Real Casa de la Moneda
Fábrica Nacional
de Moneda y Timbre

DIRECCIÓN COMERCIAL
DEPARTAMENTO COMERCIAL TARJETAS

CONTENIDO ACTIVEX PARA DNI-E

	NOMBRE	FECHA
Elaborado por:	Juan Carlos Pérez García	20/04/2017
Revisado por:		
Aprobado por:		

HISTÓRICO DEL DOCUMENTO			
Versión	Fecha	**Descripción**	**Autor**
1	24/05/2017	Creación del documento	Juan Calros Pérez García

Documento clasificado como: *Confidencial*

Índice:

MANUAL DE USO DE LA LIBRERÍA ActivexDNIe

Introducción.

Con este software se persiguen dos objetivos:
Dar acceso a las nuevas funcionalidades de la tarjeta DNIe que no están soportadas por el Drivers PKCS#11.

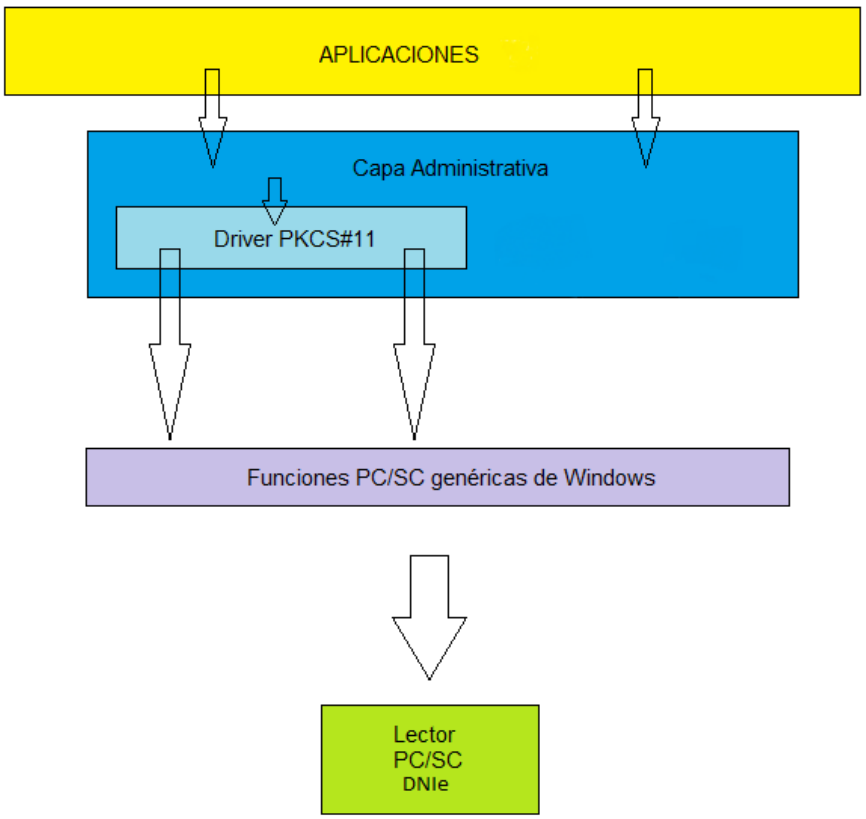
Permitir el uso de las funcionalidades habituales de la tarjeta desde cualquier software sin necesidad de ningún conocimiento específico sobre este tema.

Características principales:

Se realiza una gestión transparente de los lectores de tarjetas, los slots, etc.

Se enmascara el uso de sesiones de trabajo.

Su estructura de capas se ve en el siguiente gráfico:



Vemos que parte de los accesos se hacen de forma convencional, a través del driver y el resto de funciones extendidas, a través del envío de comandos a bajo nivel.

Objeto Administrativo:

Esta librería va a estar formada por un único objeto administrativo, con el que podremos realizar operaciones básicas con la tarjeta.

Objeto	Descripción
DLL_Admin	Proporciona las propiedades y métodos necesario para el manejo de la tarjeta criptográfica DNIe.

DLL_ADMIN

El objeto DLL_Admin suministra propiedades y métodos que nos van a permitir un fácil manejo de la librería PKCS#11.

Características del objeto

Id = "DLL_ADMINISTRATIVA_DNIE"

Classid = "clsid : 0A39EB06-E3DB-49B6-8C8D-D2EEE5335B20"

codebase = "ActiveXDNIe.dll#version=1.0.0.1">

Métodos

El objeto DLL_Admin tiene definidos los siguientes métodos que los vamos a dividir por su funcionalidad:

Gestión de Pin:

Métodos	Descripción
PresentarPin	Realiza una presentación de pin. Se inicializará previamente la propiedad **PinIn**.

