```
/* pkcs11.h include file for PKCS #11. */

/* Copyright © OASIS Open 2013. All Rights Reserved.*/
/* PKCS #11 Version 2.40 */
/* Working Draft 01 */

/*
This document and translations of it may be copied and
furnished to others, and derivative works that comment on or
otherwise explain it or assist in its implementation may be
prepared, copied, published, and distributed, in whole or in
part, without restriction of any kind, provided that the
above copyright notice and this section are included on all
such copies and derivative works. However, this document
itself may not be modified in any way, including by removing
the copyright notice or references to OASIS, except as needed
for the purpose of developing any document or deliverable
produced by an OASIS Technical Committee (in which case the
rules applicable to copyrights, as set forth in the OASIS IPR
Policy, must be followed) or as required to translate it into
languages other than English.

The limited permissions granted above are perpetual and will
not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is
provided on an "AS IS" basis and OASIS DISCLAIMS ALL
WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT
INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
*/

#ifndef _PKCS11_H_
#define _PKCS11_H_ 1

#ifdef __cplusplus
extern "C" {
#endif

/* Before including this file (pkcs11.h) (or pkcs11t.h by
 * itself), 6 platform-specific macros must be defined.
These
 * macros are described below, and typical definitions for
them
 * are also given.  Be advised that these definitions can
depend
 * on both the platform and the compiler used (and possibly
also
 * on whether a Cryptoki library is linked statically or
 * dynamically).
 *
 * In addition to defining these 6 macros, the packing
convention
 * for Cryptoki structures should be set.  The Cryptoki
 * convention on packing is that structures should be 1-byte
 * aligned.
 *
```

```
 * If you're using Microsoft Developer Studio 5.0 to produce
 * Win32 stuff, this might be done by using the following
 * preprocessor directive before including pkcs11.h or
pkcs11t.h:
 *
 * #pragma pack(push, cryptoki, 1)
 *
 * and using the following preprocessor directive after
including
 * pkcs11.h or pkcs11t.h:
 *
 * #pragma pack(pop, cryptoki)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to produce Win16 stuff, this might be done by using
 * the following preprocessor directive before including
 * pkcs11.h or pkcs11t.h:
 *
 * #pragma pack(1)
 *
 * In a UNIX environment, you're on your own for this.  You
might
 * not need to do (or be able to do!) anything.
 *
 *
 * Now for the macros:
 *
 *
 * 1. CK_PTR: The indirection string for making a pointer to
an
 * object.  It can be used like this:
 *
 * typedef CK_BYTE CK_PTR CK_BYTE_PTR;
 *
 * If you're using Microsoft Developer Studio 5.0 to produce
 * Win32 stuff, it might be defined by:
 *
 * #define CK_PTR *
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to produce Win16 stuff, it might be defined by:
 *
 * #define CK_PTR far *
 *
 * In a typical UNIX environment, it might be defined by:
 *
 * #define CK_PTR *
 *
 *
 * 2. CK_DEFINE_FUNCTION(returnType, name): A macro which
makes
 * an exportable Cryptoki library function definition out of
a
 * return type and a function name.  It should be used in the
 * following fashion to define the exposed Cryptoki functions
in
 * a Cryptoki library:
 *
```

```
 * CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
 *   CK_VOID_PTR pReserved
 * )
 * {
 *   ...
 * }
 *
 * If you're using Microsoft Developer Studio 5.0 to define a
 * function in a Win32 Cryptoki .dll, it might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType __declspec(dllexport) name
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to define a function in a Win16 Cryptoki .dll, it
 * might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType __export _far _pascal name
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DEFINE_FUNCTION(returnType, name) \
 *   returnType name
 *
 *
 * 3. CK_DECLARE_FUNCTION(returnType, name): A macro which
makes
 * an importable Cryptoki library function declaration out of
a
 * return type and a function name.  It should be used in the
 * following fashion:
 *
 * extern CK_DECLARE_FUNCTION(CK_RV, C_Initialize)(
 *   CK_VOID_PTR pReserved
 * );
 *
 * If you're using Microsoft Developer Studio 5.0 to declare
a
 * function in a Win32 Cryptoki .dll, it might be defined by:
 *
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType __declspec(dllimport) name
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to declare a function in a Win16 Cryptoki .dll, it
 * might be defined by:
 *
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType __export _far _pascal name
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DECLARE_FUNCTION(returnType, name) \
 *   returnType name
 *
 *
 * 4. CK_DECLARE_FUNCTION_POINTER(returnType, name): A macro
```

```
 * which makes a Cryptoki API function pointer declaration or
 * function pointer type declaration out of a return type and
a
 * function name.  It should be used in the following
fashion:
 *
 * // Define funcPtr to be a pointer to a Cryptoki API
function
 * // taking arguments args and returning CK_RV.
