MCCI Corporation
3520 Krums Corners Road
Ithaca, New York 14850 USA
Phone +1-607-277-1029
Fax +1-607-277-6844
www.mcci.com

MCCI USB DataPump Embedded Host Class Driver Development Guide

Engineering Report 950000761
Rev. C
Date: 2011/09/30

PROPRIETARY NOTICE AND DISCLAIMER

Unless noted otherwise, this document and the information herein disclosed are proprietary to MCCI Corporation, 3520 Krums Corners Road, Ithaca, New York 14850 ("MCCI"). Any person or entity to whom this document is furnished or having possession thereof, by acceptance, assumes custody thereof and agrees that the document is given in confidence and will not be copied or reproduced in whole or in part, nor used or revealed to any person in any manner except to meet the purposes for which it was delivered. Additional rights and obligations regarding this document and its contents may be defined by a separate written agreement with MCCI, and if so, such separate written agreement shall be controlling.

The information in this document is subject to change without notice, and should not be construed as a commitment by MCCI. Although MCCI will make every effort to inform users of substantive errors, MCCI disclaims all liability for any loss or damage resulting from the use of this manual or any software described herein, including without limitation contingent, special, or incidental liability.

MCCI, TrueCard, TrueTask, MCCI Catena, and MCCI USB DataPump are registered trademarks of MCCI Corporation.

MCCI Instant RS-232, MCCI Wombat and InstallRight Pro are trademarks of MCCI Corporation.

All other trademarks and registered trademarks are owned by the respective holders of the trademarks or registered trademarks.

NOTE: The code sections presented in this document are intended to be a facilitator in understanding the technical details. They are for illustration purposes only, the actual source code may differ from the one presented in this document.

Document Release History

Rev. A	2009/11/13	Original release
Rev. B	2011/06/13	Changed document numbers to nine digit versions. DataPump 3.0 Updates.
Rev. C	2011/09/30	Added source code disclaimer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Purpose

This document provides technical training on the internal details of the implementation of MCCI USB DataPump Embedded Host Class Drivers. The suggested audiences are engineers who have to understand, update, and debug existing Class Drivers codes and especially for engineers who have to develop new Class Drivers.

1.2 Scope

This document illustrates how to create a Class Driver Framework from the Sample Class Driver and how to customize the Class Driver Framework for your Class Driver. It also describes the details of the Class Driver Development Kit (ClassKit).

This document assumes familiarity with the MCCI USB DataPump.

1.3 Glossary

{HOME}	Home directory of the Class Driver. This is usbkern/host/class/{Your Class Driver Name}
ClassKit	Class Driver Development Kit. This is the software component that provides common routines to Class Drivers.
Class Driver	MCCI USB DataPump Embedded Host Class Driver. This provides low-level access to USB devices.
Class Driver Framework	The source tree that provides a skeletal structure of a Class Driver that is based on the ClassKit. This is generated from the Sample Class Driver.
Device	The hardware component that provides the USB descriptors and data to Class Driver.
FSM	Finite State Machine
Sample Class Driver	The source tree that has skeleton source codes and headers.

1.4 Referenced Documents

[DPREF]	*MCCI USB DataPump User's Guide,* MCCI Engineering Report 950000066
[MOB]	*MCCI Object Brokerage Specification*, MCCI Engineering report 950000961
[USBDI]	*MCCI USB DataPump Embedded USBDI*, MCCI Engineering report 950000325

2 Procedure To Develop New Class Driver

2.1 Create Class Driver Framework From Sample Class Driver

The Sample Class Driver code resides at usbkern/host/class/sample. The Sample Class Driver code contains the following directories.

Table 1 Sample Class Driver Directory Structure

Directory	Description
Common	Common code for Sample Class Driver
Doc	Converting files from Sample Class Driver to specific Class Driver
I	Include files

Refer to usbkern/host/class/sample/doc/Readme.txt for the detailed description of how to create Class Driver Framework from the Sample Class Driver.

2.2 Modify Class Driver Configuration Header

2.2.1 Modify Configuration Tags

The configuration structure of a Class Driver begins and ends with 4-byte tags. These tags are used to validate a VOID * type configuration. The created {HOME}/i/xxxx_config.h file contains the following lines for these tags.

```
#define   __TMS_USBPUMP_USBDI_CLASS_GENERIC_CONFIG_MAGIC_BEGIN \
          __TMS_UHIL_MEMTAG('<', 'T', 's', 'd')
#define   __TMS_USBPUMP_USBDI_CLASS_GENERIC_CONFIG_MAGIC_END   \
          __TMS_UHIL_MEMTAG('>', 'T', 's', 'd')
```

The '<', 'T', 's', 'd' means “Tag of sample Class Driver” so you have to modify this to a proper tag name to represent the Class Driver you are developing. For example, if you are developing CDC ACM Class Driver, you may choose '<', 'T', 'a', 'd', which stands for “Tag of ACM Class Driver”. If you modify the beginning tag, you must modify the ending tag as well.

2.2.2 Add the Class Driver specific configuration information

You can put additional Class Driver specific configuration information in this configuration. The default member of the configuration structure is the MaxSession that is the maximum number of sessions that the Class Driver supports.

The session is a virtual connection between a supplier and a consumer. (For the detail about the supplier/consumer concept, refer to [MOB].) The Class Driver is usually a supplier that provides a class interface and a function interface. So there are two kinds of sessions; one is a class session and the other is a function session. The number of the sessions counts both class and function sessions. So the sum of the class and function sessions must be equal to or less than the MaxSession value. When the sum of the sessions is equal to the MaxSession value, a command to open a class or function session returns NO_MORE_SESSIONS status code. For more about the status codes, refer to the section 3.1 in this document.

If you add any extra configuration information in the structure, you have to modify the XXXX_CONFIG_INIT_V1() and XXXX_CONFIG_SETUP_V1() macros accordingly. The configuration object may be initialized either at compile time or at runtime. For compile-time initialization, use XXXX_CONFIG_INIT_V1() and for run-time initialization of the configuration structure, use XXXX_CONFIG_SETUP_V1().

Whenever you modify header files, you have to generate the uncloaking #defines by using the script "uncloak-defs.sh" (follow the instructions below).

1) Execute "uncloak-defs.sh {input file} > some-temp-file.txt"
2) Copy text from the temp file into the target file.

The uncloaking, by convention, always appears last in the file.

2.3 Generate New GUID

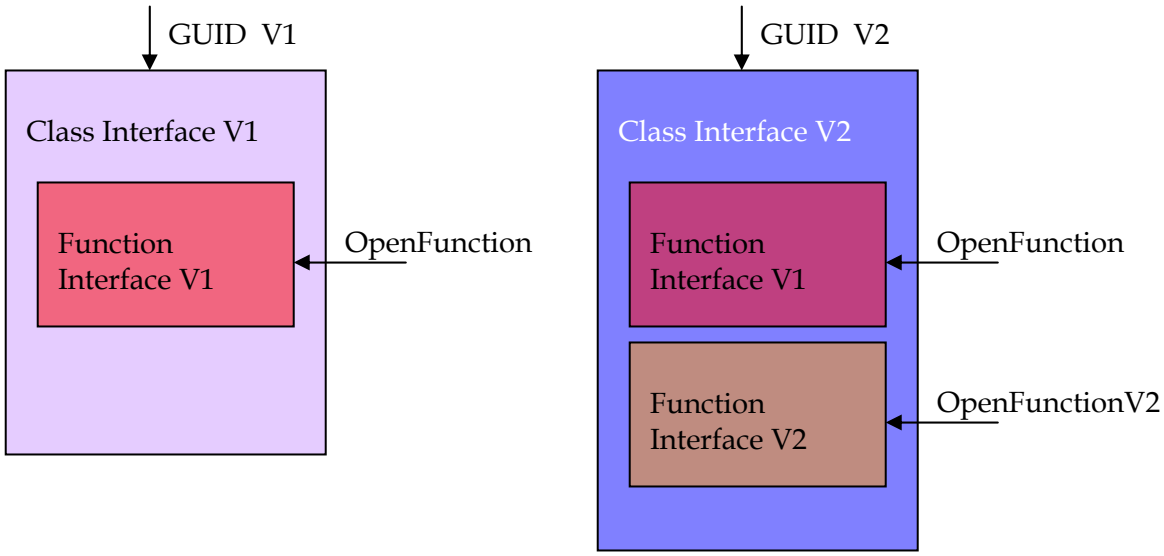
You can generate a new GUID using "uuidgen –s" command in the TTK shell.

