Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Ministère du budget, des comptes publics et de la fonction publique

Direction Générale de la Modernisation de l'Etat

Middleware IAS

API IAS
Guide de programmation

AVEC VOUS l'administration SE MODERNISE

Middleware IAS	
IAS API	
Référence	**Date**
MDWIAS_SF_IASAPIF_v1 09_MS.doc	27/09/07
Identification d'objet (OID)	**Racine OID et gestionnaire**
1.2.250.1.137.2.3.2.2.1	SDAE
Responsable	**Version**
DGME/SDAE	V1.09
Critère de diffusion	**Nombre de pages**
Public	53

HISTORIQUE DES VERSIONS			
DATE	**VERSION**	**EVOLUTION DU DOCUMENT**	**REDACTEUR**
26/09/06	0.8	Mise à jour	Dictao
28/09/06	0.9	Corrections	Dictao
29/09/06	1.0	Validation	Dictao
05/10/06	1.01	Ajout de la description de l'API	Dictao
06/10/06	1.02	Modification après commentaires de la DGME	Dictao
10/10/06	1.03	Modification après commentaires de la DGME	Dictao
08/11/2006	1.04	Modification après comité de relecture	Dictao
15/11/06	1.05	Modification après comité projet	Dictao
27/11/06	1.06	Modification après comité de relecture du 24/11/06	Dictao
02/02/07	1.07	Modification suite à l'appel à commentaire	Dictao
02/07/07	1.08	Clarifications mode transparent et support lecteurs « pinpad »	Dictao
27/09/07	1.09	Modification après commentaires DGME	Dictao

Sommaire

1 - Introduction

Les applications souhaitant employer des services cryptographiques de haut niveau ont le choix entre deux interfaces standards **[PKCS #11]** et **[CryptoAPI]**. Ces interfaces sont très stables et très utilisées par la plupart des logiciels, car elles fournissent un accès à n'importe quel genre de dispositif cryptographique, du simple certificat enregistré sur le disque dur d'un utilisateur à la dernière génération de cartes à empreinte digitale.
Toutefois, ce niveau d'abstraction, bien qu'adapté pour toutes les utilisations directes, peut être une limitation quand les applications doivent avoir accès aux fonctions avancées ou aux détails d'implémentation d'un matériel spécifique.
Par exemple, des transactions par « secure messaging », des services de personnalisation, sont impossibles au travers de ces APIs (Application Programming Interface = Interface de programmation).Dans ce dernier cas, le mode transparent est l'unique solution envisageable.

1.1 - But du document

Ce document spécifie les fonctions et les classes nécessaires pour construire une représentation objet **[ISO 7816-15]** basée sur le contenu de la carte en conformité avec la spécification **[IAS]**. De plus, cette API fournit un canal transparent pour effectuer des tâches administratives en « secure messaging », comme par exemple gérer des informations secrètes depuis un serveur distant ou depuis son bureau personnel.

Cette API implémente donc deux types de fonctionnalités qui sont décrites en détail dans le document.

Il est à noter que les applications classiques doivent utiliser les interfaces **[PKCS #11]** ou **[CryptoAPI]** afin d'avoir accès aux services cryptographiques, tandis que les applications désirant vérifier/afficher la structure de la carte ou administrer son contenu doivent employer cette API IAS spécifique, ainsi que les applications effectuant de la signature qualifiée. Les applications utilisant l'API IAS peuvent réaliser aussi les mêmes fonctions que celles accessibles par les interfaces **[PKCS #11]** ou **[CryptoAPI]**. Ceci leur permet de ne gérer qu'un seul contexte en n'appelant qu'un seul module.

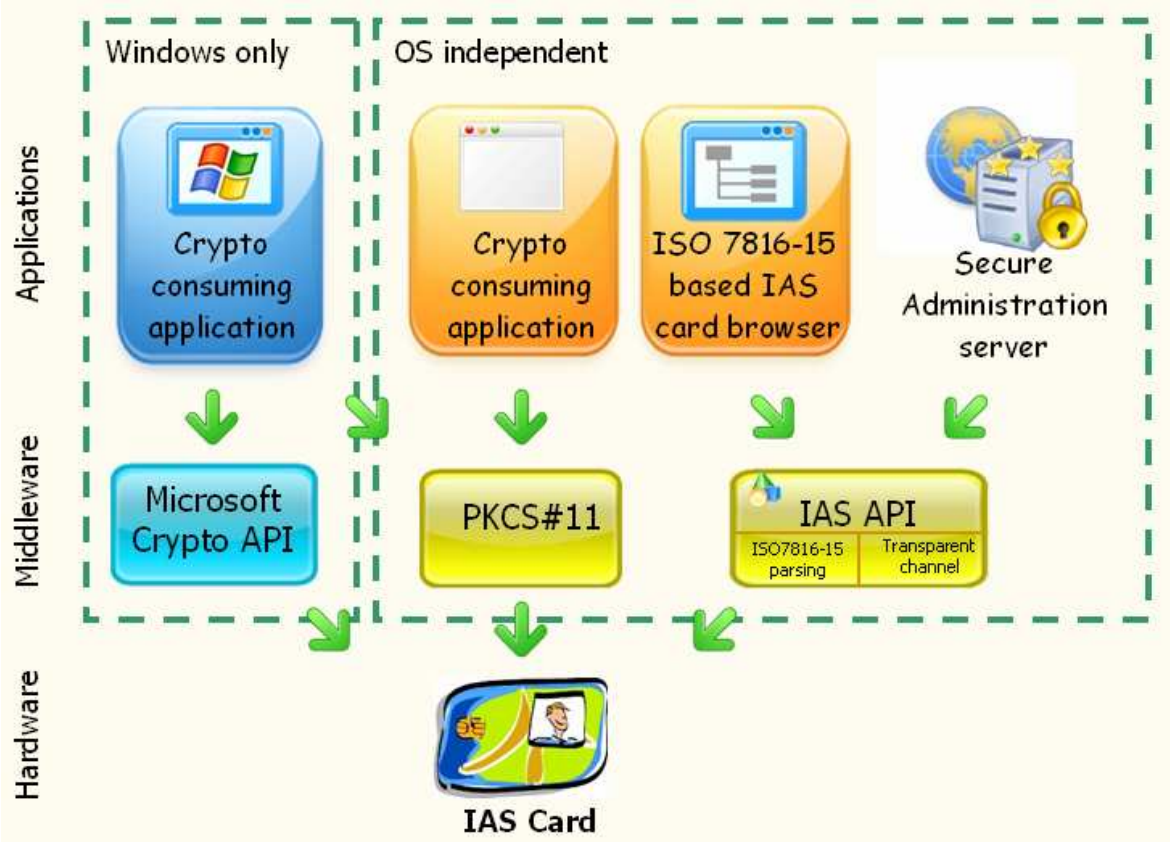


Figure 1 : Les différentes APIs d'accès à la carte IAS

1.2 - Description de l'API IAS

L'API IAS fonctionne selon deux modes différents pour les divers types de données de la carte IAS, certaines classes permettent de gérer les objets de type PKCS#15 (associés aux objets décrits en section 6.2 et 6.3), d'autres vont servir à administrer la carte par canal transparent (associées aux objets définis en section 6.1).

L'API IAS est construite de la façon suivante :

1. Les types de données ont été définis afin d'être manipulés directement dans le code des applications appelant l'API. La plupart des membres des classes sont de type de l'un de ceux définis.

2. Les classes de base donnent des informations d'ordre général directement liées à la carte comme par exemple quels sont les contrôles d'accès à la carte ou des informations sur les clés de la carte. Certaines de ces classes sont très utilisées par les autres classes de l'API.

3. Les fonctions définies dans la section 4 - sont des fonctions relatives à l'API (elles vont permettre par exemple d'initialiser l'API). Certaines spécifiques gèrent les accès à la carte.

4. Les objets sont des représentations des données de la carte telles que les fichiers ou les objets cryptographiques (PKCS#15).

1.3 - Les différents usages de l'API IAS

Quel que soit le déroulement des actions sur la carte, la première étape doit être la connexion à la carte, les fonctions de la section 4 permettent d'établir cette connexion.

Le code suivant expose un exemple de la procédure à suivre pour se connecter à une carte IAS :

```
IASRESULT result;

// Initialisation de l'API IAS
IASCONTEXT iasContext;
result = IASInitialize(&iasContext);

// Récupération de la liste des lecteurs
char mszReaders[1000];
DWORD cchReaders=sizeof(mszReaders);
result = IASListReaders(iasContext, mszReaders,&cchReaders);

// Tentative de connexion à la carte présente dans le 1er lecteur
CIASCard *pIASCard;
result= IASConnect(mszReader,&pIASCard);

// …
```

L'API IAS va permettre d'effectuer différentes opérations sur les données de la carte :

- Lire des objets dont le contrôle d'accès est en lecture libre ou protégée par le PIN global.

- Consulter les contrôles d'accès des différents objets.

- Administrer la carte par un canal transparent.

1.3.1 - Les objets en lecture libre ou protégés par PIN Global

L'API permet de gérer différents types d'objets à condition qu'ils soient créés au format PKCS15 par l'émetteur de la carte. Dans tous les autres formats il faudra se reporter sur l'usage du « canal transparent »

Les objets fichiers

Les classes associées à ces objets sont décrites dans la section 5.2 - elles vont permettre de décomposer les principales structures de fichiers IAS.

Par exemple, la classe CApplicationDirectoryFile sera utilisée pour lire les données du fichier EFDIR et on connaîtra ainsi la liste de toutes les applications contenue dans la carte IAS. Pour un exemple complet voir section 7.2 -

Les objets de données

Les classes associées à ces objets sont décrites dans la section 5.3 - elles vont permettre de décomposer les informations des objets de type PKCS#15.

Par exemple, on pourra obtenir les informations d'un certificat à savoir l'émetteur du certificat, le condensé du certificat, etc.… Pour un exemple complet voir section 7.2 -

Ainsi la plupart des données de la carte pourront être lues (après vérification des contrôles d'accès) et cela quelle que soit l'application à laquelle elles appartiennent.

1.3.2 - Consultation des contrôles d'accès

Avant toute lecture ou modification de données, il est nécessaire de savoir quelles sont les conditions d'accès. Pour les obtenir, on utilisera les méthodes suivantes :

- fileControlParameters() de la classe CCiafile pour les objets fichiers.
- accesControlRules() de la classe CCioObject pour les objets de données.

```
IASRESULT result;

// Initialisation del'API IAS
IASCONTEXT iasContext;
result = IASInitialize(&iasContext);

// Récupération de la liste des lecteurs
char mszReaders[1000];
DWORD cchReaders=sizeof(mszReaders);
result = IASListReaders(iasContext, mszReaders,&cchReaders);

// Tentative de connexion à la carte présente dans le 1er lecteur
CIASCard *pIASCard;
result= IASConnect(mszReader,&pIASCard);

// Lecture du répertoire de l'application à partir de la carte (EF 2F00)
CCiaPath dirPath;
dirPath.setEfidOrPath(_HB("2F 00"));

CApplicationDirectoryFile efDIR(pIASCard, dirPath);
efDIR.load();

// Lecture des conditions d'accès du fichier EFDIR
CFileControlParameter FCP = efDIR.fileControlParameters();
FileType type = FCP.type();
FileState state = FCP.state();

// Voir section 7.2 pour la récupération d'un objet de fichier CCertificateFile

// On récupère tous les certificates puis le 1er certificat
std::vector<CCioCertificate *> certs = certFile.objects();
std::vector<CCioCertificate *>::const_iterator cert;
cert=certs.begin();

// On extrait la liste des contrôles d'accès de cert, puis le 1er
std::vector<CCioAccessControlRule> ACRs = cert.accessControlRules();
std::vector<CCioAccessControlRule>::const iterator ACR ;
ACR=ACRs.begin();

AccessModes AM = ACR.accessMode();

// …
```

1.3.3 - Administration de la carte par canal transparent

L'établissement d'un canal transparent avec la carte IAS va permettre d'envoyer directement des commandes APDU (Application Protocol Data Unit) à la carte en passant par l'API IAS.

Il est à noter que l'usage du canal transparent implique que le développeur maitrise l'intégralité des commandes APDUs, c'est-à-dire la connaissance précise des commandes IAS attendues par la carte.

