Go 2.0 specification and consists of an independent OTG module and changes at the driver and stack level to the current Device and Host protocols to introduce the required functions for USB dual role devices and switching protocols.

The independent OTG module provides the following functions:

- Initialization of the OTG state machine implemented for detecting the device type (Type A or Type B)
- Provides handling for the OTG interrupts.
- Provides implementation for the A-state machine and B-state machine according to the OTG standard
- Provides application indications when OTG events occur
- Loads and unloads dynamically host and peripheral stacks
- Session Request Protocol (SRP) support. This functionality allows the device to request the Host to turn on the power supply on the USB bus at the start of a session.
- Host Negotiation Protocol (HNP) support. Provides functionality to allow the peripheral to become Host when the Host has finished using the Bus.

The usage of the OTG functionality is possible only by starting the OTG standalone driver which handles the transitions to A or B state machines, handles OTG on chip and external interrupts, loads and unloads dynamically host or peripheral stacks, and interacts with the application level through the OTG callback function and through the OTG API.

The following image shows the architecture of the standalone OTG module and its interaction with the application and with the host and peripheral stacks.

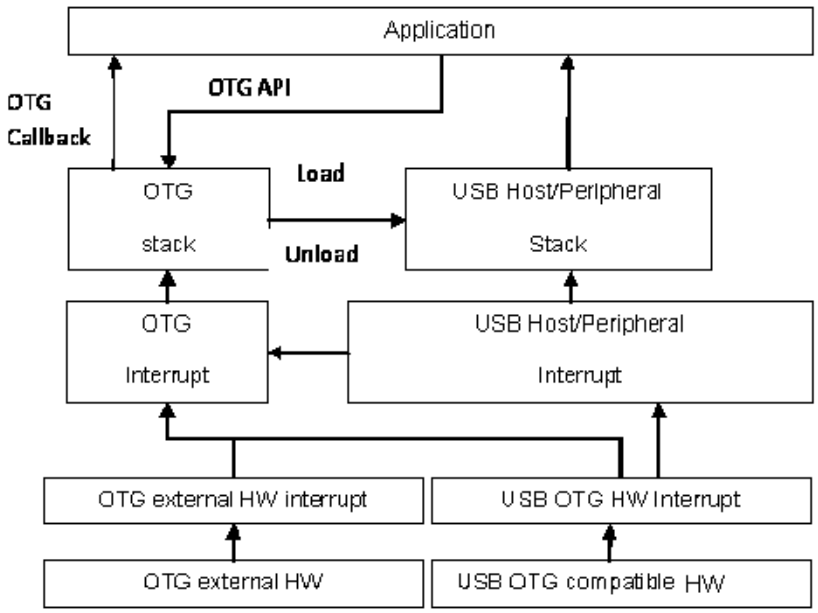

Figure 2-1. Architecture of OTG Module

The OTG callback is an application function which is registered to the OTG stack by using an OTG API function. OTG stack calls this function every time an OTG event occurs and passes it to the application.

Table 2-1 illustrates these events.

Table 2-1. List of events and its description

Event	Description
OTG_B_IDLE	OTG state changes to b_idle which has some sub states
OTG_B_IDLE_SRP_READY	b_idle, SRP ready to start
OTG_B_SRP_INIT	b_idle, SRP init state
OTG_B_SRP_FAIL	b_idle, SRP failed to get a response
OTG_B_PERIPHERAL	OTG state changes to b_peripheral which has some sub states
OTG_B_PERIPHERAL_HNP_READY	b_peripheral, HNP ready to be performed
OTG_B_PERIPHERAL_HNP_START	b_peripheral, HNP started
OTG_B_PERIPHERAL_HNP_FAIL	b_peripheral, HNP failed