2.4 Define Class Driver's Interfaces

A Class Driver object exports an interface – class interface. The class interface provides APIs for the Class Driver object itself. The function object that the USBD creates, provides API for the Class Driver function instance that is bound to a USB device and this API is a function interface. The class interface conforms to [MOB], and thus supports interface versioning by virtue of the GUID. However, the function instance exports a function interface in different way, and the function interface does not conform to [MOB].

The Class Driver object can have multiple versions of the class interface. If the class interface is extended then there will be a V2 (version 2) of that interface. Each version of the class interface has its own GUID. If there are two versions of the class interfaces - V1 (Original version), V2 (Extended version) -, there will be 2 GUIDs for the class interfaces, the GUID representing V1 interface, and the GUID representing V2 one. The Class Driver object should keep V1 interface to provide backward compatibility. A client that wants to use the V1 class interface should ask the Class Driver object using the GUID for the V1 interface. The other clients that want to exchange V2 interface (In/Out-Calls) have to request the interface using the GUID for the V2 interface.

However the function interface behavior is different because it's not compliant to [MOB]. At first, the function interface doesn't have any GUID, so a client has to call a special Class In-Call (OpenFunction) of the class interface to obtain the function interface. To extend and update the function interface to version 2, the class interface must be versioned to V2, and it should expose OpenFunctionV2 Class In-Call as well as OpenFunction. For example, if a client wants to obtain a V2 of the function interface, the client first has to get the V2 of the class interface using GUID for V2 class interface. And then the client should call OpenFunctionV2 Class In-Call to get the V2 function interface. Refer to the below illustration for this concept.



Each interface has In-Calls and Out-Calls. Refer to the [MOB] for the concept of In-Calls and Out-Calls.

2.4.1 Class In-Calls

The Class Driver Framework has four Class In-Calls by default. These In-Calls are mandatory only if your Class Driver exposes a class interface and a function interface. If your Class Driver doesn't provide the interfaces to any clients, you don't have to keep these default In-Calls. Otherwise, you should keep these In-Calls to provide the interfaces. The default Class In-Calls run as follows.

Table 2 Default Class In-Calls

Class In-Call	Description
Close	This Class In-Call closes the session opened to this class interface. This function is a counterpart of the UsbPumpObject_OpenSession(). The prototype of the close function is declared in the [MOB]. The ClassKit defines this function. For the description of this function, refer to the section 3.2.1 in this document.

Class In-Call	Description
Open Function	This Class In-Call opens a function session to a specific Class Driver function instance. A Class Driver function instance is bound to the USB device, so a client must open a function session to handle a specific USB device. The ClassKit declares and defines this function. For the description of this function, refer to the section 3.2.2 in this document.
Get Number of Devices	This Class In-Call returns the number of functions to this Class Driver. The ClassKit declares and defines this function. For the description of this function, refer to the section 3.2.3 in this document.
Get Bound Device List	This Class In-Call returns a list of functions. A function is a representation of a USB device bound to this Class Driver. A client has to provide a buffer for the function list and the bound function list will be returned in the callback function. The ClassKit declares and defines this function. For the description of this function, refer to the section 3.2.4 in this document.

You can define any Class Driver specific In-Calls besides these four default In-Calls. For example, if you want to introduce a Class In-Call that returns USBD's features such as Isochronous transfer support or Transaction Translation support, you have to declare the prototypes of the Class In-Call and its callback function in the {HOME}/i/xxxx_api.h. In addition, you should add the function pointer into the Class In-Call structure and modify INIT and SETUP macros for the Class In-Calls structure accordingly.

For example, if you want to add two Class In-Calls, the ***Bold and Italic font*** below show how to make this change.

1) Add the function pointer members for the In-Calls to the In-Call structure

```
__TMS_TYPE_DEF_STRUCT      (USBPUMP_USBDI_CLASS_XXXX_INCALL);
struct __TMS_STRUCTNAME    (USBPUMP_USBDI_CLASS_XXXX_INCALL)
       {
       __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_HEADER;

       /* XXXX Class Driver specific In-Calls */
       __TMS_USBPUMP_USBDI_XXXX_GET_USBD_FEATURE_FN *       pGetUsbdFeatureFn;
       __TMS_USBPUMP_USBDI_XXXX_GET_DRIVER_FEATURE_FN *     pGetDrvFeatureFn;

       __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_TRAILER;
       };
```

2) Add parameters of In-Call initialization macro

```
#define      __TMS_USBPUMP_USBDI_CLASS_XXXX_INCALL_INIT_V1(            \
       a_pCloseFn,                                                    \
       a_pOpenFunctionFn,                                             \
       a_pGetNumDevicesFn,                                            \
       a_pGetBoundDevicesFn,                                          \
       a_pGetUsbdFeatureFn,                                           \
       a_pGetDrvDeatureFn                                             \
```

```
        )                                                               \
        {{                                                              \
        __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_HEADER_INIT_V1(             \
                sizeof(__TMS_USBPUMP_USBDI_CLASS_XXXX_INCALL),          \
                a_pCloseFn,                                             \
                a_pOpenFunctionFn,                                      \
                a_pGetNumDevicesFn,                                     \
                a_pGetBoundDevicesFn                                    \
                ),                                                      \
        a_pGetUsbdFeatureFn,                                            \
        a_pGetDrvFeatureFn,                                             \
        __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_TRAILER_INIT_V1(            \
                __TMS_USBPUMP_API_INCALL_MAGIC_END                      \
                )                                                       \
        }}
```

3) Add parameters of In-Call setup macro

```
#define       __TMS_USBPUMP_USBDI_CLASS_XXXX_INCALL_SETUP_V1(           \
        a_pInCall,                                                      \
        a_pCloseFn,                                                     \
        a_pOpenFunctionFn,                                              \
        a_pGetNumDevicesFn,                                             \
        a_pGetBoundDevicesFn,                                           \
        a_pGetUsbdFeatureFn,                                            \
        a_pGetDrvDeatureFn                                              \
        )                                                               \
    do {                                                                \
        __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_HEADER_SETUP_V1(            \
                a_pInCall,                                              \
                sizeof(*(a_pInCall)),                                   \
                a_pCloseFn,                                             \
                a_pOpenFunctionFn,                                      \
                a_pGetNumDevicesFn,                                     \
                a_pGetBoundDevicesFn                                    \
                );                                                      \
        (a_pInCall)->XXXXDrv.pGetUsbdFeatureFn = a_pGetUsbdFeatureFn;   \
        (a_pInCall)->XXXXDrv.pGetDrvFeatureFn = a_pGetDrvFeatureFn;     \
        __TMS_USBPUMP_CLASSKIT_CLASS_INCALL_TRAILER_SETUP_V1(           \
                a_pInCall,                                              \
                XXXXDrv,                                                \
                __TMS_USBPUMP_API_INCALL_MAGIC_END                      \
                );                                                      \
        } while (0)
```

2.4.2 Class Out-Calls

The Class Driver Framework has one Class Out-Call by default. This Out-Call is used for your Class Driver to send class event notifications to clients that open a class session to the Class Driver's class interface. The default Class Out-Call runs as follows.

Table 3 Default Class Out-Calls

Class Out-Call	Description
Notification	This Class Out-Call sends a class event notification to clients that open a class session to the Class Driver. The ClassKit declares and defines this function. For the description of this function. For the description of this function, refer to the section 3.3.1 in this document.

You can define any Class Driver specific Out-Calls besides the default Out-Call. If you want to do this, you have to declare the prototypes of the Class Out-Calls in the {HOME}/i/xxxx_api.h. In addition, you should add the function pointer into the Class Out-Call structure and modify INIT and SETUP macros for the Class Out-Calls structure accordingly.

For example, if you want to add one Class Out-Call, the ***Bold and Italic font*** below show how to make this change.