Pour cela, dans la section 5.1.1 est décrite la classe CIASCard, elle va permettre de communiquer de manière transparente avec la carte sans présupposer un stockage au format PKCS#15 pour les objets cryptographiques ou non.

On pourra alors :

- Lire les données protégées par canal sécurisée non accessibles directement par l'API IAS.
- Modifier les données.
- Gérer le PIN global comme par exemple le débloquer si besoin.
- Réaliser l'opération de signature qualifiée, protégée par canal sécurisé. A noter que pour la réalisation de la signature qualifiée le dernier tour de hash doit être réalisé par la carte. Le téléservice de signature qualifiée devra donc faire au préalable le calcul du hash partiel. Ce type d'opération pourra être réalisé avec un outil standard présent sur la marché, du type Open SSL."

Exemple :

```
IASRESULT result;

// Initialisation del'API IAS
IASCONTEXT iasContext;
result = IASInitialize(&iasContext);

// Récupération de la liste des lecteurs
char mszReaders[1000];
DWORD cchReaders=sizeof(mszReaders);
result = IASListReaders(iasContext, mszReaders,&cchReaders);

// Tentative de connexion à la carte présente dans le 1er lecteur
CIASCard *pIASCard;
result= IASConnect(mszReader,&pIASCard);

// Etablissement du canal transparent
pIASCard.beginTransaction();

// Envoi d'une commande APDU
pIASCard.sendAPDU(…) ;

// Fermeture du canal transparent
pIASCard.endTransaction() ;

IASFinalize(iasContext);
```

1.3.4 - Gestion des lecteurs avec écran/clavier

Ce chapitre décrit comment les applications peuvent faire usage de lecteurs sécurisés de type “pinpad » c'est-à-dire étant équipés d’un clavier permettant l’entrée du PIN de manière sécurisée (Secured Pin Entry).

Pour communiquer avec les cartes, les applications utilisent traditionnellement les méthodes **IASListReaders** et **IASConnect** pour obtenir en retour un handle de connexion valide qui leur permet d’utiliser tous les services offerts par l’API IAS.

L’API IAS permet en outre au travers de la méthode **CIASCard getPCSCHandle()** de recevoir en retour la valeur du handle de connexion au niveau PC/SC et donc de bénéficier des services offerts par le « ressource manager » en charge de communiquer avec les lecteurs.

Il sera ainsi possible de communiquer directement avec le lecteur au travers d’appels à la fonction SCardControl (fonction fournie par PC/SC). Par exemple, si le lecteur « pinpad » est conforme aux commandes PC/SC v2, l’application devra se conformer à ce standard pour tirer avantage de la fonction « pinpad ».

1.3.5 - Erreurs

Les erreurs se produisant pendant l’exécution des fonctions de l’API IAS décrites dans la section 4 sont envoyées aux applications via les codes de retour d’erreur des fonctions définis dans la section 3.1. Les applications ne doivent pas admettre d’exécution correcte, tous les codes de retour des fonctions doivent être vérifiés et les applications doivent se comporter en conséquence.

Dès que le contexte de l’API IAS est correctement établi, les applications peuvent accéder aux objets de l’API décrit dans la section 5. Les erreurs survenant pendant la construction des objets et les appels aux fonctions des objets sont notifiés au travers d’exception.

Les applications doivent attraper les classes d’exception spécifique de l’API IAS regroupant toutes les erreurs spécifiques ; il en est de même pour les exceptions de l’OS et des entrées/sorties.

2 - Références

Référence du Document	Titre Document
[PKCS #11]	PKCS #11 v2.20. http://www.rsasecurity.com/rsalabs
[CryptoAPI]	Cryptographic API in Windows. http://msdn.microsoft.com
[IAS]	“FR Common IAS Platform Specification ”, Revision: 1.01 Premium
[ISO 7816-4]	ISO/IEC 7816-4 Second Edition 2005-01-15
[ISO 7816-15]	ISO/IEC 7816-15 First Edition 2004-01-15

3 - Types des données

Cette section décrit les différents types de base, les structures et noms associés aux valeurs du contenu de la carte.

3.1 - IASRESULT

IASRESULT est le type de code de retour des fonctions de l'API IAS.

```
typedef unsigned long IASRESULT;
```

Valeurs possibles :

IASRESULT_OK	0x00000000	Succès
IASRESULT_ALREADY_INITIALIZED	0x80000001	L'API ne peut pas être initialisée une deuxième fois
IASRESULT_NOT_INITIALIZED	0x80000002	L'API doit être initialisée avant tout appel à d'autres fonctions
IASRESULT_FUNCTION_FAILED	0x80000002	Erreur interne pendant le traitement de la demande
IASRESULT_NOT_ENOUGH_SPACE	0x80000003	Incapable de traiter la demande en raison de l'espace fourni insuffisant
IASRESULT_NO_CARD	0x80000004	Pas de carte dans le lecteur

3.2 - IASCONTEXT

IASCONTEXT est un handle sur un contexte d'API IAS initialisé.

```
typedef unsigned long IASCONTEXT;
```

3.3 - IASINFO

IASINFO est une structure qui donne des informations sur l'API IAS.

```
typedef struct {
      IASVERSION IASAPIVersion;
} CK_INFO;
```

3.4 - IASVERSION

IASVERSION est une structure représentant la version.

```
typedef struct {
      BYTE major;
      BYTE minor;
} IASVERSION;
```

3.5 - CioChoice

CioChoice est une énumération des classes IAS. Elle est particulièrement utile lorsque l'on décompose une structure de type répertoire (EFod) d'une application. Se référer à la norme [ISO 7816-15] *CIOChoice* pour une description complète.

```
enum   CioChoice
  {
      ctPrivateKeys         = 0,
      ctPublicKeys          = 1,
      ctTrustedPublicKeys   = 2,
      ctSecretKeys          = 3,
      ctCertificates        = 4,
      ctTrustedCertificates = 5,
      ctUsefulCertificates  = 6,
      ctDataContObjects     = 7,
      ctAuthObjects         = 8
};
```

3.6 - CommonObjectFlags

CommonObjectFlags est une énumération des flags IAS décrivant l'étendue des accès de l'objet. Se référer à la norme [ISO 7816-15] *CommonObjectFlags* pour une description complète.

```
enum CommonObjectFlags
      {
      flagPrivate    = 0x01,
      flagModifiable = 0x02,
      flagInternal = 0x04
};
```

3.7 - SecurityEnvironmentInfo

SecurityEnvironmentInfo est une structure décrivant un environnement de sécurité. Se référer à la norme [ISO 7816-15] *SecurityEnvironmentInfo* pour une description complète.

```
struct SecurityEnvironmentInfo {
      std::string owner;
      CBlob aid;
      };
```

3.8 - AlgorithmInfo

AlgorithmInfo est une structure donnant la description d'un algorithme supporté. Se référer à la norme [ISO 7816-15] *AlgorithmInfo* pour une description complète.

```
struct AlgorithmInfo {
      int reference;
      int algorithm;
      CBlob parameters;
      int supportedOperations;
      std::string objId;
      int algRef;
      };
```

3.9 - PasswordType

PasswordType est une énumération des types de mots de passe d'authentification. Se référer à la norme [ISO 7816-15] *PasswordType* pour une description complète.

```
enum PasswordType {
      typeBcd           = 0,
      typeAsciiNumeric  = 1,
      typeUtf8          = 2,
      typeHalfNibbleBcd = 3,
      typeIso9564_1     = 4
      };
```

3.10 - Validity

Validity est une structure contenant les périodes de validité des certificats.

```
struct Validity {
      std::string notBefore;
      std::string notAfter;
      };
```

3.11 - ObjectValueType

ObjectValueType est une énumération donnant les différents types d'emplacements de stockage de valeur d'objet (directement ou indirectement via le chemin, l'url…). Se référer à la norme [ISO 7816-15] pour une description complète.

```
enum ObjectValueType {
      ovtNone,
      ovtDirect,
      ovtPath,
      ovtUrlPrintableString,
      ovtUrlIA5String,
      ovtUrlWithDigest
};
```

3.12 - FileType

FileType est une énumération donnant les différents types de fichiers de la carte. Se référer à la norme [ISO 7816-4] pour une description complète.

```
enum FileType {
      ftNone,
      ftApplication,
      ftDedicated,
      ftTransparent,
      ftLinearFixed,
      ftLinearVariable,
      ftCyclic,
      ftTLV
};
```

3.13 - FileState

FileState est une énumération donnant les différents statuts de fichiers de la carte. Se référer à la norme [ISO 7816-4] pour une description complète.

```
enum FileState {
      fsNone,
      fsCreation,
      fsInitialisation,
      fsOperationalActivated,
      fsOperationalDeactivated,
      fsTermination,
      fsProprietary
};
```

3.14 - IdType

IdType est une énumération donnant les différents types d'identifiants. Se référer à la norme [ISO 7816-15] *KeyIdentifiers* pour une description complète.

```
enum IdType {
      idNone                        = 0,
      idIssuerAndSerialNumber       = 1,
      idSubjectKeyId                = 2,
      idIssuerAndSerialNumberHash   = 3,
      idSubjectKeyHash              = 4,
      idIssuerKeyHash               = 5,
      idIssuerNameHash              = 6,
      idSubjectNameHash             = 7,
      idPgp2KeyId                   = 8,
      idOpenPGPKeyId                = 9
};
```

3.15 - AccessModes

AccessModes est une énumération fournissant les différents modes d'accès. Se référer à la norme [ISO 7816-15] *AccessMode* pour une description complète.

```
enum AccessModes {
      flagRead= 0x01,
      flagUpdate = 0x02,
      flagExecute  = 0x04,
      flagDelete = 0x08
};
```

3.16 - Operations

Operations est une énumération des opérations possibles avec une clé. Se référer à la norme [ISO 7816-15] *Operations* pour une description complète.

```
enum Operations {
      flagOperComputeChecksum    = 0x01,
      flagOperComputeSignature   = 0x02,
```

```
      flagOperVerifyChecksum     = 0x04,
      flagOperVerifySignature    = 0x08,
      flagOperEncipher           = 0x10,
      flagOperDecipher           = 0x20,
      flagOperHash               = 0x40,
      flagOperGenerateKey        = 0x80
};
```

3.17 - ContextTag

ContextTag est une énumération des références possibles de **RecordInfo.** Se référer à la norme [ISO 7816-15] *RecordInfo* pour une description complète.

```
enum ContextTag {
      oDRecordLength    = 0,
      prKDRecordLength  = 1,
      puKDRecordLength  = 2,
      sKDRecordLength   = 3,
      cDRecordLength    = 4,
      dCODRecordLength  = 5,
      aODRecordLength   = 6
}
```

3.18 - SecurityConditionType

SecurityConditionType est une énumération des types de conditions de sécurité. Se référer à la norme [ISO 7816-15] *SecurityCondition* pour une description complète.

```
enum SecurityConditionType {
      sctNone,
      sctAlways,
      sctAuthId,
      sctAuthReference,
      sctNot,
      sctAnd,
      sctOr
}
```

3.19 - SecurityConditionByte

SecurityConditionByte est une énumération fournissant les conditions de sécurité possibles pour des modes d'accès aux différents fichiers. Se référer à la norme [ISO 7816-15] *SecurityConditionByte* pour une description complète.

```
enum AccessModeDF {
  scbAlways          = 0x00, // accès libre
  scbNever           = 0xFF, // pas d'accès

  scbSeidMask        = 0x0F, // si aucun ci-dessus, ce masque est utilisé pour
                             // obtenir seID associé aux conditions
                             // d'accès suivantes. Le masque complémentaire est
                             // utilisé pour obtenir les condition d'accès
                             // suivantes :
  scbAllCondition    = 0x80, // When set indicates all following conditions must
```

```
                                // apply, any of these otherwise
   scbSecureMessaging = 0x40, // requiert un canal sécurisé
   scbExternalAuth    = 0x20, // requiert une authentification externe
   scbUserAuth        = 0x10  // requiert une authentification de l'utilisateur
}
```