Table 2-1. List of events and its description

Event	Description
OTG_B_PERIPHERAL_LOAD_ERROR	Peripheral stack could not be loaded
OTG_B_HOST	OTG state changes to b_host
OTG_B_HOST_LOAD_ERROR	Host stack could not be loaded
OTG_B_A_HNP_REQ	A device requests to become host
OTG_A_IDLE	OTG state changes to a_idle
OTG_A_WAIT_VRISE	OTG state changes to a_wait_vrise
OTG_A_WAIT_BCON	OTG state changes to a_wait_b_con
OTG_A_HOST	OTG state changes to a_host
OTG_A_SUSPEND	OTG state changes to a_suspend
OTG_A_PERIPHERAL	OTG state changes to a_peripheral
OTG_A_WAIT_VFALL	OTG state changes to a_wait_vfall
OTG_A_VBUS_ERR	OTG state changes to a_vbus_err
OTG_A_WAIT_VRISE_TMOUT	a_wait_vrise_tmout expired
OTG_A_WAIT_BCON_TMOUT	a_wait_bcon_tmout expired
OTG_A_BIDL_ADIS_TMOUT	a_bidl_adis_tmout expired
OTG_A_AIDL_BDIS_TMOUT	a_aidi_bdis_tmout expired
OTG_A_B_HNP_REQ	B-device requests to become host
OTG_A_HOST_LOAD_ERROR	Host stack could not be loaded
OTG_A_PERIPHERAL_LOAD_ERROR	Peripheral stack could not be loaded
OTG_A_ID_TRUE	ID input becomes TRUE

The OTG API represents a collection of functions which enable the application to do the following:

- Initialize OTG stack,
- Register the callback function,
- Change dynamically between the host and peripheral roles, and
- Control OTG stack behavior.

2.3 API overview

Table 2-2 describes the list of OTG API functions and their use:

Table 2-2. List of OTG API functions and their use

Functions	Uses
_usb_otg_init()	Initializes OTG stack and OTG hardware
_usb_otg_register_callback()	Registers OTG callback
_usb_otg_session_request()	B-device requests a new session to be started by the A-device
_usb_otg_bus_request()	B-device requests to become Host
_usb_otg_bus_release()	B-device hands over the bus back to the A-device
_usb_otg_task()	OTG task
_usb_otg_ext_isr()	External OTG interrupt software routine
_usb_otg_isr()	Internal OTG interrupt software routine
_usb_otg_set_a_bus_req()	Set the value of the a_bus_req parameter
_usb_otg_set_a_bus_drop()	Set the value of the a_bus_drop parameter
_usb_otg_set_a_clear_err()	Set a_clr_err parameter value TRUE
_usb_otg_on_interface_event	To be called by the host application at interface event
_usb_otg_on_detach_event	To be called by the host application at detach event

2.4 Using API

The following steps explain how to write an OTG application:

1. Write the functions to be passed to the OTG driver through the Initialization structure.

   Example:

   The function that loads the host stack and initializes the host app:

```
USB_STATUS App_Host_Init(void)
{
   USB_STATUS status = USB_OK;
   host_stack_active = TRUE;
   dev_stack_active = FALSE;
   hid_device.DEV_STATE = USB_DEVICE_IDLE;
   DisableInterrupts;
   status = _usb_host_init(HOST_CONTROLLER_NUMBER, MAX_FRAME_SIZE, &host_handle);
    if(status != USB_OK) {
      printf("\nUSB Host Initialization failed. STATUS: %x", status);
      return status;
  }
status = _usb_host_driver_info_register(host_handle, DriverInfoTable);
  if(status != USB_OK)
     {
      return status;
  }
  _usb_event_init(&USB_Event);
  EnableInterrupts;
  printf("\nUSB HID Keyboard Demo\nWaiting for USB Keyboard to be attached...\n");
```

```
  return USB_OK;
 }
```

Example:

The function that unloads the active stack:

```
static void App_ActiveStackUninit(void)
{
 if(dev_stack_active)
 {
  App_PeripheralUninit();
 }
 if(host_stack_active)
 {
  App_Host_Shut_Down();
 }
}
```