1) Add the function pointer members for the In-Calls to the In-Call structure

```
__TMS_TYPE_DEF_STRUCT      (USBPUMP_USBDI_CLASS_XXXX_OUTCALL);
struct __TMS_STRUCTNAME    (USBPUMP_USBDI_CLASS_XXXX_OUTCALL)
      {
      __TMS_USBPUMP_CLASSKIT_CLASS_OUTCALL_HEADER;

      /* XXXX Class Driver specific Out-Calls */
      __TMS_USBPUMP_USBDI_XXXX_FOO_FN *       pFooFn;

      __TMS_USBPUMP_CLASSKIT_CLASS_OUTCALL_TRAILER;
      };
```

2) Add parameters of In-Call initialization macro

```
#define      __TMS_USBPUMP_USBDI_CLASS_XXXX_OUTCALL_INIT_V1(            \
      a_pNotificationFn,                                               \
      a_pFooFn                                                         \
      )                                                                \
      {{                                                               \
      __TMS_USBPUMP_CLASSKIT_CLASS_OUTCALL_HEADER_INIT_V1(             \
            sizeof(__TMS_USBPUMP_USBDI_CLASS_XXXX_OUTCALL),            \
            a_pNotificationFn                                          \
            ),                                                         \

      a_pFooFn,                                                        \
```

```
        __TMS_USBPUMP_CLASSKIT_FUNCTION_OUTCALL_TRAILER_INIT_V1(          \
                __TMS_USBPUMP_API_OUTCALL_MAGIC_END                       \
                )                                                         \
        }}
```

3) Add parameters of In-Call setup macro

```
#define       __TMS_USBPUMP_USBDI_CLASS_XXXX_OUTCALL_SETUP_V1(          \
        a_pOutCall,                                                     \
        a_pNotificationFn,                                              \
        a_pFooFn                                                        \
        )                                                               \
    do {                                                                \
        __TMS_USBPUMP_CLASSKIT_CLASS_OUTCALL_HEADER_SETUP_V1(           \
                a_pOutCall,                                             \
                sizeof(*(a_pOutCall)),                                  \
                a_pNotificationFn                                       \
                );                                                      \

        (a_pOutCall)->XXXXDrv.pFooFn = a_pFooFn;                        \

        __TMS_USBPUMP_CLASSKIT_FUNCTION_OUTCALL_TRAILER_SETUP_V1(       \
                a_pOutCall,                                             \
                XXXXDrv,                                                \
                __TMS_USBPUMP_API_OUTCALL_MAGIC_END                     \
                );                                                      \
        } while (0)
```

2.4.3 Function In-Calls

The Class Driver Framework has one Function In-Call by default. The default Class In-Call runs as follows.

Table 4 Default Function In-Calls

Function In-Call	Description
Close	This Function In-Call closes the function session opened to this function interface. This function is a pair function with the open function Class In-Call. The ClassKit declares and defines this function. For the description of this function, refer to the section 3.4.1 in this document.

You can define additional Function In-Calls like the section 2.4.1 describes.

2.4.4 Function Out-Calls

The Class Driver Framework has one Function Out-Call by default. This Out-Call is used for your Class Driver to send function event notifications to clients that open a function session to the Class Driver's function interface. The default Function Out-Call runs as follows.

Table 5 Default Function Out-Calls

Function Out-Call	Description
Notification	This Function Out-Call sends a function event notification to clients that open a function session to the Class Driver function instance. The ClassKit declares and defines this function. For the description of this function. For the description of this function, refer to the section 3.5.1 in this document.

You can define additional Function Out-Calls like the section 2.4.2 describes.

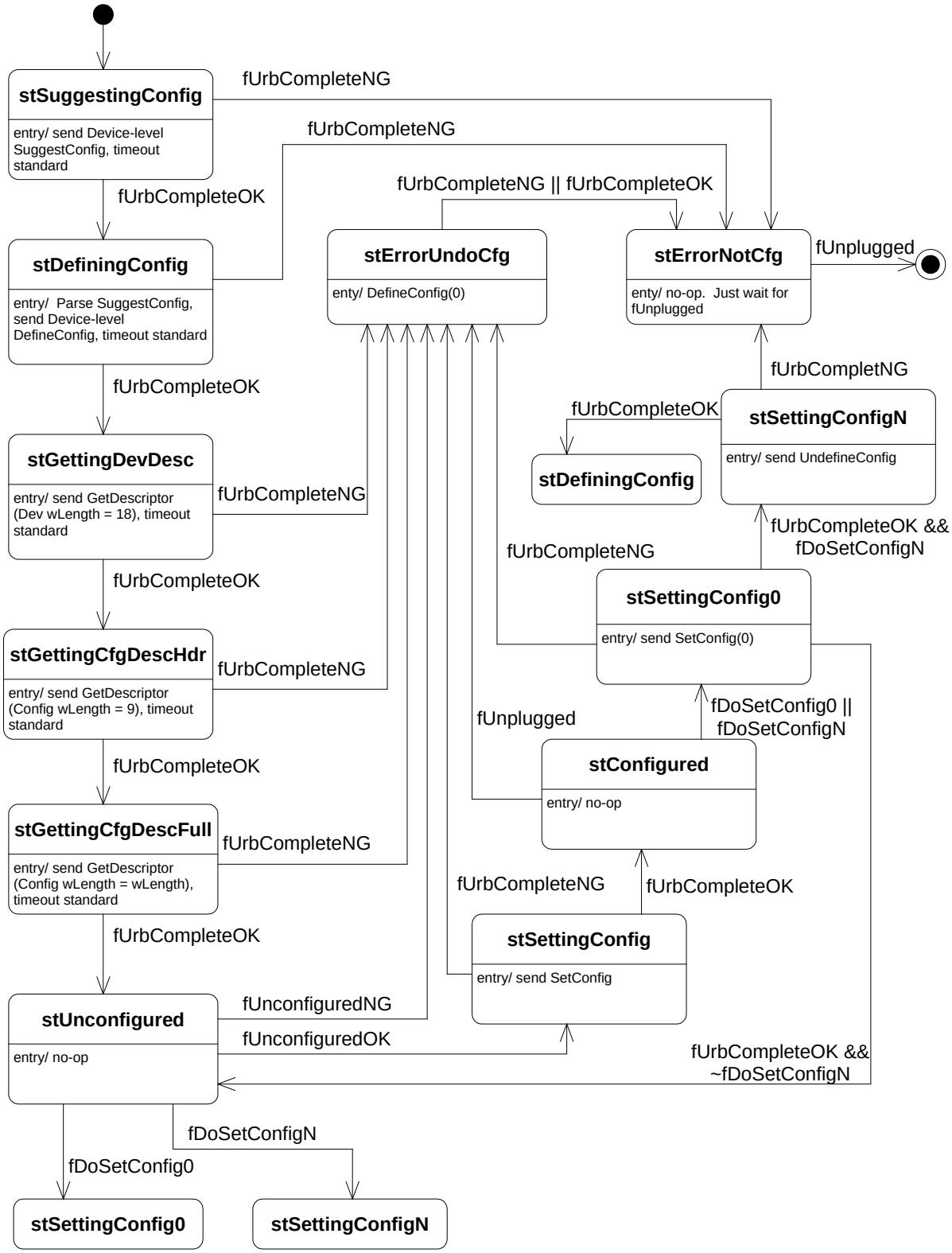
2.5 Implement Sub FSMs

The Class Driver Framework uses the ClassKit FSM. The state diagram of the class FSM is in Figure 1. A Class Driver that uses the Class Driver Framework has two sub-FSMs; one is the unconfigured state sub-FSM (which is optional), the other is the configured state sub-FSM (which is mandatory). If you don't want to perform any additional process in the unconfigured state, you should delete xxxx_UnconfiguredProcess() and xxxx_UnconfiguredSwitch() functions in the {HOME}/common/uxxxx_fsm.c file. If you want to perform an extra process in the unconfigured state, you have to modify {HOME}/common/uxxxx_fsm.c and {HOME}/i/usbpump_xxxx_fsm.h. For the detail of this function, refer to section 3.10.2 in this document.

The configured state sub-FSM that the Class Driver Framework provides by default calls UsbPumpClassKitI_ClassKitFsm_SetEnumerated() to make the ClassKit send the DEVICE_ARRIVAL notification to the clients. If you want to perform additional process in the configured state, you have to modify {HOME}/common/uxxxx_fsm.c and {HOME}/i/usbpump_xxxx_fsm.h. For the detail of this function, refer to section 3.10.3 in this document.

The ClassKit notifies every change of state of the ClassKit FSM by calling xxxx_StateChange() function. If you wish to be notified whenever the state of the ClassKit FSM changes, you must implement xxxx_StateChange(). For the detail of this function, refer to section 3.10.4 in this document.