3.20 - AccessModes

AccessModeDF, **AccessModeEF**, **AccessModeDO** sont des énumérations fournissant les modes d'accès possibles aux entités. Se référer à la norme [ISO 7816-15] *AccessModeByte* pour une description complète. **AccessModePrivKey**, **AccessModePublKey**, **AccessModeSymKey**, **AccessModeSymKeySet**, **AccessModeUserAuth** sont des énumérations fournissant des modes d'accès additionnels spécifiques IAS aux entités. Se référer à la norme [ISO 7816-15] *SMB for SD0* pour une description complète.

```
enum AccessModeDF {
      amdfDeleteSelf     = 0x40,
      amdfTerminateDF    = 0x20,
      amdfActivateFile   = 0x10,
      amdfDeactivateFile = 0x08,
      amdfCreateDF       = 0x04,
      amdfCreateEF       = 0x02,
      amdfDeleteChild    = 0x01
}

enum AccessModeEF {
      amefDeleteFile     = 0x40,
      amefTerminateEF    = 0x20,
      amefActivateFile   = 0x10,
      amefDeactivateFile = 0x08,
      amefWrite          = 0x04,
      amefUpdate         = 0x02,
      amefRead           = 0x01
}

enum AccessModeDO {
      amdoMSE            = 0x04,
      amdoPutData        = 0x02,
      amdoGetData        = 0x01
}

enum AccessModePrivKey  {
      amprPsoCds       = 0x40,
      amprIntAuth      = 0x20,
      amprPsoDecipher  = 0x10,
      amprGenAKP       = 0x08,
      amprPutData      = 0x02,
      amprGetData      = 0x01
}

enum AccessModePublKey   {
      ampuPsoVerCert   = 0x40,
      ampuExtAuth      = 0x20,
      ampuPsoEncipher  = 0x10,
      ampuGenAKP       = 0x08,
      ampuPutData      = 0x02,
      ampuGetData      = 0x01
}
```

```
enum AccessModeSymKey{
      amskIntAuth      = 0x10,
      amskMutAuth      = 0x08,
      amskPutData      = 0x02,
      amskGetData      = 0x01
}

enum AccessModeSymKeySet {
      amsksExtAuth     = 0x20,
      amsksMutAuth     = 0x08,
      amsksPutData     = 0x02,
      amsksGetData     = 0x01
}

enum AccessModeUserAuth  {
      amuaChRefData    = 0x40,
      amuaVerify       = 0x20,
      amuaResetTryCnt  = 0x10,
      amuaPutData      = 0x02,
      amuaGetData      = 0x01
}
```

4 - Fonctions

4.1 - Fonctions Générales

4.1.1 - IASInitialize

IASInitialize initialise la librairie IAS. Une application doit appeler cette fonction avant toute autre, et appeler **IASFinalize** quand l'utilisation de l'API IAS est terminée.

```
IASRESULT IASInitialize(IASCONTEXT *pIASContext)
```

pIASContext [out] reçoit un handle sur un contexte IAS lors d'une exécution réussie.

Valeurs retournées :
IASRESULT_ALREADY_INITIALIZED, IASRESULT_FUNCTION_FAILED, IASRESULT_OK.

4.1.2 - IASFinalize

IASFinalize est appelée lorsqu'une application a fini d'utiliser l'API IAS. Elle doit être la dernière fonction de l'API IAS appelée par une application.

```
IASRESULT IASFinalize(IASCONTEXT IASContext)
```

IASContext [in] est le handle de contexte IAS reçu lors d'un succès de l'appel à **IASInitialize.**

Valeurs retournées :
IASRESULT_NOT_INITIALIZED, IASRESULT_FUNCTION_FAILED, IASRESULT_OK.

4.1.3 - IASGetInfo

IASGetInfo renvoie les informations à propos de l'API IAS.

```
IASRESULT IASGetInfo(IASINFO *pIASInfo)
```

pIASInfo [out] est une structure recevant les informations de l'API IAS.

Valeurs retournées :
IASRESULT_FUNCTION_FAILED, IASRESULT_OK.

4.2 - Fonctions de gestion des lecteurs et des cartes

4.2.1 - IASListReaders

IASListReaders renvoie une liste des lecteurs connectés.

```
IASRESULT IASListReaders(IASCONTEXT IASContext, LPSTR mszReaders, LPDWORD
pcchReaders)
```

IASContext [in] est le contexte IAS reçu lors d'un appel sans erreur à **IASInitialize.**

mszReaders [out] Buffer pour stocker la liste des lecteurs connectés. Chaque nom de lecteur se termine par le caractère NULL et la liste par l'ajout d'un caractère NULL. Si la valeur fournie est NULL, **IASListReaders** ignore la taille du buffer renseignée dans **pcchReaders**, elle écrit la longueur du buffer qu'elle aurait du retourner si ce paramètre n'avait pas été NULL dans *pcchReaders***,** et renvoie un code de succès.

pcchReaders [in, out] En entrée précise la taille allouée pour le buffer *mszReaders*. Lors d'une exécution réussie, ce paramètre reçoit la taille retournée dans la liste *mszReaders.*

Valeurs retournées :
IASRESULT_NOT_INITIALIZED, IASRESULT_NOT_ENOUGH_SPACE, IASRESULT_FUNCTION_FAILED, IASRESULT_OK.

4.2.2 - IASConnect

IASConnect ouvre une session avec la carte insérée dans le lecteur spécifié.

```
IASRESULT IASConnect(IASCONTEXT IASContext, LPSTR szReader, CIASCard **ppIASCard)
```

IASContext [in] est le contexte IAS reçu lors d'un appel réussi à **IASInitialize**.
szReader [in] est le nom du lecteur auquel l'API IAS tente de se connecter.
ppIASCard [out] Lors d'une exécution réussie, contient la référence d'un objet de type CIASCard.

Valeurs retournées :
IASRESULT_NOT_INITIALIZED, IASRESULT_NOCARD, IASRESULT_FUNCTION_FAILED, IASRESULT_OK.

Note :
Il est possible de créer un objet de type CIASCard directement en fournissant une valeur de type SCARDHANDLE obtenue directement par l'application auprès de l'API PS/SC.
Note 2 :
La fermeture de la communication avec la carte et la libération des ressources systèmes doivent être effectuées par l'application en supprimant l'objet CIASCard.

5 - Objets

Note : Se référer au profil de personnalisation de la carte afin de vérifier si un environnement de sécurité doit être initialisé avant tout appel à une fonction spécifique de la carte.

5.1 - Objets de base

5.1.1 - CIASCard

CIASCard est un objet provenant des fonctions de base de la norme ISO 7816-4 pour un accès transparent à la carte et l'identification de la carte.

```
CIASCard
   CIASCard()
   CIASCard(SCARDHANDLE hCard)
   void verify (unsigned char p2, CBlob const &data)
   void changePIN(unsigned char p2, CBlob const &data)
   CBlob atr()
   void reset()
   void beginTransaction()
   void endTransaction()
   void sendAPDU (  unsigned char cla, unsigned char ins,
                    unsigned char p1, unsigned char p2
                    CBlob const & data, unsigned long ne,
                    CBlob &response, unsigned short &sw12)
   SCARDHANDLE getpcschandle ()
```

```
CIASCard()
```

Les applications devront employer la fonction **IASConnect** pour récupérer un nouvel objet de type **CIASCard** connecté à une carte valide.

```
CIASCard(SCARDHANDLE hCard)
```

Les applications devront employer la fonction **IASConnect** pour récupérer un nouvel objet de type CIASCard connecté à une carte valide, ou peuvent utiliser ce constructeur, en fournissant un handle PC/SC valide, établi au préalable par l'application elle même.

```
void verify (unsigned char p2, CBlob const &data)
```

Permet de vérifier les données de référence data fournies avec celles enregistrées dans la carte et identifiées par *p2*. Lors d'une vérification réussie, le niveau de sécurité associé est validé et les opérations associées sont permises.
Se référer à la norme [ISO 7816-4] pour plus d'informations.

```
void changePIN(unsigned char p2, CBlob const &data)
```

Permet de modifier le code secret sur la carte. Le code secret doit être vérifié avant d'exécuter cette opération.
Note : Les applications devraient entourer les appels aux méthodes **verify** et **changePIN** avec **beginTransaction** et **endTransaction**.

```
CBlob atr()
```

Utilisez cette fonction pour obtenir un ATR (Answer To Reset) de la carte connectée, permettant d'identifier le type et le modèle de la carte. Se référer à la norme [ISO 7816-4] pour une description complète.

```
void reset()
```

Appelez cette fonction pour demander la réinitialisation de la carte.

```
void beginTransaction()
```

Utilisez cette fonction pour demander le mode d'accès exclusif à la carte et la garantie de toutes les commandes suivantes séquentiellement. Les autres applications sont bloquées jusqu'à ce qu'un appel à **endTransaction** soit exécuté. Il est demandé aux applications de finir la transaction dès que possible. Elles doivent également s'assurer que chaque appel à **beginTransaction** a un **endTransaction** qui lui correspond. Il est fortement recommandé d'utiliser cette commande dans les applications établissant un canal sécurisé, comme par exemple un serveur d'administration.

```
void endTransaction()
```

Appelez cette fonction pour finir le mode d'accès exclusif à la carte établi avec **beginTransaction** et pour débloquer les autres applications.

```
void sendAPDU (unsigned char cla, unsigned char ins,
      unsigned char p1, unsigned char p2
      CBlob const & data, unsigned long ne,
      CBlob &response, unsigned short &sw12)
```

Cette fonction permet d'envoyer une commande APDU à la carte. Se référer à la norme [ISO 7816-4] pour une description complète.
Note : Les applications qui envoient des commandes APDU multiples devraient utiliser les méthodes **beginTransaction** et **endTransaction** pour garantir si nécessaire le séquencement des commandes APDU.

```
SCARDHANDLE getPCSCHandle ()
```

Cette fonction permet de récupérer le handle au niveau PC/SC. Ce handle peut être utilisé par exemple pour envoyer des commandes de présentation de PIN à des lecteurs munis de clavier. Pour cela, PC/SC offre des fonctions de type « SCardControl » qui prend ce paramètre en compte. Se reporter ensuite à la documentation du lecteur PIN/PAD.

5.1.2 - IASException

IASException regroupe et décrit les erreurs apparaissant durant l'exécution des fonctions de l'API IAS.

```
CIASException
      CIASException(unsigned long errorCode)
      void raise()
      std::string descriptionA() const
      std::wstring descriptionW() const
      unsigned long error() const
      unsigned int sw() const
```

```
CIASException(unsigned long errorCode)
```

Constructeur.

```
void raise()
```

Lève l'exception.

```
std::string descriptionA() const
```

Donne une description de l'exception.

```
std::wstring descriptionW() const
```

Donne une description de l'exception sous forme d'une chaîne de caractères longs (*wide chars*).

```
unsigned long error() const
```

Renvoie le code d'erreur de l'exception.

```
unsigned int sw() const
```

Récupère le SW1/SW2 (StatusWord1/StatusWord2) de la carte associé à une exception. Renvoie 0 si non applicable.