2. Declare an OTG handle.

```
_usb_otg_handle      otg_handle;
```

3. Declare an OTG initialization structure and initialize it.

```
static const OTG_INIT_STRUCT otg_init=
{
 TRUE,                                                  /* ext_circuit_use */
 _otg_max3353_enable_disable,                  /* ext_enable_disable_func */
 _otg_max3353_get_status,                        /* ext_get_status_func*/
 _otg_max3353_get_interrupts,                 /*ext_get_interrupts_func */
 _otg_max3353_set_VBUS,                                 /*ext_set_VBUS*/
 _otg_max3353_set_pdowns,                         /* ext_set_pdowns*/
 App_Host_Init,  /* load_usb_host */
 App_PeripheralInit,                                    /* load_usb_device */
 App_Host_Shut_Down,                                   /* unload_usb_device*/
 App_PeripheralUninit,                                 /* unload_usb_device*/
 App_ActiveStackUninit                                 /* unload_usb_active*/
};
```

4. Write the OTG callback. The functions will manage OTG events.

   Example:

```
void App_OtgCallback(_usb_otg_handle handle, OTG_EVENT event)
{
     if(event & OTG_B_IDLE)
     {
         printf("\n\r>B: OTG state change to B idle");
     }
  if(event & OTG_B_PERIPHERAL)
     {
           printf("\n\r>B: OTG state change to B peripheral.");
           printf("\n\r>B: USB peripheral stack initialized.");
     }
  if(event & OTG_B_HOST)
     {
           printf("\n\r>B: OTG is in the Host state");
           printf("\n\r>B: USB host stack initialized.");
     }
  if(event & OTG_B_A_HNP_REQ)
```

```
        {
        if(_usb_otg_bus_release(otg_handle) == USB_OK)
          {
            printf("\n\rBus release");
          }
        else
          {
               printf("\n\rError releasing the bus");
          }
        }

    if(event & OTG_A_WAIT_BCON_TMOUT)
    {
         printf("\n\r>A: OTG_A_WAIT_BCON_TMOUT");
         _usb_otg_set_a_bus_req(otg_handle , FALSE) ;
    }

    if(event & OTG_A_BIDL_ADIS_TMOUT)
    {
         printf("\n\r>A: OTG_A_BIDL_ADIS_TMOUT");
         _usb_otg_set_a_bus_req(otg_handle , TRUE) ;
    }
    if(event & OTG_A_B_HNP_REQ)
    {
         printf("\n\r>A: OTG_A_B_HNP_REQ");
         _usb_otg_set_a_bus_req( handle , FALSE);
    }

    if(event & OTG_A_IDLE)
    {
        printf("\n\r>OTG state change to A_IDLE");
    }

    if(event & OTG_A_HOST_LOAD_ERROR)
    {
        printf("\n\r>A: OTG state change to OTG_A_HOST");
        printf("\n\r>A: USB host stack initialization failed.");
     }
    ..................
    ..................
    }
```

5. Declare the interrupt function for the external OTG circuit and place the _usb_otg_ext_isr call into it.

   Example:

```
void interrupt VectorNumber_Vkeyboard Kbi_ISR(void)
{
    if(!(PTBD & 0x10))
    {
     _usb_otg_ext_isr(0);
     KBI1SC_KBACK = 1;                    /* clear KBI interrupt (for S08)*/
    }
}
```

6. Declare the interrupt function for the USB OTG on chip hardware and place the usb_otg_isr call into it as well as the host and peripheral USB ISR.

Example:

```
void interrupt VectorNumber_Vusb USB_OTG_ISR(void)
{
   _usb_otg_isr(0);
   if(dev_stack_active)
   {
    USB_ISR();
   }
   if(host_stack_active)
   {
    USB_ISR_HOST();
   }
}
```

7. Place a call to _usb_otg_on_detach_event in the function that manages usb host events, in case that detach event is received. Place a call to _usb_otg_on_interface_event in case that interface event is received.