Figure 1 State Diagram of ClassKit FSM



2.6 Declare Class Driver Specific Contents

If necessary, you can declare contents which are specific to your Class Driver in the USBPUMP_USBDI_FUNCTION_XXXX_CONTENTS structure located in the {HOME}/common/usbpump_xxxx_implementation.h file.

2.7 Declare Class Driver Specific Internal Functions

You can declare internal functions specific to your Class Driver in the {HOME}/common/usbpump_xxxx_implementation.h file, if they are necessary.

2.8 Customize Memory Pool Size

If you perform a text search of "XXX" at the {HOME}/common/ucxxxx_init.c, you can find out where you have to customize the constant values according to the sort of USB device class which the Class Driver supports.

For example, if you are developing the CDC ACM Class Driver which has only one configuration, three interfaces, and no additional alternate settings for each interface: Interface 1 has one interrupt endpoint, Interface 2 has a pair of bulk in and out endpoints, and Interface 0 has no endpoint. ***`The bold and italic font`*** below shows this customization.

1) Customize the configuration tree size

```
/* XXX: make valid config tree size */
#define   USBPUMP_USBDI_HWA_CONFIG_TREE_SIZE           \
              ({numCFG} * sizeof(__TMS_USBPUMP_USBDI_CFG_NODE) +    \
               {numIFC} * sizeof(__TMS_USBPUMP_USBDI_IFC_NODE) +    \
               {numALT} * sizeof(__TMS_USBPUMP_USBDI_ALTSET_NODE) + \
               {numPIPE} * sizeof(USBPUMP_USBDI_PIPE_NODE)     \
              )
```

numCFG is 1, numIFC is 3, numALT is 3 and numPIPE is 3.

2) Customize the Class Driver pool size

```
sizeClassPool =
          pXXXXConfig->MaxSession * sizeof(USBPUMP_CLASSKIT_SESSION) +
          /* XXX: pool overhead if allocate more than 3 times */ 0 +
          /* XXX: generic Class Driver pool size */ 0;
```

The class pool exists only once for the Class Driver. By default, the class pool is used only for allocating sessions. However, if you want to use this pool for some other purpose, you should enlarge the pool size by adding non-zero value to the `sizeClassPool.`

3) Customize the Class Driver function instance pool size

The instance pool exists for each Class Driver function instance. That is, the number of instance pools is equal to the number of the USB devices bound to this Class Driver. The instance pool consists of a space for storing the configuration tree, the URB used in the ClassKit FSM, a configuration bundle, pool overhead, and customizable space for Class Driver specific data.

If you want the Class Driver to use additional URBs other than the URB for the ClassKit FSM, you should allocate a larger instance pool. For example, assume that this CDC-ACM Class Driver needs two more URBs: one for the interrupt endpoint, and another for a bulk in/out pair, and you expect at most 128 bytes configuration bundle size for USB devices the Class Driver supports. Then your code would look something like:

```
sizeInstancePool =
          USBPUMP_USBDI_GENERIC_CONFIG_TREE_SIZE +
          sizeof(USBPUMP_URB) + USBPUMP_USBDI_URB_EXTRA_SIZE_DEFAULT +
          /* XXX: configuration descriptor bundle size */ 128 +
          /* XXX: pool overhead if allocate more than 2 times */ 0 +
          /* XXX: ACM Class Driver function instance pool size */
          2 * (sizeof(USBPUMP_URB) + USBPUMP_USBDI_URB_EXTRA_SIZE_DEFAULT);
```

4) Customize the number of pipes per instance and the size of management pool.

```
/* XXX: should set correct information */
USBPUMP_USBDI_DRIVER_CLASS_IMPLEMENTATION_SETUP_V1(               \
          &XXXXClassImplementation,                               \
          sizeof(USBPUMP_USBDI_CLASS_XXXX),                       \
          sizeof(USBPUMP_USBDI_FUNCTION_XXXX),                    \
          USBPUMP_USBDI_CLASS_XXXX_NAME,                          \
          USBPUMP_USBDI_FUNCTION_XXXX_NAME,                       \
          /* Class Driver IOCTL handler */                        \
              UsbPumpUsbdXXXXI_ClassIoctl,                        \
          /* function (instance) IOCTL handler */                 \
              UsbPumpUsbdXXXXI_FunctionIoctl,                     \
          /* number of pipes per instance -- 3 is typical */ 3,   \
          /* number of bytes of class pool */ sizeClassPool,      \
          /* number of bytes of instance pool */ sizeInstancePool, \
          /* management pool size: use default */ 512            \
          );
```

2.9 Fix Makefile

The Class Driver source files reside in {HOME}/common. If you've added source files to this directory, you have to add the source file list you added at UsbMakefile.inc file.

3 ClassKit APIs

This section describes the ClassKit interface. Much of the common work that is necessary for any Class Driver has been abstracted into the ClassKit module. Using the ClassKit speeds the development of new Class Drivers.

3.1 ClassKit Status Codes

These status codes are used in Class In-Calls and Function In-Calls. Usually they are returned at the callback routines of the In-Calls. The prefix of the status codes of Table 6 is "USBPUMP_CLASSKIT_STATUS_".

Table 6 ClassKit Status Codes

Status Code	Description
OK	The ClassKit returns this status code when it handles an In-Call successfully.
INVALID_PARAMETER	This status code is returned at the open session API and open function API when a client passed invalid parameters like NULL In-Call buffer.
ARG_AREA_TOO_SMALL	This status code is return when the open request memory is too small.
BUFFER_TOO_SMALL	This status code is returned at the open session API when the In-Call buffer size and the Out-Call buffer size are too small.
NOT_SUPPORTED	This status code is returned when a client invokes an unsupported In-Call.
NO_MORE_SESSIONS	The ClassKit returns this status code when a client tries to open a class or function session and there is no free session to open. The maximum number of sessions is configured by the Class Driver configuration.
INVALID_SESSION_HANDLE	The ClassKit returns this status code if the return value of UsbPumpClassKitI_ValidateSessionHandle() to the session passed into a Class In-Call is NULL. For the session handle validation API, refer to the section 3.7.1 in this document.
INVALID_FUNCTION_HANDLE	The ClassKit returns this status code if the return value of UsbPumpClassKitI_ValidateSessionHandle() to the session passed into a function In-Call is NULL.
FUNCTION_ALREADY_OPENED	The ClassKit returns this status code if a client tries to open a function session to a specific function instance when: 1) The Class Kit has not been properly configured to support multiple sessions, and 2) Another client has already opened a session to the specific function instance The support of multiple sessions is decided during initializing the ClassKit. Refer to the section 3.5.1 in this document for the ClassKit initializing API.

Status Code	Description
FUNCTION_NOT_OPENED	The ClassKit returns this status code when a client calls a Function In-Call using a function session that is already closed or never opened.

3.2 Class In-Calls

3.2.1 Close Session In-Call

Description

This function closes the session opened at UsbPumpObject_OpenSession(). If a client doesn't want to use the session any longer, the client should call this Class In-Call API to de-allocate the session resource.

Declaration of Prototype

The prototype of this API is declared in the [MOB].

Implementation

```
__TMS_USBPUMP_API_CLOSE_FN
UsbPumpClassKitI_CloseSession;
```

The ClassKit has routines for maintaining the class sessions and the implementation of this API. The developer who develops a new Class Driver using the ClassKit can simply set the pointer of `UsbPumpClassKitI_CloseSession()` in the Class In-Calls of the Class Driver. The Class Driver Framework sets this function pointer in its Class In-Calls.

3.2.2 Open Function In-Call

Description

This function opens a function session to a specific Class Driver function instance from Class Driver. The function instance binds to the USB device. To call function APIs for handling the USB device, a Client has to call this API to open a function session. The Client will then receive a function handle to the Class Driver function instance through the callback function of this API. The function handle is necessary to call Function In-Calls. This API is also one of the Class In-Calls.

This API will return one of the following return codes through its callback routine.