5.1.3 - CCiaPath

CCiaPath est la représentation du chemin d'un objet CIA. Cette classe doit être utilisée afin de décrire l'endroit où l'on peut retrouver un objet dans l'application CIA. (Par exemple dans quel fichier, avec quel index et quelle est la longueur allouée à l'objet)

```
CCiaPath class
   CCiaPath ()
   CCiaPath (CBlob const &blob)
   CCiaPath(CCiaPath const & that)
   CCiaPath & operator=(CCiaPath const & that)
   void setEfidOrPath(CBlob const & efidOrPath)
   void setIndex(int index)
   void setLength(int length)
   void setAid(CBlob const & aid)
   CBlob efidOrPath ()
   int index()
   int length()
   CBlob aid() const
   CCiaPath relativeToEF(CCiaPath const & efPath) const
   CCiaPath relativeToDF(CCiaPath const & dfPath) const
```

```
CCiaPath ()
```

Ce constructeur permet de créer un objet vide. L'objet peut être initialisé avec l'opérateur = ou la fonction setEfidOrPath.

```
CCiaPath (CBlob const &blob)
```

Ce constructeur permet de créer une représentation du chemin d'un fichier à partir d'un CBlob**.** En spécifiant un chemin direct, il peut être utile pour les applications d'utiliser la macro _**HB** pour fournir directement la chaîne de caractères du chemin. (par exemple **_HB("2F00")**)

```
CCiaPath (CCiaPath const & that)
```

Constructeur par copie.

```
CCiaPath & operator=(CCiaPath const & that)
```

Opérateur d'affectation pour initialiser un objet à partir d'un autre.

```
void setEfidOrPath(CBlob const & efidOrPath)
```

Cette fonction permet d'initialiser ou de modifier le chemin vers lequel pointe l'objet. Voir **CCiaPath (CBlob const &blob)** pour la description des paramètres.

```
void setIndex(int index)
```

Cette fonction permet d'initialiser ou de changer l'index (ou numéro d'enregistrement) définissant un emplacement spécifique de départ dans le fichier référencé. Par défaut, l'index de départ est défini à 0.

```
void setLength(CBlob const & efidOrPath)
```

Cette fonction permet d'initialiser ou de changer la longueur définissant une partie spécifique dans le fichier référencé, en commençant à l'index défini (s'il y en a un).

```
void setAid(CBlob const & aid)
```

L'appel de cette fonction initialise ou change l'aid définissant dans quelle application le fichier référencé s'applique.

```
CBlob efidOrPath ()
```
Permet de récupérer le chemin du fichier vers lequel pointe l'objet.

```
int index()
```
Appelez cette fonction pour obtenir l'emplacement de départ dans le fichier référencé vers lequel pointe l'objet.

```
int length()
```
Appelez cette fonction pour obtenir la longueur de l'emplacement dans le fichier référencé vers lequel pointe l'objet.

```
CBlob aid () const
```
L'Appel de cette fonction récupère l'identifiant de l'application à laquelle l'objet appartient.

```
CCiaPath relativeToEF(CCiaPath const & efPath) const
```
Appelez cette fonction pour construire un nouvel objet **CCiaPath** contenant une version absolue de l'objet **CCiaPath** qui contient le chemin relatif et une autre référence de l'objet **CCiaPath** contenant le chemin absolu de EF**.**

```
CCiaPath relativeToDF(CCiaPath const & dfPath) const
```
Appelez cette fonction pour construire un nouvel objet **CCiaPath** contenant une version absolue de l'objet **CCiaPath** contenant un chemin relatif, et une autre référence de l'objet **CCiaPath** contenant un chemin absolu de DF.

Note : Quand un fichier de [ISO 7816-15] référence un autre fichier, (par exemple EF CD référence un certificat), le chemin vers la référence du fichier est relatif au fichier qui le référence.Les objets fichiers de l'API ont besoin d'un chemin absolu, vous devez utiliser les méthodes **relativeToEF** et **relativeToDF** décrites précédemment pour convertir le chemin d'un fichier de relatif vers absolu.

Exemple :

Si EF A contient une référence de EF B:

```
CCiaPath relativePathOfB;
CCiaPath absolutePathOfA;
```

Alors

```
CCiaPath absolutePathOfB = relativePathOfB.relativeToEF(absolutePathOfA);
```

De la même façon, avec le chemin absolu de DF qui contient A:

```
CCiaPath relativePathOfB;
CCiaPath absolutePathOfDFContainingA;
```

Alors

```
CCiaPath absolutePathOfB = relativePathOfB.relativeToDF(absolutePathOfDFContainingA);
```

5.1.4 - CBlob

CBlob est la représentation d'un buffer binaire, très utilisé par les autres classes. Une application type manipule généralement de nombreux **CBlob**, tandis qu'elle n'en crée que quelques uns par elle même. Néanmoins, la création d'un **CBlob** est utile -presque seulement- pour les chemins de fichiers et les aids d'applications, dans ce cas la macro **_HB** peut être intéressante. Les applications voulant afficher le contenu d'un **CBlob** peuvent employer les macros **_HB** et **AsString.**

```
CBlob
   CBlob()
CBlob::CBlob(const std::string &s)
   CBlob(unsigned char const *pBuffer, size_t nCount=0)
   CBlob(size_t nCount, unsigned char const &val)
   CBlob(const CBlob &blob)
   CBlob & operator= (unsigned char const &rt)
   CBlob & operator+= (CBlob const &rt)
   bool   operator== (CBlob const &rother) const
   unsigned char &  operator[] (size_t nIdx)
   size_t size () const
   size_t length () const
   CBlob    substr(size_t offset, size_t count) const
```

`CBlob()`

Constructeur par défaut, crée un blob vide.

`CBlob(const std::string &s)`

Crée un blob initialisé à partir d'une chaîne de caractères.

`CBlob(size_t nCount, unsigned char const &val)`

Crée un blob initialisé avec la valeur donnée *val.*

`CBlob(unsigned char *pBuffer, size_t nCount = 0)`

Crée un blob initialisé avec *nCount* octets à l'adresse de *pBuffer.* Si *nCount* n'est pas fourni *pBuffer* est traité comme une chaîne de caractère se terminant par NULL.

`CBlob(const CBlob &blob)`

Constructeur par copie.

`CBlob & operator= (unsigned char const &rt)`

Opérateur d'affectation.

`bool  operator+= (CBlob const &rother) const`

Opérateur de concaténation.

`unsigned char &   operator[] (size_t nIdx)`

Opérateur d'indexation, permet d'obtenir l'octet à l'adresse *nIdx.*

`bool  operator== (CBlob const &rother) const`

Opérateur de comparaison de blob.

`size_t  size () const`

Retourne la taille du blob allouée en octets.

`size_t  length () const`

Retourne la taille en octets du blob actuel.

`CBlob & substr(size_t offset, size_t count) const`

Renvoie un nouveau CBlob initialisé avec une sous partie du blob, commençant à l'emplacement *offset* et contenant *count* octets.

5.1.5 - CCioSecurityCondition

CCioSecurityCondition est la représentation des conditions d'accès sécurisées d'un objet de la carte. Se référer à la norme [ISO 7816-15] *SecurityCondition* pour une description complète.

```
CCioSecurityCondition
      CCioSecurityCondition ()
      CCioSecurityCondition (CBlob const &blob)
      CCioSecurityCondition (CCioSecurityCondition const &that)
      CCioSecurityCondition & operator= (CCioSecurityCondition const &that)
      SecurityConditionType type() const
      CBlob authId() const
      CCioAuthReference authReference() const
      CCioSecurityCondition not() const
      std::vector< CCioSecurityCondition > and()
      std::vector< CCioSecurityCondition > or()
```

`CCioSecurityCondition ()`

Constructeur par défaut.

`CCioSecurityCondition (CBlob const &blob)`

Crée et initialise un objet avec le blob fourni.

`CCioSecurityCondition (CCioSecurityCondition const &that)`

Constructeur par copie.

`CCioSecurityCondition & operator= (CCioSecurityCondition const &that)`

Opérateur d'affectation.

`SecurityConditionType type() const`

Renvoie le type de conditions. Selon le type de conditions, les méthodes **authId**, **authReference**, **not**, **and** ou **or** peuvent être appelées pour obtenir les détails des conditions.

`CBlob authId() const`

Retourne l'id d'autorisation impliqué dans le contrôle d'accès à l'objet.

`CCioAuthReference authReference() const`

Renvoie la référence de l'autorisation impliquée dans le contrôle d'accès à l'objet.

`CCioSecurityCondition not() const`

Renvoie les conditions de sécurité impliquées dans le contrôle d'accès à l'objet avec l'opération **not**.

`std::vector< CCioSecurityCondition > and()`

Renvoie les conditions de sécurité impliquées dans le contrôle d'accès à l'objet avec l'opération **and**.

`std::vector< CCioSecurityCondition > or()`

Renvoie les conditions de sécurité impliquées dans le contrôle d'accès à l'objet avec l'opération **or**.

5.1.6 - CCioAuthReference

CCioAuthReference est la représentation des méthodes spécifiques d'authentification à la carte. Se référer à la norme [ISO 7816-15] *AuthReference* pour une description complète.

`CCioAuthReference`

```
CCioAuthReference ()
CCioAuthReference (CBlob const &blob)
CCioAuthReference (CCioAuthReference const &that)
CCioAuthReference & operator= (CCioAuthReference const &that)
int authMethod() const
int seIdentifier() const
```

`CCioAuthReference ()`

Constructeur par défaut.

`CCioAuthReference (CBlob const &blob)`

Crée et initialise un objet avec le blob fourni.

`CCioAuthReference (CCioAuthReference const &that)`

Constructeur par copie.

`CCioAuthReference & operator= (CCioAuthReference const &that)`

Opérateur d'affectation.

`int authMethod() const`

Renvoie les méthodes d'authentification sous la forme d'un champ de bits. Se référer à la norme [ISO 7816-15] *AuthMethod* pour une description complète.

`int seIdentifier() const`

Renvoie l'id de l'environnement de sécurité impliqué dans les contrôles d'accès. Se référer à la norme [ISO 7816-15] *AuthReference* pour une description complète.

5.1.7 - CCioAccessControlRule

CCioAccessControlRule est une représentation des règles de contrôle d'accès de la carte. Se référer à la norme [ISO 7816-15] *AccessControlRule* pour une description complète.

```
CCioAccessControlRule
CCioAccessControlRule()
CCioAccessControlRule (CBlob const &blob)
CCioAccessControlRule (CCioAccessControlRule const &that)
CCioAccessControlRule & operator= (CCioAccessControlRule const &that)
AccessModes accessMode ()
CCioSecurityCondition securityCondition() const
```

`CCioAccessControlRule ()`

Constructeur par défaut.

`CCioAccessControlRule (CBlob const &blob)`

Crée et initialise un objet avec le blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

`CCioAccessControlRule (CCioAccessControlRule const &that)`

Constructeur par copie.

`CCioAccessControlRule & operator= (CCioAccessControlRule const &that)`

Opérateur d'affectation.

`AccessModes accessMode ()`

Renvoie les modes d'accès sous forme d'un champ de bits. Voir **AccessModes** pour les valeurs de masques possibles.

```
CCioSecurityCondition securityCondition ()
```
Renvoie les conditions d'accès.

5.1.8 - CCioObjectValue

CCioObjectValue est une représentation de la valeur d'un objet. Se référer à la norme [ISO 7816-15] *ObjectValue* pour une description complète.

```
CCioObjectValue
  CCioObjectValue()
  CCioObjectValue(CBlob const &blob)
  CCioObjectValue (CCioObjectValue const &that)
  CCioObjectValue & operator= (CCioObjectValue const &that)
  ObjectValueType type ()
  CBlob value () const
```

```
CCioObjectValue()
```
Constructeur par défaut.

```
CCioObjectValue(CBlob const &blob)
```
Crée et initialise un objet avec le blob fourni.

```
CCioObjectValue (CCioObjectValue const &that)
```
Constructeur par copie.

```
CCioObjectValue & operator= (CCioObjectValue const &that)
```
Opérateur d'affectation.

```
ObjectValueType type ()
```
Retourne le type de la valeur de l'objet.

```
CBlob value () const
```
Renvoie la valeur de l'objet actuel.

5.1.9 - CFileControlParameters

CFileControlParameters est une représentation des paramètres de contrôle de fichier (FCP) de la carte. Se référer à la norme [ISO 7816-4] pour une description complète.

```
CFileControlParameters
  CFileControlParameters()
  CFileControlParameters(CFileControlParameters const & that)
  CFileControlParameters(CBlob const &blob)
  CFileControlParameters & operator= (CFileControlParameters const &that)
  operator CBlob () const
  FileType type () const
  int size () const
  CBlob fileIdentifier () const
  CBlob shortFileIdentifier () const
  CBlob dfName () const
  FileState state () const
  CSecurityAttributeCompact securityAttributeCompact() const
```

```
CBlob proprietaryBerTlvData () const
CBlob proprietaryNonBerTlvData () const
```

```
CFileControlParameters()
```
Constructeur par défaut.

```
CFileControlParameters(CBlob const &blob)
```
Crée et initialise un objet avec le blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

```
CFileControlParameters(CFileControlParameters const &that)
```
Constructeur par copie.

```
CFileControlParameters & operator= (CFileControlParameters const &that)
```
Opérateur d'affectation.

```
operator CBlob () const
```
Retourne la valeur brute du FCP comme spécifiée dans [ISO 7816-4].

```
FileType  type () const
```
Renvoie le type du fichier. Voir l'énumération de **FileType** pour les valeurs possibles.

```
int   size () const
```
Renvoie la taille du fichier.

```
CBlob   fileIdentifier () const
```
Renvoie l'identifiant du fichier.

```
CBlob   shortFileIdentifier () const
```
Renvoie l'identifiant court du fichier.

```
CBlob   dfName () const
```
Renvoie le nom du DF parent.

```
FileState  state () const
```
Renvoie l'état du fichier. Voir l'énumération **FileState** pour les valeurs possibles.

```
CSecurityAttributeCompact securityAttributeCompact() const
```
Renvoie les conditions d'accès au fichier comme étant un objet **CSecurityAttributeCompact**. Se référer à [ISO 7816-4] *Security Attributes/compact format.*

```
CBlob proprietaryBerTlvData () const
```
Renvoie des données propriétaires (ber tlv).

```
CBlob proprietaryNonBerTlvData () const
```
Renvoie des données propriétaires (non ber tlv).