   Example:

```
void usb_host_hid_keyboard_event(
    /* [IN] pointer to device instance */
    _usb_device_instance_handle dev_handle,
    /* [IN] pointer to interface descriptor */
    _usb_interface_descriptor_handle intf_handle,
    /* [IN] code number for event causing callback */
    uint_32 event_code)
{
    INTERFACE_DESCRIPTOR_PTR intf_ptr = (INTERFACE_DESCRIPTOR_PTR) intf_handle;
       switch (event_code) {
       case USB_ATTACH_EVENT:
        ……….
       case USB_CONFIG_EVENT:
       case USB_INTF_EVENT:
        printf("\n\r----- Interfaced Event -----\n");
        _usb_otg_on_interface_event(dev_handle);
        break;
       case USB_DETACH_EVENT:
        printf("\n\r----- Detach Event -----\n");
        …….
        _usb_otg_on_detach_event(dev_handle);
        break;
    }
    /* notify application that status has changed */
    _usb_event_set(&USB_Event, USB_EVENT_CTRL);
}
```

8. Place a call to _usb_otg_init function and a call to _usb_otg_register_callback function in the initialization part of the application.

   Example:

```
status = _usb_otg_init(0, (OTG_INIT_STRUCT*)&otg_init, &otg_handle);
   if(status == USB_OK)
   {
       status = _usb_otg_register_callback(otg_handle, App_OtgCallback);
   }
```

9. Place a call to _usb_otg_task and a call to the active task in the application loop.

Example:

```
for(;;) {
      _usb_otg_task();
         if(dev_stack_active)
        {
                App_PeripheralTask();
        }
        if(host_stack_active)
        {
          App_Host_Task();
        }
            App_HandleUserInput();
      __RESET_WATCHDOG(); /* feeds the dog */

  }                                                          /* loop forever */
```

Chapter 3 USB OTG Layer New Types Definition

3.1 otg_ext_enable_disable

This type defines a pointer to a function that enables/disables the external OTG circuit.

Synopsis

```
typedef void (*otg_ext_enable_disable)(boolean enable);
```

3.2 otg_ext_set_VBUS

This type defines a pointer to a function that enables/disables the VBUS generator.

Synopsis

```
typedef void (*otg_ext_set_VBUS)(boolean a_device);
```

3.3 otg_ext_get_status

This type defines a pointer to a function that gets status from the external circuit.

Synopsis

```
typedef uint_8 (*otg_ext_get_status)(void);
```

3.4 otg_ext_get_interrupts

This type defines a pointer to a function that gets the active interrupts from the external OTG circuit.

Synopsis

```
typedef uint_8 (*otg_ext_get_interrupts)(void);
```

3.5 otg_ext_set_pdowns

This type defines a pointer to a function that activates/deactivates the DP and DM pull downs from the external OTG circuit.

Synopsis

```
typedef uint_8 (*otg_ext_set_pdowns)(uint_8 bitfield);
```

3.6 otg_load_usb_stack

This type defines a pointer to a function that loads the host or peripheral stack and initializes the host or peripheral application.

Synopsis

```
typedef uint_32 (*otg_load_usb_stack)(void);
```

3.7 otg_unload_usb_stack

This type defines a pointer to a function that unloads the host or peripheral stack and finishes the host or peripheral application.

Synopsis

```
typedef void (*otg_unload_usb_stack)(void);
```

3.8 OTG_Init_Struct

This type defines a structure that is a collection of pointers to functions used by the OTG stack to access functions from the external OTG circuit driver and from the OTG application. The address of an instance of this structure is used as a parameter for the _usb_otg_init function.

Synopsis

```
typedef struct otg_init_struct
{
 boolean  ext_circuit_use;
 otg_ext_enable_disable ext_enable_disable_func;
 otg_ext_get_status         ext_get_status_func;
 otg_ext_get_interrupts     ext_get_interrupts_func;
 otg_ext_set_VBUS           ext_set_VBUS;
 otg_ext_set_pdowns         ext_set_pdowns;
 otg_load_usb_stack         load_usb_host;
 otg_load_usb_stack         load_usb_device;
 otg_unload_usb_stack       unload_usb_host;
 otg_unload_usb_stack       unload_usb_device;
 otg_unload_usb_stack       unload_usb_active;
}OTG_INIT_STRUCT;
```

Fields:

- *ext_circuit_use*—Specifies whether the OTG stack uses external circuits or not
- *ext_enable_disable_func*—Pointer to the function that enables/disables the external OTG circuit
- *ext_get_status_func*—Pointer to the function that gets status from the external circuit
- *ext_get_interrupts_func*—Pointer to the function that gets the active interrupts from the external OTG circuit
- *ext_set_VBUS*—Pointer to the function that enables/disables the VBUS generator
- *ext_set_pdowns*—Pointer to the function that activates/deactivates the DP and DM pull downs from the external OTG circuit
- *load_usb_host*—Pointer to the function that loads the host stack and initializes the host application

- *load_usb_device*—Pointer to the function that loads the peripheral stack and initializes the peripheral application
- *unload_usb_host*—Pointer to the function that unloads the host stack and finishes the host application
- *unload_usb_device*—Pointer to the function that unloads the peripheral stack and finishes the peripheral application
- *unload_usb_active*—Pointer to the function that decides which stack (host or peripheral) is active, unloads the respective stack and finishes its application

3.9 otg_event_callback

This type defines a pointer to a callback function called by the OTG stack to communicate OTG events to the application.

Synopsis

```
typedef void    (*otg_event_callback)(_usb_otg_handle handle, OTG_EVENT event)
```

Chapter 4 USB OTG Layer API Function Listing

4.1 _usb_otg_init

Initializes OTG stack and OTG hardware.

Synopsis:

```
uint_32 _usb_otg_init(uint_8 controller_ID, OTG_INIT_STRUCT *init_struct,
_usb_otg_handle *otg_handle)
```

Parameters:

controller_ID[in]—USB/OTG controller number

init_struct[in]—Pointer to the OTG initialization structure

otg_handle[out]—Pointer to _usb_otg_handle

Description:

This function should be called prior to any other function of the OTG API. It verifies the input parameters and if they are correct it allocates memory for the USB_OTG_STRUCT, initializes the structure, passes the pointer to this structure to application through the otg_handle parameter, and initializes the internal (on chip) and external OTG hardware.

Return Value:

- **USB_OK**(success)
- **USB_INVALID_PARAMETER** (for wrong input parameters)
- **USBERR_INIT_FAILED** (if this device controller was already initialized)
- **USB_OUT_OF_MEMORY** (if there is not enough memory to allocate for the USB_OTG_STRUCT)

4.2 _usb_otg_register_callback

Registers the OTG callback.

Synopsis:

```
uint_32    _usb_otg_register_callback(_usb_otg_handle handle, otg_event_callback
callback)
```

Parameters:

handle[in]—OTG handle

callback[in]—pointer to the function that will be called by the OTG stack when an OTG event occurs.

Description:

This function initializes a pointer to a callback function. The callback is used to communicate events from the OTG stack to the application.

Return Value:

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)

4.3 _usb_otg_session_request

B-device requests a new session to be started by the A device.

Synopsis:

```
uint_32 _usb_otg_session_request(_usb_otg_handle handle);
```

Parameters:

handle[in]—OTG handle

Description:

This function modifies a parameter that determines the OTG stack running on a B-device to start SRP.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on an A-device)

4.4 _usb_otg_bus_request

B-device requests to become Host.

Synopsis:

```
uint_32 _usb_otg_bus_request(_usb_otg_handle handle)
```

Parameters:

handle[in]—OTG handle

Description:

This function sets the Host Request Flag in OTG status of the B device and sets a parameter which informs the OTG stack that the B-device wishes to become host. The OTG stack running on the A-device polls B-device for OTG status and when it finds Host Request Flag TRUE it suspends the bus and waits for the B device to start HNP.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on an A-device)

4.5 _usb_otg_bus_release

B-device hands over the bus back to the A device.

Synopsis

```
uint_32 _usb_otg_bus_release(_usb_otg_handle handle);
```

Parameters

handle[in]—OTG handle

Description

This function sets a parameter which informs the OTG stack that B-device does not want to be a host anymore. B-device returns to peripheral and A-device becomes host again.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on an A-device or if the B device is not host)

4.6 _usb_otg_task

OTG task

Synopsis

```
void _usb_otg_task(void);
```

Parameters

None

Description

This function is the OTG task. It must be called in the application loop to have the OTG stack running.