- `USBPUMP_CLASSKIT_STATUS_OK`
- `USBPUMP_CLASSKIT_STATUS_INVALID_SESSION_HANDLE`
- `USBPUMP_CLASSKIT_STATUS_INVALID_PARAMETER`
- `USBPUMP_CLASSKIT_STATUS_FUNCTION_ALREADY_OPENED`

- USBPUMP_CLASSKIT_STATUS_NO_MORE_SESSIONS

Declaration of Prototype

```
typedef VOID
USBPUMP_CLASSKIT_OPEN_FUNCTION_FN(
          USBPUMP_SESSION_HANDLE                    SessionHandle,
          USBPUMP_CLASSKIT_OPEN_FUNCTION_CB_FN *    pCallBack,
          VOID *                                    pCallBackContext,
          USBPUMP_USBDI_FUNCTION *                  pFunction,
          USBPUMP_CLASSKIT_FUNCTION_INCALL *        pInCallApiBuffer,
          RECSIZE                                   sizeInCallApiBuffer,
          VOID *                                    pClientHandle,
          CONST USBPUMP_CLASSKIT_FUNCTION_OUTCALL *pOutCallApiBuffer,
          RECSIZE                                   sizeOutCallApiBuffer
          );
```

- `SessionHandle` is the session handle for the Class Driver object returned from `UsbPumpObject_OpenSession()`.

- `pCallBack` is the pointer to the callback routine provided by the Client to return the result of this operation.

- `pCallBackContext` is provided by the Client to be used for the callback routine.

- `pFunction` is the pointer to the function instance to a bound USB device. The pFunction can be retrieved by calling GetBoundDevices Class In-Call or from DEVICE_ARRIVAL class event notification.

- `pInCallApiBuffer` is a pointer of the buffer which will store the Function In-Call structure. The Client provides this.

- `sizeInCallApiBuffer` is the size of the Function In-Call buffer.

- `pClientHandle` is passed as a parameter to each of the asynchronous callback functions (Out-Calls). The ClassKit in any way does not interpret it. The ClassKit simply stores this and passes it when the ClassKit calls a class or function Out-Call.

- `pOutCallApiBuffer` is a pointer of the buffer which stores the function Out-Calls structure. The Client provides this. The Class Driver Framework uses the ClassKit's function Out-Call structure and it has only one Out-Call, Notification API.

- `sizeOutCallApiBuffer` is the size of the function Out-Call buffer.

Implementation

```
USBPUMP_CLASSKIT_OPEN_FUNCTION_FN
UsbPumpClassKitI_OpenFunction;
```

The ClassKit has internal routines maintaining function sessions and the implementation of this API. Class Drivers can simply use this function in their Class In-Calls.

Callback Routine

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_OPEN_FUNCTION_CB_FN,
VOID,
          (
          VOID *                          /* pCallBackContext */,
          USBPUMP_CLASSKIT_STATUS         /* ErrorCode */,
          USBPUMP_SESSION_HANDLE          /* FunctionHandle */
          ));
```

- `ErrorCode` is the result of the Open Function API.
- `FunctionHandle` is the session handle of the function session to the Class Driver function instance. This is necessary to call Function In-Calls.

3.2.3 Get the Number of Devices In-Call

Description

This function returns the number of functions currently bound to this Class Driver.

This API will return one of the following return codes through its callback routine.

- `USBPUMP_CLASSKIT_STATUS_OK`
- `USBPUMP_CLASSKIT_STATUS_INVALID_SESSION_HANDLE`

Declaration of Prototype

```
typedef VOID
USBPUMP_CLASSKIT_GET_NUM_DEVICES_FN(
          USBPUMP_SESSION_HANDLE                        SessionHandle,
          USBPUMP_CLASSKIT_GET_NUM_DEVICES_CB_FN *      pCallback,
          VOID *                                        pCallbackContext
          );
```

- `SessionHandle` is the session handle for the Class Driver object returned from `UsbPumpObject_OpenSession()`.
- `pCallBack` is the pointer to the callback routine provided by the Client to return the result of this operation.
- `pCallBackContext` is provided by the Client to be used for the callback routine.

Implementation

```
USBPUMP_CLASSKIT_GET_NUM_DEVICES_FN
```

```
UsbPumpClassKitI_GetNumDevices;
```

The ClassKit has internal routines to obtain the number of the function instances and the implementation of this API. Class Drivers can just use this function in their Class In-Calls.

Callback Routine

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_GET_NUM_DEVICES_CB_FN,
VOID,
        (
        VOID *                      /* pCallbackContext */,
        USBPUMP_CLASSKIT_STATUS     /* ErrorCode */,
        UINT32                      /* ulNumOfInstances */
        ));
```

- `ErrorCode` is the result of the Get the Number of Devices API.
- `ulNumOfInstances` is the number of the function instances bound to this Class Driver.

3.2.4 Get the Bound Devices In-Call

Description

This function is used to get the information about the functions currently bound to the Class Driver. A client provides a buffer and the buffer will be filled with bound function list information. The buffer that a client passes must not be NULL.

This API will return one of the following return codes through its callback routine.

- `USBPUMP_CLASSKIT_STATUS_OK`
- `USBPUMP_CLASSKIT_STATUS_INVALID_SESSION_HANDLE`
- `USBPUMP_CLASSKIT_STATUS_INVALID_PARAMETER`

Declaration of Prototype

```
typedef VOID
USBPUMP_CLASSKIT_GET_BOUND_DEVICES_FN(
        USBPUMP_SESSION_HANDLE                          SessionHandle,
        USBPUMP_CLASSKIT_GET_BOUND_DEVICES_CB_FN *      pCallback,
        VOID *                                          pCallbackContext,
        USBPUMP_USBDI_FUNCTION **                       pFunctionVector,
        BYTES                                           sizeFunctionVector
        );
```

- `SessionHandle` is the session handle for the Class Driver object returned from `UsbPumpObject_OpenSession()`.

- `pCallBack` is the pointer to the callback routine provided by the Client to return the result of the 'Open Function' operation.

- `pCallBackContext` is provided by the Client to be used for the callback routine.

- `pFunctionVector` is a pointer of the buffer for a vector of function instances that are bound to this Class Driver. A pointer to USBPUMP_USBDI_FUNCTION represents each function instance. This is used to open a function session to the function instance by calling OpenFunction Class In-Call. This must not be NULL.

- `sizeFunctionVector` is the size of the buffer referenced by `pFunctionVector.`

Implementation

```
USBPUMP_CLASSKIT_GET_BOUND_DEVICES_FN
UsbPumpClassKitI_GetBoundDevices;
```

The ClassKit has internal routines to retrieve the function vector and the implementation of this API. Class Drivers can just use this function in their Class In-Calls.

Callback Routine

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_GET_BOUND_DEVICES_CB_FN,
VOID,
        (
        VOID *                          /* pCallbackContext */,
        USBPUMP_CLASSKIT_STATUS         /* ErrorCode */,
        UINT32                          /* nWrittenFunction */,
        USBPUMP_USBDI_FUNCTION **       /* pFunctionVector */,
        BYTES                           /* sizeFunctionVector */
        ));
```

- `pCallBackContext` is provided by the Client to be used for the callback routine.

- `ErrorCode` is the result of the Get the Number of Devices API.

- `nWrittenFunction` is the number of copied instances information.

- `pFunctionVector` is a pointer of the buffer which contains the vector of the functions bound to this Class Driver.

- `sizeFunctionVector` is the size of the buffer referenced by `pFunctionVector .`

3.3 Class Out-Calls

3.3.1 Class Event Notification Out-Call

Description

The ClassKit delivers the class event notifications to the clients using this Out-Call. The ClassKit declares this function. But because this is an Out-Call function, a client of the Class Driver must define and implement this function and pass the pointer of this function when it calls an open session API.

Declaration of Prototype

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_NOTIFICATION_FN,
VOID,
        (
        VOID *                          /* pClientHandle */,
        USBPUMP_CLASSKIT_NOTIFICATION   /* NotificationId */,
        CONST VOID *                    /* pNotificationInfo */,
        BYTES                           /* sizeNotificationInfo */
        ));
```

- `pClientHandle` is passed as a parameter when the UsbPumpObject_OpenSession() is called.
- `NotificationId` is the identifier for the class event notification. The class events are:
  - `USBPUMP_CLASSKIT_EVENT_DEVICE_ARRIVAL` – If a client wants to open a function session for a specific function, the client has to listen for this class event notification. The client can obtain a function session by calling OpenFunction Class In-Call when this notification happens.
  - `USBPUMP_CLASSKIT_EVENT_DEVICE_DEPARTURE` – If this class event notification for the Class Driver object which your client is using happens, the client should call the Close Class In-Call to close the class session and release other resources.
  - `USBPUMP_CLASSKIT_EVENT_FUNCTION_OPEN` – When a client opens a function session, this class event notification happens.
  - `USBPUMP_CLASSKIT_EVENT_FUNCTION_CLOSE` - When a client closes a function session, this class event notification happens.
- `pNotificationInfo` contains the notification information. Each notification type has its own notification information structure. The structures are below in Table 7.
- `sizeNotificationInfo` is the size of the notification information.