5.1.10 - CSecurityAttributeCompact

CSecurityAttributeCompact représente l'attribut de sécurité dans une forme compacte comme définie dans [ISO 7816-4].

```
CSecurityAttributeCompact
      CSecurityAttributeCompact ()
```

```
CSecurityAttributeCompact (CSecurityAttributeCompact const & that)
CSecurityAttributeCompact (CBlob const &blob)
CSecurityAttributeCompact & operator=(CSecurityAttributeCompact const &that)
CSecurityAttributeCompact & operator=(CBlob const &blob)
operator CBlob () const
std::pair<bool, SecurityConditionByte> securityConditionByte(int amb) const
bool proprietary () const
```

```
CSecurityAttributeCompact()
```
Constructeur par défaut.

```
CSecurityAttributeCompact (CSecurityAttributeCompact const &that)
```
Constructeur par copie.

```
CSecurityAttributeCompact (CBlob const &blob)
```
Crée un objet **CSecurityAttributeCompact** initialisé en faisant une analyse syntaxique du blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

```
CSecurityAttributeCompact & operator= (CSecurityAttributeCompact const
&that)
```
Opérateur d'affectation.

```
CSecurityAttributeCompact & operator= (CBlob const &blob)
```
Initialise un objet **CSecurityAttributeCompact** à partir d'un blob.

```
operator CBlob () const
```
Renvoie le contenu brut d'un objet **CSecurityAttributeCompact** spécifié dans [ISO 7816-4].

```
std::pair<bool, SecurityConditionByte> securityConditionByte(int amb) const
```
Renvoie les conditions d'accès à l'objet pour le mode d'accès fourni. Selon le type d'entité, le mode d'accès *amb* est de type **AccessModeDF**, **AccessModeEF** ou **AccessModeDO**. Pour les modes d'accès spécifiques IAS, *amb* peut aussi être de type **AccessModePrivKey**, **AccessModePublKey**, **AccessModeSymKey**, **AccessModeSymKeySet**, **AccessModeUserAuth**.
Voir la méthode **proprietary()**.

```
bool proprietary() const
```
Renvoie *true*, si le paramètre *amd* de **SecurityConditionByte()** peut être interprété comme spécifié dans [IAS].

5.1.11 - CCioUsage

CCioUsage est une représentation des différentes utilisations des clés de la carte. Se référer à la norme [ISO 7816-15] *Usage* pour une description complète.

```
CCioUsage
  CCioUsage()
  CCioUsage(CBlob const &blob)
  CCioUsage(CCioUsage const & that)
```

```
   CCioUsage & operator= (CCioUsage const &that)
   int keyUsage () const
   std::vector< std::string >  extKeyUsage () const
```

```
CCioUsage()
```
Constructeur par défaut.

```
CCioUsage(CBlob const &blob)
```
Crée et initialise un objet avec le blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

```
CCioUsage(CCioUsage const & that)
```
Constructeur par copie.

```
CCioUsage & operator= (CCioUsage const &that)
```
Opérateur d'affectation.

```
int keyUsage () const
```
Renvoie le flag de l'utilisation de la clé comme spécifié dans [ISO 7816-15].

```
std::vector< std::string > extKeyUsage () const
```
Renvoie la liste des utilisations étendues de la clé, sous forme d'une liste d'OIDs.

5.1.12 - CCioCredentialIdentifier

CCioCredentialIdentifier est une représentation d'un identifiant de clés ou de certificats de la carte. Se référer à la norme [ISO 7816-15] *CredentialIdentifier* pour une description complète.

```
CCioCredentialIdentifier
   CCioCredentialIdentifier ()
   CCioCredentialIdentifier (CBlob const &blob)
   CCioCredentialIdentifier & operator= (CCioCredentialIdentifier const &that)
   IdType idType() const
   CBlob idValue () const
   CBlob octetStringIdValue () const
```

```
CCioCredentialIdentifier ()
```
Constructeur par défaut.

```
CCioCredentialIdentifier (CBlob const &blob)
```
Crée et initialise un objet avec le blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

```
CCioUsage & operator= (CCioUsage const &that)
```
Opérateur d'affectation.

```
IdType idType() const
```
Renvoie le type d'identifiant utilisé pour décrire la clé ou le certificat. Voir l'énumération de **IdType** pour les valeurs possibles et la méthode **idValue** pour obtenir la valeur de l'identifiant actuel.

```
CBlob idValue () const
```
Renvoie la valeur brute de l'identifiant encodée en DER.

```
CBlob octetStringIdValue () const
```

Renvoie la valeur de l'identifiant encodée en OCTET STRING

5.1.13 - CCioKeyInfo

CCioKeyInfo donne une représentation des informations d'une clé de la carte. Se référer à la norme [ISO 7816-15] *KeyInfo* pour une description complète.

```
CCioKeyInfo
   CCioKeyInfo()
   CCioKeyInfo(CBlob const &blob)
   CCioKeyInfo & operator= (CCioKeyInfo const &that)
   CBlob parameters() const
   int operations()const
   bool isNull()const
```

```
CCioKeyInfo()
```
Constructeur par défaut.

```
CCioKeyInfo(CBlob const &blob)
```
Crée et initialise un objet avec le blob fourni. Se référer à la norme [ISO 7816-15] pour une description complète.

```
CCioKeyInfo & operator= (CCioKeyInfo const &that)
```
Opérateur d'affectation.

```
CBlob parameters() const
```
Renvoie le détail des paramètres des algorithmes. Se référer à la norme [ISO 7816-15] pour une description complète.

```
Operations operations()const
```
Renvoie les opérations possibles avec la clé sous forme d'un champ de bits. Voir l'énumération **Operations** pour les masques possibles.

```
bool isNull()const
```
Renvoie TRUE quand cet objet a une valeur nulle.

5.2 - Objets fichiers

5.2.1 - CCiaFile

CCiaFile est la représentation d'un EF [ISO 7816-4]. Cette classe peut être utilisée pour récupérer n'importe quel contenu de fichier de la carte au format brut. Les sous classes spécialisées permettent de décomposer les principales structures de fichiers IAS définies (cf. **CApplicationDirectoryFile**, **CCiaInfoFile**, **CObjectDirectoryFile** et toutes les classes **CxxxFile**).

```
CCiaFile
   CCiaFile()
   CCiaFile(&card, CBlob const &aidCia, CCiaPath const &path
   void   open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
   const CFileControlParameters & fileControlParameters () const
   void   load (CBlob const &file)
   void load()
```

```
operator CBlob () const
CCiaPath const path() const
int size() const
```

```
CCiaFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CCiaFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
```
Permet de créer la représentation d'un fichier spécifié par *path*, relatif à l'application spécifiée.

```
void  open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
```
Cette fonction relie l'objet **CCiaFile** à un EF de la carte spécifié par *path*, relatif à l'application *aidCia* spécifiée.

```
void load(CBlob const &file)
```
Cette fonction permet de parser un buffer à la place du contenu du fichier.

```
void load()
```
L'appel de cette fonction permet de lire le contenu du fichier EF.

```
const CFileControlParameters & fileControlParameters () const
```
Renvoie les informations sur le fichier EF référencé par l'objet courant.

```
operator CBlob () const
```
Renvoie le contenu du fichier EF référencé par l'objet courant. Un appel à la fonction **load** doit être effectué avant d'appeler cette fonction.

```
CCiaPath const path () const
```
Renvoie le chemin du fichier EF référencé par l'objet courant.

```
int size () const
```
Renvoie la taille du fichier EF référencé par l'objet courant.

5.2.2 - CApplicationDirectoryFile

CApplicationDirectoryFile est la représentation du fichier EFDIR [ISO 7816-4]. Cette classe peut être employée pour récupérer la liste des applications déclarées.

```
CApplicationDirectoryFile : public CCiaFile
  CApplicationDirectoryFile ()
  CApplicationDirectoryFile (CCiaCard &card)
  CApplicationDirectoryFile (CCiaCard &card, CCiaPath const &path)
  void load (CBlob const &file)
  void load()
  operator CBlob () const
  std::vector< CApplicationTemplate > & applicationTemplates()
```

```
CApplicationDirectoryFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CApplicationDirectoryFile (CCiaCard &card)
```
Ce constructeur permet d'accéder au registre des applications (EFdir), avec l'ID standard 2F00.

```
CApplicationDirectoryFile (CCiaCard &card, CCiaPath const &path)
```

Ce constructeur permet d'accéder au registre des applications (EFdir), avec un ID non standard.

```
void load(CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format de fichier EFdir.

```
void load()
```
L'appel de cette fonction permet de lire le contenu du fichier EFdir.

```
operator CBlob () const
```
Renvoie le contenu sur le fichier EFdir associé à l'objet courant. Un appel à la fonction **load** doit être fait préalablement à l'appel de cette fonction.

```
std::vector< CApplicationTemplate > & applicationTemplates()
```
L'application appelante peut appeler cette méthode afin d'obtenir un vecteur de **CApplicationTemplate**, chaque élément étant une représentation d'une application (ou un enregistrement du fichier EFdir). Un appel à la fonction **load** doit être fait préalablement à l'appel de cette fonction.

5.2.3 - CCiaInfoFile

CCiaInfoFile est une représentation d'**EFCiaInfo**. Cette classe peut être utilisée afin d'obtenir les attributs d'une application spécifiée. Se référer à la norme [ISO 7816-15] *CiaInfo* pour une description complète.

```
CciaInfoFile : public CCiaFile
  CCiaInfoFile ()
  CCiaInfoFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  int version ()
  CBlob serialNumber ()
  CBlob manufacturerID ()
  CBlob label ()
  int cardflags ()
  std::vector< SecurityEnvironmentInfo >  seInfo ()
  std::vector< std::pair< int, int > >  recordInfo () const
  std::vector< AlgorithmInfo >   supportedAlgorithms () const
  CBlob  issuerId () const
  CBlob  holderId () const
  std::string  lastUpdate () const
  CBlob  preferredLanguage () const
  std::vector< std::pair< int, std::string > >   profileIndication ()
```

```
CCiaInfoFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CCiaInfoFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
```
Ce constructeur permet d'accéder à la représentation du fichier EF**CiaInfo** spécifié par *path*, relatif à l'application spécifiée.

```
void  load (CBlob const &fileContent)
```
Cette fonction permet de parser un buffer conforme avec le format **EFciaInfo**.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'**EFciaInfo**.

```
operator CBlob () const
```

Retourne le contenu brut de l'**EFciaInfo** associé à l'objet **CCiaInfoFile**. Un appel à la fonction **load** doit être fait préalablement à l'appel de cette fonction.

```
int version ()
```
Retourne la version de la spécification CIA (cf. [ISO 7816-15]).

```
CBlob serialNumber ()
```
Renvoie le numéro de série de l'application (cf. [ISO 7816-15]).

```
CBlob manufacturerID ()
```
Renvoie l'ID du fabricant de la carte (cf. [ISO 7816-15]).

```
CBlob label ()
```
Renvoie le label de l'application, identifiant l'application (cf. [ISO 7816-15]).