Gestión de conexión:

Métodos	Descripción
InicializarConexion	Se va a encargar de cargar las librerías necesarias para utilizar el activeX. Otra de las funciones será detectar los lectores instalados, ve cuáles tienen tarjeta introducida, etc.
FinalizarConexion	Libera los recursos y tira cualquier canal establecido.
ObtenerTipoDocumento	Obtiene que documento se encuentra en el lector. El valor se encontrará en la propiedad **DatosOut**. Valores: 0 -> DNIe acutal. 1->DNIe 3.0. 2->Pasaportes Nuevos. 6->Tarjetas permiso residencia con can
ObtenerNumeroSerie	Obtiene de la tarjeta su número de serie. El valor se encontrará en la propiedad **NumeroSerieOut**.
ObtenerNumeroSerie_Soporte	Obtiene de la tarjeta el número

	de serie de soporte de la tarjeta. El valor se encontrará en la propiedad **DatosOut**.
ObtenerCertCAComponente	Obtiene de la tarjeta el certificado de la CA intermedia de Componete. El valor se encontrará en la propiedad **DatosOut**.
ObtenerCertificadoCV_CA	Obtiene de la tarjeta el certificado CV_CA. El valor se encontrará en la propiedad **DatosOut**.
ObtenerVersionDNIe	Obtiene de la tarjeta la versión**.** El valor se encontrará en la propiedad **DatosOut**.
ObtenerVersionesLibrerias	Función que se encarga de obtener la versión de la librería DLL_ADMISTRATIVA. El valor se encontrará en las propiedades NombreLibreriaOut, VersionLibreriaOut, ArrayVersionLibreriasLenOut.

Gestión de datos:

Métodos	Descripción
ObtenerDatoPublico	Lee un dato público de la tarjeta. Se inicializará previamente la propiedad **EtiquetaDatoIn** y el valor se devolverá en **DatosOut**.
ObtenerDatoPrivado	Lee un dato privado de la tarjeta. Se inicializará previamente la propiedad **EtiquetaDatoIn** y el valor se devolvera en **DatosOut**.
ObtenerCAN	Lee el dato CAN de 6 bytes. El valor se devolverá en **DatosOut**.

ObtenerMRZ	Lee el dato MRZ. El valor se devolverá en **DatosOut**.
ObtenerDisplayMessage	Lee el dato DisplayMessage de 8 bytes. El valor se devolverá en **DatosOut**. Es necesario presentar el pin.
ObtenerEtiquetasDatosPublicos	Devuelve las etiquetas de todos los datos públicos. El número de etiquetas se encontrará en la propiedad **ArrayEtiquetasLenOut** y posteriormente el valor de la etiqueta se devolverá en **EtiquetaOut**.
ObtenerEtiquetasDatosPrivados	Devuelve las etiquetas de todos los datos privados. El número de etiquetas se encontrará en la propiedad **ArrayEtiquetasLenOut** y posteriormente el valor de la etiqueta se devolverá en **EtiquetaOut**.

Gestión de la generación de Hash:

Métodos	Descripción
Hash_Init	Inicializa el cálculo de un hash.
Hash_Update	Realiza una pasada del bucle para obtener un hash. Se inicializará previamente la propiedad **DatosIn**.
Hash_Final	Finaliza el bucle de llamadas a *Hash_Update* para obtener un hash. El valor se encontrará en la propiedad **HashOut**.
Hash	Nos devuelve el hash SHA-1 del dato, sin relleno PKCS#1. Se inicializará previamente la propiedad **DatosIn**. El valor se encontrará en la propiedad

	HashOut.

Gestión de Firma:

Métodos	Descripción
Firmar	Nos devuelve el dato firmado por DNIe. Se inicializarán previamente las propiedades **EtiquetaKeyPrvIn** y **DatosIn**. El valor se encontrará en la propiedad **FirmaOut**. Es necesario presentar al PIN cada vez que se realice una firma.
Verificar	Nos devuelve CKR_OK si todo ha ido bien y la verificación del dato es correcta. Se inicializarán previamente las propiedades **EtiquetaKeyPubIn, DatosIn** y **FirmaIn**.

Gestión de la Biometría:

Métodos	Descripción
ObtenerInfoBiometrica	Obtiene información de las huellas almacenadas. El valor se encontrará en la propiedad **DatosOut**.