Table 7 Notification Information Structure

USBPUMP_CLASSKIT_EVENT_DEVICE_ARRIVAL_INFO		
Description	This information structure is delivered with the `USBPUMP_CLASSKIT_EVENT_DEVICE_ARRIVAL`.	
Members	USBPUMP_USBDI_FUNCTION * pFunction	The pointer of Function instance. A client uses this to open a function session to the Function instance.
USBPUMP_CLASSKIT_EVENT_DEVICE_DEPARTURE_INFO		
Description	This information structure is delivered with the `USBPUMP_CLASSKIT_EVENT_DEVICE_DEPARTURE`.	
Members	USBPUMP_USBDI_FUNCTION * pFunction	The pointer of Function instance. A client uses this to close a function session which the client is using.
USBPUMP_CLASSKIT_EVENT_FUNCTION_OPEN_INFO		
Description	This information structure is delivered with the `USBPUMP_CLASSKIT_EVENT_FUNCTION_OPEN`	
Members	USBPUMP_USBDI_FUNCTION * pFunction	The pointer of Function instance.
	UINT nOpenClients	The number of clients that opened the function
USBPUMP_CLASSKIT_EVENT_DEVICE_CLOSE_INFO		
Description	This information structure is delivered with the `USBPUMP_CLASSKIT_EVENT_DEVICE_CLOSE`.	
Members	USBPUMP_USBDI_FUNCTION * pFunction	The pointer of Function instance.
	UINT nOpenClients	The number of clients that closed the function

Implementation

A client of the Class Driver should define and implement this function.

3.4 Function In-Calls

3.4.1 Close Function In-Call

Description

This function closes the function session opened at open function Class In-Call. If a client doesn't want to use the session any longer, the client should call this Function In-Call API to de-allocate the session resource.

This API will return one of the following return codes through its callback routine.

- `USBPUMP_CLASSKIT_STATUS_OK`
- `USBPUMP_CLASSKIT_STATUS_INVALID_FUNCTION_HANDLE`
- `USBPUMP_CLASSKIT_STATUS_FUNCTION_NOT_OPENED`

Declaration of Prototype

This API uses the same prototype with the close session API in the section 3.2.1 in this document.

Implementation

```
__TMS_USBPUMP_API_CLOSE_FN
UsbPumpClassKitI_CloseFunction;
```

The ClassKit has routines maintaining the function sessions and the implementation of this API. The developer who develops a new Class Driver using the ClassKit simply sets the pointer of `UsbPumpClassKitI_CloseFunction()` in the Function In-Calls of the Class Driver. The Class Driver Framework sets this function pointer in its Class In-Calls.

3.5 Function Out-Calls

3.5.1 Function Event Notification Out-Call

Description

The ClassKit delivers the function event notifications to the clients using this API. The ClassKit declares this function. But because this is an Out-Call function, a client of the Class Driver must define and implement this function and pass the pointer of this function when it calls an open session API.

Declaration of Prototype

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_NOTIFICATION_FN,
VOID,
          (
```

```
VOID *                          /* pClientHandle */,
USBPUMP_CLASSKIT_NOTIFICATION   /* NotificationId */,
CONST __TMS_VOID *              /* pNotificationInfo */,
BYTES                           /* sizeNotificationInfo */
));
```

- `pClientHandle` is passed as a parameter when the open function is called.
- `NotificationId` is the identifier for the function event notification. The function events are
  - `USBPUMP_CLASSKIT_EVENT_DEVICE_ARRIVAL` – If a client is interested in the other functions arrival, the client should listen this function notification.
  - `USBPUMP_CLASSKIT_EVENT_DEVICE_DEPARTURE` - If this function notification for the function which your client is using happens, the client should call the Close Function In-Call to close the function session and release other resource.
  - `USBPUMP_CLASSKIT_EVENT_FUNCTION_OPEN` – When a client opens a function session, this function notification happens.
  - `USBPUMP_CLASSKIT_EVENT_FUNCTION_CLOSE` - When a client closes a function session, this function notification happens.
- `pNotificationInfo` contains the notification information. Each notification type has its own notification information structure. For the detail of the structures, refer to the Table 6 in this document.
- `sizeNotificationInfo` is the size of the notification information.

Implementation

A client of the Class Driver should define and implement this function.

3.6 ClassKit Class API

These are the internal operations that the Class Driver uses to communicate with the ClassKit. The client of the Class Driver doesn't use these operations directly.

3.6.1 UsbPumpClassKitI_InitDriverClass_V1

Description

This function initializes the ClassKit portion of a Class Driver object that is based on the ClassKit. This API returns TRUE if success, otherwise FALSE.

Declaration of Prototype

```
BOOL
UsbPumpClassKitI_InitDriverClass_V1(
        USBPUMP_CLASSKIT_CLASS *                    pDriverClass,
        CONST USBGUID *                             pGuid,
        USBPUMP_CLASSKIT_INIT_FUNCTION_INCALL_FN *  pInitFunctionIncallFn,
        UINT                                        MaxSession,
        BYTES                                       sizeConfigTree
        )
```

- `pDriverClass` is a Class Driver object that is created at the Class Driver initialization routine.
- `pGuid` is the GUID of the Class Driver.
- `pInitFunctionIncallFn` is the out-switch function which initializes the Function In-Call structure. This function is called by the ClassKit when a client calls the open function Class In-Call to exchange function In-Calls and Out-Calls. Refer to section 3.10.1 for the details of this out-switch.
- `MaxSession` is the maximum session that the ClassKit maintains for the class and function sessions.
- `sizeConfigTree` is the maximum size of the configuration tree. The configuration tree consists of the configuration nodes, interface nodes, alternative setting nodes, and pipe nodes. This value varies depending on the configuration bundle that the Class Driver supports.

3.6.2 UsbPumpClassKitI_OpenSession

Description

This function opens a class session when the Class Driver receives USBPUMP_IOCTL_API_OPEN I/O control. If a client calls the UsbPumpObject_OpenSession() with the GUID of the Class Driver, the USBPUMP_IOCTL_API_OPEN I/O control is delivered to the Class Driver's I/O control handling routine. Because the ClassKit maintains the sessions, the Class Driver calls this API to open a session and return the session handle to the client.

This API will return one of the following return codes.

- `USBPUMP_IOCTL_RESULT_SUCCESS`
- `USBPUMP_IOCTL_RESULT_FAILED`
- `USBPUMP_IOCTL_RESULT_NOT_CLAIMED`

Declaration of Prototype

```
USBPUMP_IOCTL_RESULT
UsbPumpClassKitI_OpenSession(
```

```
        USBPUMP_CLASSKIT_CLASS *     pDriverClass,
        CONST VOID *                 pInParam,
        VOID *                       pOutParam,
        CONST VOID *                 pvInCalls,
        RECSIZE                      sizeInCalls
        )
```

- `pDriverClass` is a Class Driver object that is created in the Class Driver initialization routine.
- `pInParam` is a pointer to the input parameters of the USBPUMP_IOCTL_API_OPEN I/O control. Refer to the [MOB] for USBPUMP_IOCTL_API_OPEN_ARG.
- `pOutParam` is a pointer to the output parameters of the USBPUMP_IOCTL_API_OPEN I/O control. Refer to the [MOB] for USBPUMP_IOCTL_API_OPEN_ARG.
- `pvInCalls` is a pointer of the Class In-Call buffer of the Class Driver. The content referenced by the pointer will be copied to the pointer of the buffer in `pOutParam` if handling the I/O control is successful.
- `sizeInCalls` is the size of the buffer referenced by `pvInCalls`.

3.6.3 UsbPumpClassKitI_SendNotification

Description

This function sends a driver class event notification to clients.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_SendNotification(
        USBPUMP_CLASSKIT_CLASS *         pDriverClass,
        USBPUMP_CLASSKIT_NOTIFICATION    NotificationId,
        CONST VOID *                     pNotificationInfo,
        BYTES                            sizeNotificationInfo
        )
```

- `pDriverClass` is a Class Driver object that the Class Driver initialization routine created.
- `NotificationId` is the notification identifier.
- `pNotificationInfo` is a pointer of the notification information.
- `sizeNotificationInfo` is the size of the buffer referenced by `pNotificationInfo`.