```
int cardflags ()
```
Retourne les capacités de la carte (cf. **[ISO 7816-15]**)

```
std::vector< SecurityEnvironmentInfo >   seInfo ()
```
Renvoie un vecteur d'informations sur les environnements de sécurité applicables à l'application courante.

```
std::vector< std::pair< int, int > >   recordInfo () const
```
Renvoie le type de certains EF : enregistrement linéaire ou fichier transparent. (cf. [ISO 7816-4])
Cette information est fournie sous la forme d'un vecteur de paires : contextTag, recordLength.
L'énumération **ContextTag** fourni les valeurs possibles pour *contextTag*, afin d'identifier le type d'objet dont la structure est linéaire, avec *recordLength* fournissant la longueur d'enregistrement associée.
Se référer à la norme [ISO 7816-15] pour de plus amples informations.

```
std::vector< AlgorithmInfo >  supportedAlgorithms () const
```
Renvoie un vecteur d'algorithmes applicables à l'application courante.

```
CBlob   issuerId () const
```
Retourne l'ID de l'émetteur de la carte. (cf. **[ISO 7816-15]**)

```
CBlob   holderId () const
```
Retourne l'ID du porteur de la carte. (cf. **[ISO 7816-15]**)

```
std::string  lastUpdate () const
```
Retourne la date de la dernière mise à jour de l'application. (cf. **[ISO 7816-15]**)

```
CBlob   preferredLanguage () const
```
Renvoie la langue utilisée par le porteur de la carte. (cf. **[ISO 7816-15]**)

```
std::vector< std::pair< int, std::string >  > profileIndication ()
```
Retourne la liste des profils avec lesquels la carte est compatible. (cf. **[ISO 7816-15]**)
Cette information est fournie sous la forme d'un vecteur de paires : profileType, profileName. ProfileType peut être 0 pour indiquer que le nom du profil est fourni sous la forme d'un OID, ou 1 comme un nom de profil.
Se référer à la norme [ISO 7816-15] pour de plus amples informations.

```
void  open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
```
Cette fonction rattache le **CCiaInfoFile** à un EF spécifié par *path*, relatif à l'application spécifiée *aidCia*.

```
const CFileControlParameters & fileControlParameters () const
```
Renvoie les informations sur l'**EF** référencé par le **CCiaInfoFile** courant.

5.2.4 - CObjectDirectoryFile

CObjectDirectoryFile est une représentation de l'**EFod**. Cette classe peut être utilisée pour obtenir les chemins relatifs (à partir de l'application CIA racine) des fichiers supportés par l'application et leurs classes associées. Se référer à la norme [ISO 7816-15] *EFod* pour une description complète.

```
CobjectDirectoryFile : public CCiaFile
      CObjectDirectoryFile ()
      CObjectDirectoryFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const
&path)
      void  load (CBlob const &file)
      void  load ()
      operator CBlob () const
      std::vector< std::pair< CioChoice, CCiaPath > > &cioFiles ()
```

```
CObjectDirectoryFile ()
```

Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CObjectDirectoryFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const
&path)
```

Ce constructeur permet de créer une représentation du fichier répertoire spécifié par *path*, relatif à l'application spécifiée.

```
void  load (CBlob const &fileContent)
```

Cette fonction permet de parser un buffer conforme avec le format **EFod**.

```
void  load ()
```

Cette fonction demande la lecture du contenu de l'**EFod**.

```
operator CBlob () const
```

Retourne le contenu d'EF associé à l'objet **CObjectDirectoryFile**. Un appel à la fonction **load** doit être effectué préalablement à l'appel de cette fonction.

```
std::vector< std::pair< CioChoice, CCiaPath > > &cioFiles ()
```

Renvoie un vecteur de paires listant tous les fichiers supportés par l'application. Chaque paire décrit un fichier et est constituée d'un indicateur de classe et du chemin du fichier, relatif à l'application racine.

5.2.5 - CAuthObjectFile

CAuthObjectFile est la représentation du fichier contenant les objets d'authentification. Se référer à la norme [ISO 7816-15] *AuthObject* pour une description complète.

```
CAuthObjectFile : public CCiaFile
  CAuthObjectFile()
  CAuthObjectFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  std::vector< CCioAuthentication * > & objects ()
```

```
CAuthObjectFile ()
```

Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CAuthObjectFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Employez ce constructeur pour instancier un objet associé au fichier identifié par le chemin *path* dans le contexte de l'application *aid.*

```
void  load (CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format **EFod**.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'EF référencé par path (provenant de l'objet dans le constructeur).

```
operator CBlob () const
```
Utilisez cette fonction pour obtenir le contenu brut du fichier.

```
std::vector< CCioAuthentication * > &  objects ()
```
Retourne des objets dont la classe de base est **CCioAuthentication**, dont hérite **CCioPasswordAuth**. Utilisez cette fonction pour obtenir le contenu du fichier parsé sous la forme d'une liste d'objets **CCioAuthentication.** Les applications devraient utiliser **CCioAuthentication::choice()** pour déterminer la classe d'objet dérivée exacte.

5.2.6 - CCertificateFile

CCertificateFile est une représentation du fichier contenant les certificats au format X509 (les certificats PRIS V2 sont au format X509). Se référer à la norme [ISO 7816-15] *Certificates* pour une description complète.

```
CCertificateFile : public CCiaFile
  CCertificateFile ()
  CCertificateFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  std::vector< CCioCertificate * > & objects ()
```

```
CCertificateFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CCertificateFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```
Employez ce constructeur pour instancier un objet associé au fichier identifié par le chemin *path* dans le contexte de l'application *aid.*

```
void  load (CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format de l'EF.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'EF référencé par *path* (provenant de l'objet dans le constructeur).

```
operator CBlob () const
```
Utilisez cette fonction pour obtenir le contenu brut du fichier.

```
std::vector< CCioCertificate * > &   objects ()
```
Retourne des objets dont la classe de base est **CCioCertificate**, dont hérite **CCioX509Certificate**. Utilisez cette fonction pour obtenir le contenu du fichier parsé sous la forme d'une liste d'objets **CCioCertificate**. Les applications devraient utiliser **CCioCertificate::choice()** pour déterminer la classe d'objet dérivée exacte.

5.2.7 - CDataContainerFile

CDataContainerFile est une représentation d'un fichier contenant des données. Se référer à la norme [ISO 7816-15] *DataContainerObjects* pour une description complète.

```
CDataContainerFile : public CCiaFile
  CDataContainerFile ()
  CDataContainerFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  std::vector< CCioDataContainer * > & objects ()
```

```
CDataContainerFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CDataContainerFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```
Employez ce constructeur pour instancier un objet associé à l'identifiant du fichier avec le chemin *path* dans le contexte de l'application *aid.*

```
void  load (CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format de l'EF.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'EF référencé par *path* (provenant de l'objet dans le constructeur).

```
operator CBlob () const
```
Utilisez cette fonction pour obtenir le contenu brut du fichier.

```
std::vector< CCioDataContainer * > &  objects ()
```
Retourne des objets dont la classe de base est **CCioDataContainer**, dont hérite **CCioOpaqueDO**. Utilisez cette fonction pour obtenir le contenu du fichier parsé sous la forme d'une liste d'objets **CCioDataContainer**. Les applications devraient utiliser **CCioDataContainer::choice()** pour déterminer la classe d'objet dérivée exacte.

5.2.8 - CPrivateKeyFile

CPrivateKeyFile est une représentation d'un fichier contenant des clés privées. Se référer à la norme [ISO 7816-15] *PrivateKeys* pour une description complète.

```
CPrivateKeyFile : public CCiaFile
  CPrivateKeyFile ()
  CPrivateKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  std::vector< CCioPrivateKey * > & objects ()
```

```
CPrivateKeyFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CPrivateKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Employez ce constructeur pour instancier un objet associé au fichier identifié par le chemin *path* dans le contexte de l'application *aid.*

```
void  load (CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format de l'EF.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'EF référencé par *path* (provenant de l'objet dans le constructeur).

```
operator CBlob () const
```
Utilisez cette fonction pour obtenir le contenu brut du fichier.

```
std::vector< CCioPrivateKey * > &   objects ()
```
Retourne des objets dont la classe de base est **CCioPrivateKey**, dont héritent **CCioPrivateDHKey** et **CCioPrivateRSAKey**. Utilisez cette fonction pour obtenir le contenu du fichier parsé sous la forme d'une liste d'objets **CCioPrivateKey**. Les applications devraient utiliser **CCioPrivateKey::choice()** pour déterminer la classe d'objet dérivée exacte.

5.2.9 - CPublicKeyFile

CPublicKeyFile est une représentation du fichier contenant les clés publiques. Se référer à la norme [ISO 7816-15] *PublicKeys* pour une description complète.

```
CPublicKeyFile : public CCiaFile
  CPublicKeyFile ()
  CPublicKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
  void   load (CBlob const &file)
  void   load ()
  operator CBlob () const
  std::vector< CCioPublicKey * > & objects ()
```

```
CPublicKeyFile ()
```
Constructeur par défaut. Utilisez **CCiaFile::open** pour rattacher cet objet à un fichier de la carte.

```
CPublicKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```
Employez ce constructeur pour instancier un objet associé au fichier identifié par le chemin *path* dans le contexte de l'application *aid.*

```
void  load (CBlob const &file)
```
Cette fonction permet de parser un buffer conforme avec le format de l'EF.

```
void  load ()
```
Cette fonction demande la lecture du contenu de l'EF référencé par *Path* (provenant de l'objet dans le constructeur).

```
operator CBlob () const
```
Utilisez cette fonction pour obtenir le contenu brut du fichier.

```
std::vector< CCioPublicKey * > &  objects ()
```
Retourne des objets dont la classe de base est **CCioPublicKey**, dont héritent **CCioPublicDHKey** et **CCioPublicRSAKey**. Utilisez cette fonction pour obtenir le contenu du fichier parsé sous la forme d'une liste d'objets **CCioPublicKey**. Les applications devraient utiliser **CCioPublicKey::choice()** pour déterminer la classe d'objet dérivée exacte.

5.3 - Objets de données

5.3.1 - CApplicationTemplate

CApplicationTemplate représente une application [ISO 7816-4], ou un enregistrement d'EFDIR. Cette classe peut être utilisée pour obtenir les attributs des applications enregistrées. Se référer à la norme [ISO 7816-4] *Application template* pour une description complète.

```
CApplicationTemplate
  CApplicationTemplate ()
  CApplicationTemplate (CBlob const &blob)
  CApplicationTemplate(CApplicationTemplate const & that)
  CApplicationTemplate & operator= (CApplicationTemplate const &that)
  CBlob aid()
  CBlob path()
  CBlob label()
  CBlob discretionaryDOBerTlv() const
  CBlob discretionaryDONonBerTlv() const
```

`CApplicationTemplate ()`

Constructeur par défaut.

`CApplicationTemplate (CBlob const &blob)`

Ce constructeur permet de parser un buffer conforme avec le format d'enregistrement **EFdir**.

`CApplicationTemplate (CApplicationTemplate const & that)`

Constructeur par copie.

`CApplicationTemplate & operator= (CApplicationTemplate const &that)`

Opérateur d'assignation.

`CBlob aid()`

Fournit l'identifiant enregistré pour l'application.

`CBlob path()`

Fournit le chemin de l'application enregistrée.

`CBlob label()`

Fournit le label de l'application enregistrée.

`CBlob discretionaryDOBerTlv() const`

Use this method to get application inter industry data element.

`CBlob discretionaryDONonBerTlv() const`

Use this method to get application inter industry data element.

5.3.2 - CCioObject

CCioObject est une classe abstraite pour les objets Cio. Se référer à la norme [ISO 7816-15] *CommonObjectAttributes* pour une description complète.

```
CCioObject
  CBlob label () const
      CommonObjectFlags flags () const
      CBlob authIdReference () const
      int userConsent () const
      std::vector< CCioAccessControlRule > accessControlRules () const
```

`CBlob label () const`

Fournit le label associé à l'objet.

`CommonObjectFlags flags () const`

Renvoie les informations d'accès à propos de l'objet.

`CBlob authIdReference () const`

Retourne l'ID d'authentification associé à l'objet.

`int userConsent () const`

Indique si l'authentification est nécessaire avant l'exécution d'une signature.