 * CK_DECLARE_FUNCTION_POINTER(CK_RV, funcPtr)(args);
 *
 * or
 *
 * // Define funcPtrType to be the type of a pointer to a
 * // Cryptoki API function taking arguments args and
returning
 * // CK_RV, and then define funcPtr to be a variable of type
 * // funcPtrType.
 * typedef CK_DECLARE_FUNCTION_POINTER(CK_RV, funcPtrType)
(args);
 * funcPtrType funcPtr;
 *
 * If you're using Microsoft Developer Studio 5.0 to access
 * functions in a Win32 Cryptoki .dll, in might be defined
by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType __declspec(dllimport) (* name)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to access functions in a Win16 Cryptoki .dll, it
might
 * be defined by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType __export _far _pascal (* name)
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
 *   returnType (* name)
 *
 *
 * 5. CK_CALLBACK_FUNCTION(returnType, name): A macro which
makes
 * a function pointer type for an application callback out of
 * a return type for the callback and a name for the
callback.
 * It should be used in the following fashion:
 *
 * CK_CALLBACK_FUNCTION(CK_RV, myCallback)(args);
 *
 * to declare a function pointer, myCallback, to a callback
 * which takes arguments args and returns a CK_RV.  It can
also
 * be used like this:
 *
 * typedef CK_CALLBACK_FUNCTION(CK_RV, myCallbackType)(args);
```

```
 * myCallbackType myCallback;
 *
 * If you're using Microsoft Developer Studio 5.0 to do Win32
 * Cryptoki development, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType (* name)
 *
 * If you're using an earlier version of Microsoft Developer
 * Studio to do Win16 development, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType _far _pascal (* name)
 *
 * In a UNIX environment, it might be defined by:
 *
 * #define CK_CALLBACK_FUNCTION(returnType, name) \
 *   returnType (* name)
 *
 *
 * 6. NULL_PTR: This macro is the value of a NULL pointer.
 *
 * In any ANSI/ISO C environment (and in many others as
well),
 * this should best be defined by
 *
 * #ifndef NULL_PTR
 * #define NULL_PTR 0
 * #endif
 */


/* All the various Cryptoki types and #define'd values are in
the
 * file pkcs11t.h. */
#include "pkcs11t.h"

#define __PASTE(x,y)      x##y


/*
==============================================================
=
 * Define the "extern" form of all the entry points.
 *
==============================================================
=
 */

#define CK_NEED_ARG_LIST  1
#define CK_PKCS11_FUNCTION_INFO(name) \
  extern CK_DECLARE_FUNCTION(CK_RV, name)

/* pkcs11f.h has all the information about the Cryptoki
 * function prototypes. */
#include "pkcs11f.h"

#undef CK_NEED_ARG_LIST
```

```
#undef CK_PKCS11_FUNCTION_INFO


/*
==============================================================
=
 * Define the typedef form of all the entry points.  That is,
for
 * each Cryptoki function C_XXX, define a type CK_C_XXX which
is
 * a pointer to that kind of function.
 *
==============================================================
=
 */

#define CK_NEED_ARG_LIST  1
#define CK_PKCS11_FUNCTION_INFO(name) \
  typedef CK_DECLARE_FUNCTION_POINTER(CK_RV, __PASTE
(CK_,name))

/* pkcs11f.h has all the information about the Cryptoki
 * function prototypes. */
#include "pkcs11f.h"

#undef CK_NEED_ARG_LIST
#undef CK_PKCS11_FUNCTION_INFO


/*
==============================================================
=
 * Define structed vector of entry points.  A
CK_FUNCTION_LIST
 * contains a CK_VERSION indicating a library's Cryptoki
version
 * and then a whole slew of function pointers to the routines
in
 * the library.  This type was declared, but not defined, in
 * pkcs11t.h.
 *
==============================================================
=
 */

#define CK_PKCS11_FUNCTION_INFO(name) \
  __PASTE(CK_,name) name;

struct CK_FUNCTION_LIST {

  CK_VERSION    version;  /* Cryptoki version */

/* Pile all the function pointers into the CK_FUNCTION_LIST.
*/
/* pkcs11f.h has all the information about the Cryptoki
 * function prototypes. */
#include "pkcs11f.h"
```

```
};

#undef CK_PKCS11_FUNCTION_INFO


#undef __PASTE

#ifdef __cplusplus
}
#endif

#endif
```