3.7 ClassKit Function API

3.7.1 UsbPumpClassKitI_InitFunction_V1

Description

This function initializes the ClassKit portion of a function instance object. This will be invoked when the Class Driver handles USBPUMP_IOCTL_USBDI_FUNCTION_INIT I/O control.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_InitFunction_V1(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction,
         USBPUMP_USBDI_DRIVER_CLASS * pDriverClass,
         BOOL                         fAllowMultipleOpen
         )
```

- `pFunction` is the function instance that will be initialized here.
- `pDriverClass` is the Class Driver object that the function instance binds to.
- `fAllowMultipleOpen` is the flag that allows opening multiple function sessions to one function instance. Refer to open function API (section 3.2.2).

3.7.2 UsbPumpClassKitI_DeinitFunction

Description

This function de-initializes the ClassKit portion of a function instance object. This will be invoked when the Class Driver processes USBPUMP_IOCTL_USBDI_FUNCTION_DEINIT I/O control.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_DeinitFunction(
         USBPUMP_CLASSKIT_FUNCTION *      pFunction,
         USBPUMP_USBDI_DRIVER_CLASS *     pDriverClass
         )
```

- `pFunction` is a pointer of the function instance that will be de-initialized here.
- `pDriverClass` is a pointer of the Class Driver object that the function instance binds to.

3.7.3 UsbPumpClassKitI_InstanceArrival_V1

Description

This function gets control after a function instance has been bound to a specific port. This routine allocates an URB to be used by the ClassKit FSM to enumerate the device and memory for the configuration tree. It then starts the ClassKit FSM. Your Class Driver must call this function when the Class Driver receives USBPUMP_IOCTL_EDGE_USBDI_INSTANCE_ARRIVAL I/O control.

This API will return one of the following return codes.

- `USBPUMP_IOCTL_RESULT_SUCCESS`
- `USBPUMP_IOCTL_RESULT_IMPLEMENTATION_LIMITATION`

The second return code is returned when allocating memory for the URB and the configuration tree fails.

Declaration of Prototype

```
USBPUMP_IOCTL_RESULT
UsbPumpClassKitI_InstanceArrival_V1(
          USBPUMP_CLASSKIT_FUNCTION *              pFunction,
          USBPUMP_CLASSKIT_FSM_PROCESS_FN *        pUnconfiguredProcessFn,
          USBPUMP_CLASSKIT_FSM_PROCESS_FN *        pConfiguredProcessFn,
          USBPUMP_CLASSKIT_FSM_STATE_CHANGE_FN *   pStateChangeFn
          )
```

- `pFunction` is a pointer of the function instance that is bound to a port by USBD.

- `pUnconfiguredProcessFn` is a pointer of the out-switch routine that will be invoked when the ClassKit FSM reaches the unconfigured state. Refer to section 3.10.2 for the details of this out-switch.

- `pConfiguredProcessFn` is a pointer of the out-switch routine that will be invoked when the ClassKit FSM reaches the configured state. Refer to section 3.10.3 for the details of this out-switch.

- `pStateChangeFn` is a pointer of the out-switch routine that will be invoked whenever the ClassKit FSM transitions from one state to another state. Refer to section 3.10.4 for the detail of this out-switch.

3.7.4 UsbPumpClassKitI_InstanceDeparture

Description

This function gets control after an unplug of the device which is bound to the function instance is discovered. This function shuts down the ClassKit FSM if it's still running. The Driver must

call this function when the Class Driver receives USBPUMP_IOCTL_EDGE_USBDI_INSTANCE_DEPARTURE_ASYNC I/O control.

This API will always return `USBPUMP_IOCTL_RESULT_SUCCESS.`

Declaration of Prototype

```
USBPUMP_IOCTL_RESULT
UsbPumpClassKitI_InstanceDeparture(
         USBPUMP_CLASSKIT_FUNCTION *      pFunction,
         USBPUMP_IOCTL_QE *               pIoctlQe
         )
```

- `pFunction` is a pointer of the function instance that is bound to a port by USBD.
- `pIoctlQe` is a pointer of the queue element for the asynchronous I/O control.

3.7.5 UsbPumpClassKitI_SendFunctionNotification

Description

This routine sends a function notification to clients opening the function.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_SendFunctionNotification(
         USBPUMP_CLASSKIT_FUNCTION *      pFunction,
         USBPUMP_CLASSKIT_NOTIFICATION    NotificationId,
         CONST VOID *                     pNotificationInfo,
         BYTES                            sizeNotificationInfo
         )
```

- `pFunction` is a pointer of the Class Driver function instance that the notification is related to.
- `NotificationId` is the notification identifier.
- `pNotificationInfo` is a pointer of the notification information.
- `sizeNotificationInfo` is the size of the buffer referenced by `pNotificationInfo`.

3.8 ClassKit FSM API

3.8.1 UsbPumpClassKitI_ClassKitFsm_RequestExit

Description

This function requests that the ClassKit FSM shut down.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_ClassKitFsm_RequestExit(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction
         )
```

- `pFunction` is a pointer of the Class Driver function instance.

3.8.2 UsbPumpClassKitI_ClassKitFsm_SetEnumerated

Description

This function sets the fEnumerated flag to alert the FMS that it's time to send a notification USBPUMP_CLASSKIT_EVENT_DEVICE_ARRIVAL to clients.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_ClassKitFsm_SetEnumerated(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction
         )
```

- `pFunction` is a Class Driver function instance that is successfully enumerated.

3.8.3 UsbPumpClassKitI_ClassKitFsm_SetConfig0

Description

This function sets the fDoSetConfig0 flag to alert the FSM that it is time to do SetConfig(0).

Declaration of Prototype

```
VOID
UsbPumpClassKitI_ClassKitFsm_SetConfig0(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction
         )
```

- `pFunction` is a pointer of the Class Driver function instance.

3.8.4 UsbPumpClassKitI_ClassKitFsm_SetConfigN

Description

This function sets the fDoSetConfigN flag to alert the FSM that it is time to do SetConfig(N).

Declaration of Prototype

```
VOID
UsbPumpClassKitI_ClassKitFsm_SetConfigN(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction,
         UINT8                        iConfig
         )
```

- `pFunction` is a Class Driver function instance that is successfully enumerated.
- iConfig is a configuration index. This doesn't mean the bConfigurationValue of the USB configuration descriptor. For example, if a USB device has three configurations and bConfigurationValue is 1, 3, and 6 for each configuration, iConfig(0) means the first configuration (bConfigurationValue is 1), iConfig(1) means the second configuration (bConfigurationValue is 3), and iConfig(2) means the last configuration (bConfigurationValue is 6).

3.8.5 UsbPumpClassKitI_ClassKitFsm_UnconfiguredStateDone

Description

This function notifies the completion flag of stUnconfigured state and alert the FSM.

Declaration of Prototype

```
VOID
UsbPumpClassKitI_ClassKitFsm_UnconfiguredStateDone(
         USBPUMP_CLASSKIT_FUNCTION *  pFunction,
         BOOL                         fCompleteOk
         )
```

- `pFunction` is a Class Driver function instance that is successfully enumerated.
- `fCompleteOk` is a flag which indicates whether the process of unconfigured state of Class Driver was successful or not.

3.9 ClassKit Session API

3.9.1 UsbPumpClassKitI_ValidateSessionHandle

Description

This function validates a class or a function session that is passed to a class or a Function In-Call. If the specified session is valid, it will return the object header of the session. Otherwise, it will return NULL.

Declaration of Prototype

```
VOID *
UsbPumpClassKitI_ValidateSessionHandle(
         USBPUMP_SESSION_HANDLE   SessionHandle,
         BOOL                     fClassHandle
         )
```

- SessionHandle is a specific session handle that will be validated. This can be a class or a function session handle.

- `fClassHandle` – If this is TRUE, the SessionHandle is the session handle to a Class Driver object. Otherwise, it's the session handle to a Function instance.

Example for Class In-Call