`std::vector< CCioAccessControlRule > accessControlRules () const`

Renvoie les informations sur le mode d'accès et les mécanismes de sécurité associés.

Note : Cette classe ne peut pas être instanciée.

5.3.3 - CCioAuthentication

CCioAuthentication est une classe abstraite pour représenter les objets d'authentification. Se référer à la norme [ISO 7816-15] *CommonAuthenticationObjectAttributes* pour une description complète.

```
CCioAuthentication : public CCioObject
  virtual int choice() const=0
  CBlob authId () const
  int authReference () const
  int seIdentifier () const
```

`virtual int choice() const`

Méthode virtuelle abstraite afin d'être implémentée par des sousclasses. Permet aux applications de déterminer quel type d'objet est renvoyé par la méthode **CCioAuthObjectFile::objects().** Se référer à [ISO 7816-15] *AuthenticationObjectChoice* pour les différentes valeurs possibles (*untagged value -1*).

`CBlob authId () const`

Renvoie l'identifiant de l'authentification. Se référer à la norme [ISO 7816-15] pour une description complète.

`int authReference () const`

Renvoie la référence de l'authentification. Se référer à la norme [ISO 7816-15] pour une description complète.

`int seIdentifier () const`

Renvoie la référence de l'environnement de sécurité. Se référer à la norme [ISO 7816-15] pour une description complète.

Note : Cette classe ne peut pas être instanciée.

5.3.4 - CCioPasswordAuth

CCioPasswordAuth est une classe pour représenter l'objet mot de passe d'authentification. Se référer à la norme [ISO 7816-15] *PasswordAttributes* pour une description complète.

```
CCioPasswordAuth : public CCioAuthentication
  CCioPasswordAuth ()
  CCioPasswordAuth (CBlob const &blob)
  int choice() const int  pwdFlags () const
  PasswordType  pwdType () const
  int  minLength () const
  int  storedLength () const
  int  maxLength () const
  int  pwdReference () const
  unsigned char padChar () const
  bool   padCharDefined () const
  std::string   lastPasswordChange () const
  CCiaPath  path () const
```

```
CCioPasswordAuth ()
```
Constructeur par défaut.

```
CCioPasswordAuth (CBlob const &blob)
```
Constructeur avec initialisation à partir d'un buffer conforme avec le contenu du fichier EF**.**

```
int choice () const
```
Méthode abstraite de classe de base. Permet aux applications de déterminer si un objet **CCioAuthentication** est une instance de **CCioPasswordAuth.** Se référer à [ISO 7816-15] *AuthenticationObjectChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
int   pwdFlags () const
```
Renvoie le flag indiquant comment un mot de passe peut être manipulé. Se référer à la norme [ISO 7816-15] *PasswordFlags* pour une description complète.

```
PasswordType   pwdType () const
```
Renvoie le type de mot de passe. Se référer à la norme [ISO 7816-15] *PasswordType* pour une description complète.

```
int   minLength () const
```
Retourne la taille minimum du nouveau mot de passe (Si on a la possibilité de le changer).

```
int   storedLength () const
```
Renvoie la taille du mot de passe actuel.

```
int   maxLength () const
```
Retourne la taille maximum du nouveau mot de passe (Si on a la possibilité de le changer).

```
int   pwdReference () const
```
Renvoie la référence du mot de passe, équivalent au paramètre *p2* de la commande **CIASCard::verify**. Se référer à la norme [ISO 7816-15] *PasswordReference* pour une description complète.

```
unsigned char  padChar () const
```
Renvoie le caractère de padding à utiliser pour padder un mot de passe, si *pwdFlags* est renseigné. Se référer à la norme [ISO 7816-15] *padChar* pour une description complète.

```
bool  padCharDefined () const
```
Renvoie TRUE si l'attribut *pwdFlags* est mis à *needs-padding*, FALSE sinon.

```
std::string  lastPasswordChange () const
```
Renvoie la date du dernier changement de mot de passe.

```
CCiaPath  path () const
```
Renvoie le chemin du DF dans lequel le mot de passe est situé.

5.3.5 - CCioCertificate

CCioCertificate est une classe abstraite pour représenter un objet certificat au format X509 (les certificats PRIS V2 sont au format X509). Se référer à la norme [ISO 7816-15] *CommonCertificateAttributes* pour une description complète.

```
CCioCertificate : public CCioObject
  virtual int choice() const=0
  CBlob id () const
  bool authority ()
  CBlob certHash ()
  CCioUsage trustedUsage ()
  std::vector< CCioCredentialIdentifier > identifiers ()
  Validity validity ()
```

```
virtual int choice() const=0
```
Méthode virtuelle abstraite afin d'être implémentée par des sousclasses. Permet aux applications de déterminer quel type d'objet est renvoyé par la méthode **CCioCertificateFile::objects().** Se référer à [ISO 7816-15] *CertificateChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CBlob id () const
```
Renvoie l'id du certificat, commun aux clés publiques et privées liées au certificat.

```
bool authority ()
```
Renvoie TRUE si le certificat est celui d'une autorité.

```
CBlob certHash ()
```
Renvoie le hash du certificat.

```
CCioUsage trustedUsage ()
```
Indique pour quel usage le détenteur de la carte peut faire confiance au certificat.

```
std::vector< CCioCredentialIdentifier > identifiers ()
```
Fournit les identifiants (ou alias) à partir desquels le certificat peut être trouvé ou référencé.

```
Validity validity ()
```
Fournit les informations sur la période de validité du certificat.

5.3.6 - CCioX509Certificate

CCioX509Certificate est une classe pour représenter un objet certificat X509 (les certificats PRIS V2 sont au format X509V3). Se référer à la norme [ISO 7816-15] *X509CertificateAttributes* pour une description complète.

```
CCioX509Certificate : public CCioCertificate
   int choice() const
   CCioX509Certificate()
   CCioX509Certificate(CBlob const &blob)
   CCioObjectValue value () const
   CBlob subject () const
   CBlob issuer () const
   CBlob serialNumber () const
```

`int choice() const`

Méthode abstraite de classe de base. Permet aux applications de déterminer si un objet **CCioCertificate** est une instance de **CCioX509Certificate.** Se référer à [ISO 7816-15] *CertificateChoice* pour les différentes valeurs possibles (*untagged value -1*).

`CCioX509Certificate()`

Constructeur par défaut.

`CCioX509Certificate(CBlob const &blob)`

Constructeur avec initialisation de l'objet à partir d'un buffer conforme avec le format du contenu du fichier EF.

`CCioObjectValue value () const`

Retourne la valeur du certificat

`CBlob subject () const`

Renvoie le champ *subject* du certificat.

`CBlob issuer () const`

Renvoie le champ *issuer* du certificat.

`CBlob serialNumber () const`

Renvoie le champ *serial number* du certificat.

5.3.7 - CCioDataContainer

CCioDataContainer est une classe abstraite pour la représentation d'un objet de données. Se référer à la norme [ISO 7816-15] *CommonDataContainerObjectAttributes* pour une description complète.

```
CCioDataContainer : public CCioObject
   virtual int choice() const
   CBlob applicationName () const
   std::string applicationOID () const
   CBlob id () const
```

`virtual int choice() const`

Méthode virtuelle abstraite afin d'être implémentée par des sousclasses. Permet aux applications de déterminer quel type d'objet est renvoyé par la méthode **CDataContainer File::objects().** Se référer à [ISO 7816-15] *DataContainerObjectChoice* pour les différentes valeurs possibles (*untagged value -1*).

`CBlob applicationName () const`

Renvoie le nom de l'application à laquelle le conteneur de données appartient.

```
std::string applicationOID () const
```
Renvoie l'OID de l'application à laquelle le conteneur de données appartient.

```
CBlob id () const
```
Renvoie l'unique identifiant de l'objet.

5.3.8 - CCioOpaqueDO

CCioOpaqueDO est une classe pour représenter un objet opaque de la carte. Se référer à la norme [ISO 7816-15] *OpaqueDOAttributes* pour une description complète.

```
CCioOpaqueDO : public CcioDataContainer CCioOpaqueDO()
   CCioOpaqueDO()
   CCioOpaqueDO(CBlob const &blob)
   int choice() constCCioObjectValue value() const
   CCioObjectValue value() const
```

```
CCioOpaqueDO()
```
Constructeur par défaut.

```
CCioOpaqueDO(CBlob const &blob)
```
Constructeur avec initialisation de l'objet à partir d'un buffer conforme avec le format du contenu du fichier EF.

```
int choice() const
```
Méthode abstraite de classe de base. Permet aux applications de déterminer si un objet **CCioDataContainer** est une instance de **CCioOpaqueDO.** Se référer à [ISO 7816-15] *DataContainerObjectChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CCioObjectValue value() const
```
Renvoie la valeur du conteneur de données.

5.3.9 - CCioKey

CCioKey est une classe abstraite pour représenter un objet clé. Se référer à la norme [ISO 7816-15] *CommonKeyAttributes* pour une description complète.

```
CCioKey : CCioObject
   CBlob id () const
   int usage () const
   bool native ()
   int accessFlags ()
   int accessFlagsDefined ()
   int keyReference () const
   std::string startDate () const
   std::string endDate () const
   std::vector< int > algReference ()
```

```
CBlob id () const
```
Renvoie l'identifiant de la clé, commune à la clé privée et aux certificats.

```
int usage () const
```
Renvoie les différents usages possibles de la clé. Se référer à la norme [ISO 7816-15] *KeyUsageFlags* pour une description complète.

```
bool native ()
```
Renvoie TRUE si les algorithmes cryptographiques associés à la clé sont implémentés directement sur la carte.

```
int accessFlags ()
```
Renvoie les flags d'accès à la clé. Se référer à la norme [ISO 7816-15] *KeyAccessFlags* pour une description complète.

```
int accessFlagsDefined ()
```
Renvoie TRUE si les flags d'accès sont spécifiés pour la clé courante, FALSE sinon.

```
int keyReference () const
```
Renvoie une référence de clé spécifique à la carte. Se référer à la norme [ISO 7816-4] pour une description complète.

```
std::string startDate () const
```
Renvoie la date à partir de laquelle la clé est valide.

```
std::string endDate () const
```
Renvoie la date jusqu'à laquelle la clé est valide.

```
std::vector< int > algReference ()
```
Renvoie les algorithmes avec lesquels la clé peut être utilisée.

Note : **CCioKey** est la classe de base dont héritent **CCioPrivateKey CCioPublicKey** et **CCioSecretKey**.

5.3.10 - CCioPrivateKey

CCioPrivateKey est une classe abstraite pour représenter un objet clé privée. Se référer à la norme [ISO 7816-15] *CommonPrivateKeyAttributes* pour une description complète.

```
CCioPrivateKey : public CCioKey
  virtual int choice() const=0
  CBlob name ()
  std::vector< CCioCredentialIdentifier > keyIdentifiers ()
  CBlob generalName () const
```

```
virtual int choice() const
```
Méthode virtuelle abstraite afin d'être implémentée par des sousclasses. Permet aux applications de déterminer quel type d'objet est renvoyé par la méthode **CCioPrivateKeyFile File::objects().** Se référer à [ISO 7816-15] *PrivateKeyChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CBlob name ()
```
Renvoie le nom du possesseur de la clé, comme spécifié dans le champ *subject* du certificat correspondant.

```
std::vector< CCioCredentialIdentifier > keyIdentifiers ()
```
Fournit les identifiants (ou alias) à partir desquels la clé peut être trouvée ou référencée.

```
CBlob generalName () const
```
Renvoie le nom du propriétaire de la clé privée.

5.3.11 - CCioPrivateRSAKey

CCioPrivateRSAKey est une classe pour représenter un objet clé privée RSA. Se référer à la norme [ISO 7816-15] *PrivateRSAKeyAttributes* pour une description complète.

```
CCioPrivateRSAKey : public CCioPrivateKey
   CCioPrivateRSAKey()
   CCioPrivateRSAKey(CBlob const &blob)
   int choice() const
   CCiaPath  value()
   CCioKeyInfo keyInfo ()
   int modulusLength ()
```

```
CCioPrivateRSAKey()
```

Constructeur par défaut.

```
CCioPrivateRSAKey(CBlob const &blob)
```

Constructeur avec initialisation de l'objet à partir d'un buffer conforme avec le format du contenu du fichier EF.

```
int choice() const
```

Méthode abstraite de classe de base. Permet aux applications de déterminer si un objet **CCioPrivateKey** est une instance de **CCioPrivateRSAKey.** Se référer à [ISO 7816-15] *PrivateKeyChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CCiaPath  value()
```

Renvoie le chemin de la clé privée RSA.