Return Value

None

4.7 _usb_otg_ext_isr

External OTG interrupt software routine.

Synopsis

```
void _usb_otg_ext_isr(uint_8 controller_ID)
```

Parameters

controller_ID[in]—USB/OTG controller number

Description

This function must be called from the interrupt routine associated with the external OTG hardware (example: MAX3353). Since this interrupt can be tied to many interrupt sources (keyboard, irq), it is the application responsibility to call this function from the configured interrupt routine and to clear the respective interrupt flag.

Return Value

None

4.8 _usb_otg_isr

Internal OTG interrupt software routine.

Synopsis

```
void _usb_otg_isr(uint_8 controller_ID)
```

Parameters

controller_ID[in]—USB/OTG controller number

Description

This function is the interrupt software routine for the on chip part of the OTG hardware. This function must be called from the USB interrupt routine since OTG on chip hardware and USB hardware shares the same interrupt vector.

Return Value

None

4.9 _usb_otg_set_a_bus_req

Sets the value of the a_bus_req parameter.

Synopsis

```
uint_32 _usb_otg_set_a_bus_req(_usb_otg_handle otg_handle, boolean a_bus_req)
```

Parameters

handle[in]—OTG handle

a_bus_req[in]—The new value of the a_bus_req parameter

Description

This function is called from the application to set/clear the a_bus_req parameter. This is one of the parameters that determine A state machine behavior. If the A device is in peripheral state the OTG status changes to USB_OTG_HOST_REQUEST_FLAG.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on a B-device)

4.10 _usb_otg_set_a_bus_drop

Sets the value of the a_bus_drop parameter.

Synopsis

```
uint_32 _usb_otg_set_a_bus_drop(_usb_otg_handle otg_handle, boolean a_bus_drop);
```

Parameters

handle[in]—OTG handle

a_bus_drop[in]—The new value of the a_bus_drop parameter

Description

This function is called from the application to set/clear the a_bus_drop parameter. This is one of the parameters that determine A state machine behavior.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on a B-device)

4.11 _usb_otg_set_a_clear_err

Sets a_clr_err parameter value TRUE.

Synopsis

```
uint_32 _usb_otg_set_a_clear_err(_usb_otg_handle otg_handle)
```

Parameters

handle[in]—OTG handle

Description

This function is called from the application to set the a_clr_err parameter which is one way to exit from the a_vbus_err state. The other two are id = FALSE and a_bus_drop = TRUE.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the OTG handle)
- **USBOTGERR_INVALID_REQUEST** (if the function is called on a B-device)

4.12 _usb_otg_on_interface_event

To be called by the host application at interface event.

Synopsis

```
uint_32 _usb_otg_on_interface_event(void*  dev_handle) ;
```

Parameters

dev_handle[in]—Attached device handle

Description

This function is called from the host application at interface event. The function sets the dev_inst_ptr pointer in the status struct to the (DEV_INSTANCE_PTR) dev_handle value after dev_handle value was checked and found to be valid. The dev_inst_ptr value will be used in the OTG state machine to poll the peripheral for HNP request.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the device handle, if the DEV_INSTANCE structure that device handle points to does not contain a valid host handle)

4.13 _usb_otg_on_detach_event

To be called by the host application at detach event.

Synopsis

```
uint_32 _usb_otg_on_detach_event(void*  dev_handle)
```

Parameters

dev_handle[in]—Dettached device handle

Description

This function is called from the host event function in the host application at detach event. The function resets all peripheral related parameters in the OTG state structure if the host event function was called due to a detach event. The function does not take any actions if the host event function was called due to a host stack unload.

Return Value

- **USB_OK** (success)
- **USB_INVALID_PARAMETER** (if a NULL pointer is passed for the device handle, if the DEV_INSTANCE structure that device handle points to does not contain a valid host handle)