```
VOID
UsbPumpUsbdSampleI_ClassXXX(
         USBPUMP_SESSION_HANDLE                    SessionHandle,
         USBPUMP_USBDI_SAMPLE_CLASS_XXX_CB_FN *    pCallBack,
         VOID *                                    pCallBackContext
         )
         {
         USBPUMP_USBDI_CLASS_SAMPLE * pDriverClass;

         if (pCallBack == NULL)
             {
             /* Nothing to do withOut-Callback */
             return;
             }

         pDriverClass = UsbPumpClassKitI_ValidateSessionHandle(
                 SessionHandle,
                 TRUE
                 );
         if (pDriverClass == NULL)
             {
             (*pCallBack)(
                 pCallBackContext,
                 USBPUMP_CLASSKIT_STATUS_INVALID_SESSION_HANDLE
                 );
             }

         /* XXX: put more code here */

         (*pCallBack)(
             pCallBackContext,
             USBPUMP_CLASSKIT_STATUS_OK
             );
         }
```

Example for Function In-Call

```
VOID
UsbPumpUsbdSampleI_FunctionXXX(
          USBPUMP_SESSION_HANDLE           FunctionHandle,
          USBPUMP_USBDI_SAMPLE_FUNCTION_XXX_CB_FN *pCallBack,
          VOID *                   pCallBackContext
          )
          {
          USBPUMP_USBDI_FUNCTION_SAMPLE *  pFunction;

          if (pCallBack == NULL)
              {
              /* Nothing to do withOut-Callback */
              return;
              }

          pFunction = UsbPumpClassKitI_ValidateSessionHandle(
                  FunctionHandle,
                  FALSE
                  );
          if (pFunction == NULL)
              {
              (*pCallBack)(
                  pCallBackContext,
                  USBPUMP_CLASSKIT_STATUS_INVALID_FUNCTION_HANDLE
                  );
              }

          /* XXX: put more code here */

          (*pCallBack)(
              pCallBackContext,
              USBPUMP_CLASSKIT_STATUS_OK
              );
          }
```

3.10 ClassKit Out-Switch

3.10.1 Initialize Function In-Call Buffer Function

Description

This function initializes the Function In-Call buffer of the Class Driver. This function is passed when the Class Driver initializes the ClassKit for the Class Driver object. When a client opens a function session using the open function API, the ClassKit calls this function to fill the Class

Driver's Function In-Call buffer into the buffer the client provides. A developer of a Class Driver has to define this function and implement this.

Declaration of Prototype

```
__TMS_FNTYPE_DEF(
USBPUMP_CLASSKIT_INIT_FUNCTION_INCALL_FN,
         BOOL,
             (
             USBPUMP_CLASSKIT_CLASS *         /* pDriverClass */,
             USBPUMP_CLASSKIT_FUNCTION *      /* pFunction */,
             VOID *                           /* pInCallBuffer */,
             RECSIZE                          /* sizeInCallBuffer */
             ));
```

- `pDriverClass` is a pointer of the Class Driver object that the function instance is bound to.
- `pFunction` is a pointer of the function instance that a client wants to open.
- `pInCallBuffer` is a pointer of the empty buffer that a client provides to get a copy of the Class Driver's In-Call buffer.
- `sizeInCallBuffer` is the size of the buffer referenced by `pInCallBuffer` .

3.10.2 Unconfigured State Processing Function

Description

This function processes the Class Driver specific routines when the ClassKit FSM is in the unconfigured state(stUnconfigured). The Class Driver can start a sub-FSM, if needed. As the Class Driver Framework provides a basic sub-FSM, you can easily implement a more complicated sub-FSM. Refer to the section 2.5 for the details of the sub-FSM.

This is optional, so the Class Driver can set NULL for this if there is nothing to process in this state.

If you provide this function, you must call one of following functions just before finishing the unconfigured state processing to let the ClassKit FSM keep running. You can find how the functions work at Figure 1 State Diagram of ClassKit FSM in this document.

- `UsbPumpClassKitI_ClassKitFsm_UnconfiguredStateDone(USBPUMP_CLASSKIT_FUNCTION *pFunction, BOOL fCompleteOK)` - If this function is called with fCompleteOK TRUE, the specific Class Driver's portion of handling the unconfigured state is successful and the ClassKit FSM will transition to the setting configuration state(stSettingConfig) to set the USB device's configuration to non-zero and valid configuration value(usually 1). If this function is called with fCompleteOK FALSE, it means that an erroneous situation happens in the out-switch function. This will allow

the ClassKit FSM to transition to the stErrorUndoCfg state to de-initialize the defined configuration. For the detail of this function, refer to section 3.8.5.

- `UsbPumpClassKitI_ClassKitFsm_SetConfig0(USBPUMP_CLASSKIT_FUNCTION *pFunction)` – If this function is called, the ClassKit will set the USB device's configuration to zero and make the ClassKit FSM's state the unconfigured state. For the detail of this function, refer to section 3.8.3.

- `UsbPumpClassKitI_ClassKitFsm_SetConfigN(USBPUMP_CLASSKIT_FUNCTION *pFunction, UINT8 iConfig)` – If this function is called, the ClassKit FSM will set the USB device's configuration to a specific configuration value which is non-zero and valid. For the detail of this function, refer to section 3.8.4.

Declaration of Prototype

```
__TMS_TYPE_DEF_FUNCTION(
USBPUMP_CLASSKIT_FSM_PROCESS_FN,
         VOID,
             (
             USBPUMP_CLASSKIT_FUNCTION *  /* pFunction */,
             BOOL                         /* fEntry */
             ));
```

- `pFunction` is a pointer of the function instance that the FSM is working on.

- `fEntry` is the flag that indicates if this function is called at entry of the state or not.

3.10.3 Configured State Processing Function

Description

This function processes the Class Driver specific things when the ClassKit FSM's state is the configured state(stConfigured). The Class Driver can start a sub-FSM, if needed. As the Class Driver Framework provides a basic sub-FSM, you can easily implement a more complicated sub-FSM. Refer to the section 2.5 for the details of the sub-FSM.

This is not optional. The minimum behavior of this Out-Call is to set fEnumerated flag by calling UsbPumpClassKitI_ClassKitFsm_SetEnumerated() to make the ClassKit send a DEVICE_ARRIVAL notification to clients.

You must call one of following functions in the implementation of this function.

- `UsbPumpClassKitI_ClassKitFsm_SetEnumerated(USBPUMP_CLASSKIT_FUNCTION *pFunction)` – If this function is called, the specific Class Driver's portion of handling the configured state is successful and the ClassKit FSM will send a DEVICE_ARRIVAL notification to clients.

- `UsbPumpClassKitI_ClassKitFsm_SetConfig0(USBPUMP_CLASSKIT_FUNCTION *pFunction` – If this function is called, the ClassKit will set the USB device's configuration to zero and make the ClassKit FSM's state the unconfigured state.

- `UsbPumpClassKitI_ClassKitFsm_SetConfigN(USBPUMP_CLASSKIT_FUNCTION *pFunction, UINT8 iConfig)` – If this function is called, the ClassKit FSM will set the USB device's configuration to a specific configuration value which is non-zero and valid.

- `UsbPumpClassKitI_ClassKitFsm_RequestExit(USBPUMP_CLASSKIT_FUNCTION *pFunction)` – If this function is called, that means an unplug of the USB device was discovered at the Class Driver's sub-FSM. This will make the ClassKit FSM transition to the stErrorUndoCfg state to de-initialize the defined configuration.

Declaration of Prototype

This Out-Call's prototype is the same as the unconfigured state handling function in the section 3.10.2 in this document.

3.10.4 State Change Processing Function

Description

If this function is provided when initializing the ClassKit Class Driver, this function will be invoked whenever the state of the ClassKit FSM changes. This is optional, so you can pass NULL for this function.

Declaration of Prototype

```
__TMS_TYPE_DEF_FUNCTION(
USBPUMP_CLASSKIT_FSM_STATE_CHANGE_FN,
         VOID,
             (
             USBPUMP_CLASSKIT_FUNCTION *      /* pFunction */,
             USBPUMP_CLASSKIT_FSM_STATE       /* PreviousState */,
             USBPUMP_CLASSKIT_FSM_STATE       /* CurrentState */
             ));
```

- `pFunction` is a pointer of the function instance that the ClassKit FSM is working on.

- `PreviousState` is the state that the ClassKit FSM just resided before the current state.

- `CurrentState` is the state that the ClassKit FSM resides now.