```
CCioKeyInfo keyInfo ()
```

Renvoie les informations sur la clé privée.

```
int modulusLength ()
```

Renvoie la longueur de la clé en bits.

5.3.12 - CCioPublicKey

CCioPublicKey est une classe abstraite pour représenter un objet clé publique. Se référer à la norme [ISO 7816-15] *CommonPublicKeyAttributes* pour une description complète.

```
CCioPublicKey : public CCioKey
   virtual int choice() const
   CBlob name ()
   CCioUsage trustedUsage () const
   std::vector< CCioCredentialIdentifier > keyIdentifiers () const
   CBlob generalName () const
```

```
virtual int choice() const
```

Méthode virtuelle abstraite afin d'être implémentée par des sousclasses. Permet aux applications de déterminer quel type d'objet est renvoyé par la méthode **CCioPublicKeyFile File::objects().** Se référer à [ISO 7816-15] *PublicKeyChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CBlob name ()
```

Renvoie le nom du possesseur de la clé, comme spécifié dans le champ *subject* du certificat correspondant.

```
CCioUsage  trustedUsage () const
```

Indique pour quel usage le détenteur de la carte peut faire confiance à la clé.

```
std::vector< CCioCredentialIdentifier > keyIdentifiers () const
```

Fournit les identifiants (ou alias) à partir desquels la clé peut être trouvée ou référencée.

```
CBlob generalName () const
```

Renvoie le nom du propriétaire de la clé publique.

5.3.13 - CCioPublicRSAKey

CCioPublicRSAKey est une classe pour représenter un objet clé publique RSA. Se référer à la norme [ISO 7816-15] *PublicRSAKeyAttributes* pour une description complète.

```
CCioPublicRSAKey : public CCioPublicKey
   CCioPublicRSAKey ()
   CCioPublicRSAKey (CBlob const &blob)
   int choice() const
   CCioObjectValue  value()
   int modulusLength ()
   CCioKeyInfo keyInfo ()
```

```
CCioPublicRSAKey ()
```

Constructeur par défaut.

```
CCioPublicRSAKey (CBlob const &blob)
```

Constructeur avec initialisation de l'objet à partir d'un buffer conforme avec le format du contenu du fichier **EF**.

```
int choice() const
```

Méthode abstraite de classe de base. Permet aux applications de déterminer si un objet **CCioPublicKey** est une instance de **CCioPublicRSAKey.** Se référer à [ISO 7816-15] *PublicKeyChoice* pour les différentes valeurs possibles (*untagged value -1*).

```
CCioObjectValue  value()
```
Renvoie le chemin de la clé publique RSA.

```
int  modulusLength ()
```
Renvoie la taille de la clé en bits.

```
CCioKeyInfo keyInfo ()
```
Renvoie les informations sur la clé publique.

6 - Macros

6.1 - _HB : HexaBlob convertion

Cette macro permet de créer un **CBlob** à partir d'un buffer d'octets contenant une chaîne de caractères hexadécimaux. Cette chaîne hexadécimale doit être constituée de couple de digits, optionnellement séparés par des espaces.

```
CBlob _HB(char const * hex)
CBlob _HB(std::string  hex)
```

Permet de créer un **CBlob** à partir d'une chaîne de caractères contenant des couples de digits hexadécimaux.

```
std::string _HB(CBlob const &blob)
```

Permet de convertir le contenu d'un **CBlob** en une chaîne de caractères contenant des couples de digits hexadécimaux.

Exemple :

```
// from string to CBlob
CBlob myBlob = _HB("2F 00");

// from CBlob to std::string
Std::string s = _HB(myBlob);
```

6.2 - AsString

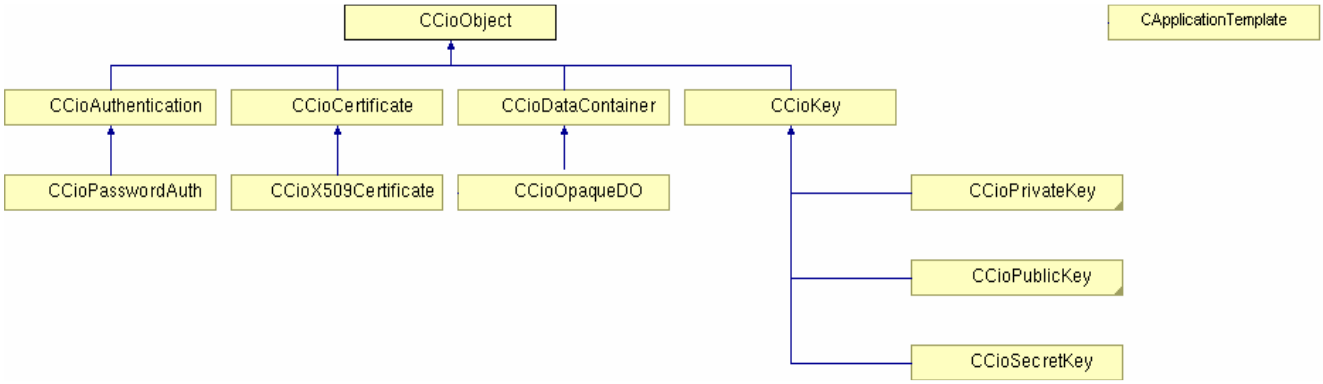
Cette macro permet d'obtenir la représentation d'un **CBlob** en std ::string. Cette macro est typiquement utilisée pour afficher un **CBlob** contenant des données affichables.

```
Std::string AsString(CBlob const &blob)
```

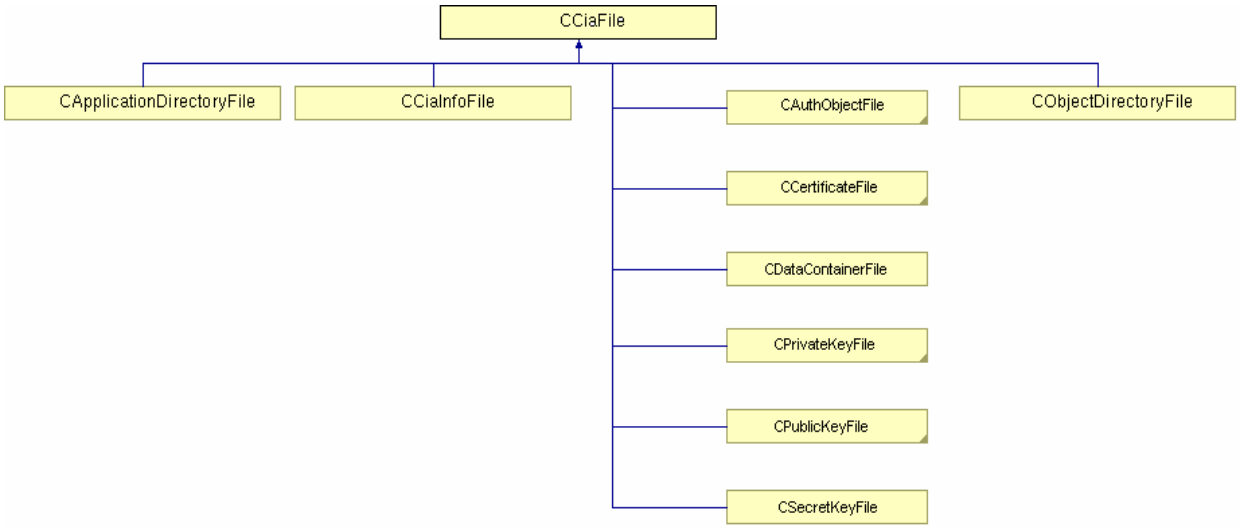
7 - Annexes

7.1 - Diagrammes de classes

7.1.1 - Objets



7.1.2 - Fichiers



7.2 - Exemple d'utilisation

Ci-dessous un exemple d'utilisation de l'API IAS, il s'agit d'un exemple de connexion au premier lecteur de carte, et de recherche des applications CIA dans le fichier EFdir.
Pour chaque application, l'exemple affiche son label et parcourt tous les objets référencés par EFod. Si un conteneur de certificat est trouvé, le programme affiche le label de chaque certificat présent dans le conteneur.

```
IASRESULT result;

// Initialize IAS API
IASCONTEXT iasContext;
result = IASInitialize(&iasContext);

// Get reader list
char mszReaders[1000];
DWORD cchReaders=sizeof(mszReaders);
```

```
result = IASListReaders(iasContext, mszReaders,&cchReaders);

// Try to connect to a card present in first reader
CIASCard *pIASCard;
result= IASConnect(mszReader,&pIASCard);

// Read Application Directory content from card (EF 2F00)
CCiaPath dirPath;
dirPath.setEfidOrPath(_HB("2F 00"));

CApplicationDirectoryFile efDIR(pIASCard, dirPath);
efDIR.load();

// search for a CIA application in application list
const CBlob ridCia(_HB("E8 28 BD 08 0F"));
std::vector< CApplicationTemplate> vat(efDIR.applicationTemplates());

for(std::vector< CApplicationTemplate>::const_iterator iat = vat.begin(); iat !=
     vat.end(); iat++)
   {
   // CIA application
   if(iat->aid().size()>=ridCia.size()                    &&                    iat-
>aid().substr(0,ridCia.size())==ridCia)
     {
     CCiaPath path;
     // read CIAInfo and display application label
     path.setEfidOrPath(_HB("50 32"));
     CCiaInfoFile ciaInfoFile (pIASCard, iat->aid(),path);
     ciaInfoFile.load();

     cout << "CIA Application Label : ";
     cout << _HB(ciaInfoFile.label());
     cout << "]\n";

     // read CIAObjectDirectory
     path.setEfidOrPath(_HB("50 31"));
     CObjectDirectoryFile odFile (pIASCard, iat->aid(),path);

     // walk through object list
     std::vector< std::pair< CioChoice, CCiaPath > > &objects = odFile.cioFiles();
     std::vector< std::pair< CioChoice, CCiaPath > >::const_iterator obj;

     for(obj= objects.begin(); obj != objects.end(); obj++)
       {
       // construct object path
       CCiaPath pathToObjectContainer(obj->second);

       // check object type
       switch ( obj->first)
           {
          // this is a certificate container
           case CioChoice::ctCertificates:
             {
             // build matching object container file
             CCertificateFile certFile(pIASCard, iat->aid(),
   pathToObjectContainer);
             certFile.load();

             // get all the certificates from this container
             std::vector<CCioCertificate *> certs = certFile.objects();
             std::vector<CCioCertificate *>::const_iterator cert;
```

```
            for( cert= certs.begin(); cert!= certs.end(); cert++)
              {
              // display certificate label
              cout << "cert label : ";
              cout << _HB((*cert)->label());
              cout << "]\n";
              }
            }
            break;

          // other container types
           default:
            break;
          }
        }
     }
         }

// …/…

IASFinalize(iasContext);
```

Note : Les options de compilation requises sont :
Runtime Library = Multi-threaded Debug DLL (/MDd)

7.2.1 - Utilisation du canal transparent

Le code suivant illustre l'utilisation d'un canal transparent pour vérifier le pin global de la carte IAS. Se référer à [ISO 7816-4] pour le détail des commandes APDU, et [IAS] pour l'ensemble des commandes supportées par la carte IAS.

```
  //…/…
  // CIASCard::sendAPDU usage (verify PIN number 1 = 00000000)

  CBlob PIN("00000000");
  CBlob response;
  unsigned short sw12;

  IASCard.sendAPDU(  0,     // classe
             0x20,          // instruction (verify)
             0,             // p1
             1,             // p2 (object number)
             PIN,           // données en entrée (pin)
             256,           // get all data if any
             response,      // données en sortie
             sw12);         // code d'erreur : 9000 = SUCCESS

  cout << "Response : " << _HB(response) << endl;
  cout << "SW       : " << hex << setfill('0') << setw(4) << sw12 << endl;
  //…/…
```