Gestión de claves:

Métodos	Descripción
ObtenerEtiquetasClavesPublicas	Devuelve las etiquetas de todas las claves públicas. El número de etiquetas se encontrará en la propiedad **ArrayEtiquetasLenOut** y posteriormente el valor de la etiqueta se devolverá en **EtiquetaOut**.
ObtenerEtiquetasClavesPrivadas	Devuelve las etiquetas de todas las claves privadas. El número de etiquetas se encontrará en la propiedad **ArrayEtiquetasLenOut** y posteriormente el valor de la etiqueta se devolverá en **EtiquetaOut**.
ObtenerClavePublicaRSA	Lee los campos de una clave pública en la tarjeta. Se inicializará previamente la propiedad **EtiquetaKeyPubIn**. El valor se encontrará en la propiedad **ModuloOut** y **ExponenteOut**. Es necesario haber presentado el PIN previamente.
ObtenerClavePublicaRSAFirmada	Exportará la parte pública de la clave almacenada en la tarjeta en un certificado y devolveré el certificado de componente. Se inicializará previamente la propiedad **EtiquetaKeyPubIn**. Los valores se encontrarán en

	las propiedades **DatosOut** y **CertComponenteOut.** Es necesario haber presentado el PIN previamente.
ObtenerClavePublicaEC	Lee valor de la clave de curva elíptica. El valor se encontrarán en las propiedades **DatosOut.**
ObtenerClavePublicaECFirmada	Lee valor de la clave de curva elíptica firmada, devolviendo la clave y el certificado de componente. Los valores se encontrarán en las propiedades **DatosOut** y **CertComponenteOut.**

Gestión de Certificados:

Métodos	Descripción
ObtenerCertificado	Lee un certificado de la tarjeta. Se inicializará previamente la propiedad **EtiquetaCertificadoIn**, el valor se encontrará en **DatosOut**.
ObtenerEtiquetasTodosCertificados	Devuelve las etiquetas de todos los certificados. El número de etiquetas se encontrará en la propiedad **ArrayEtiquetasLenOut** y posteriormente el valor de la etiqueta se devolverá en **EtiquetaOut**.

Propiedades

El objeto DLL_Admin tiene definidos las siguientes propiedades:

- **ArrayEtiquetasLenOut**
  Tipo dato: int.
  Tipo acceso: Salida.

  Devuelve el número de etiquetas disponibles. Estas pueden ser etiquetas de datos, claves o certificados.

- **BITOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve el bit de una huella.

- **DatosIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga datos para realizar operaciones de firma de datos etc...

- **DatosOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR que contiene los datos solicitados después de una operación.

- **EtiquetaCertificadoIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga una etiqueta de un certificado para realizar alguna operación de recuperación de un certificado por su etiqueta.

- **EtiquetaDatoIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga una etiqueta de un dato para realizar alguna operación de recuperación de un dato por su etiqueta, eliminación de un dato por su etiqueta o almacenamiento de un dato.

- **EtiquetaKeyPubIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga una etiqueta de una clave pública para realizar alguna operación.

- **EtiquetaKeyPrivIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga una etiqueta de una clave privada para realizar alguna operación.

- **EtiquetaOut**
  Tipo dato: int, BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR que contiene una etiqueta que podrá ser de datos, claves, certificados. Este valor se extrae de un array de etiquetas por esa razón habrá que indicarle el índice del array mediante un int.

- **ExponenteOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR con el exponente de una clave.

- **FirmaIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

Carga una firma para realizar operaciones de firma de datos y carga de datos firmados.

- **FirmaOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR con una firma.

- **hashOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR con un hash.

- **hashIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga un hash.

- **lastresultOut**
  Tipo dato: int.
  Tipo acceso: Salida.

  Devuelve el valor de la operación realizada anteriormente. El código de error devuelto corresponderá con los códigos de error del estándar PKCS#11. No será responsabilidad del ActiveX formar una cadena con el mensaje de error.

- **ModuloOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR con el módulo de una clave.

- **NumeroSerieOut**
  Tipo dato: BSTR.
  Tipo acceso: Solo lectura.

Devuelve un BSTR que contiene el número de serie de la tarjeta.

- **PinIn**
  Tipo dato: BSTR.
  Tipo acceso: Entrada.

  Carga el pin con el que queremos hacer login.
  La longitud del pin oscilará entre 4 bytes y 16 bytes.

- **ulNumeroSerieLen**
  Tipo dato: unsigned long.
  Tipo acceso: Solo lectura.

  Devuelve la longitud del número de serie.

- **CertComponenteOut**
  Tipo dato: int, BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR que contiene un certificado de componente X509.

- **TipoBiometriaOut**
  Tipo dato: byte.
  Tipo acceso: Salida.

  Nos devolverá el tipo de algoritmo biométrico almacenado en la tarjeta. Un 0x10 corresponderá al algoritmo Sagem y un 0x20 corresponderá al algoritmo Siemems.

- **ArrayVersionLibreriasLenOut**
  Tipo dato: int.
  Tipo acceso: Salida.

  Devuelve el número de librerías disponibles.

- **VersionLibreriaOut**
  Tipo dato: BSTR.

Tipo acceso: Salida.

Devuelve un BSTR que contiene la versión de la librería. Este valor se extrae de un array de etiquetas por esa razón habrá que indicarle el índice del array mediante un int.

- **NombreLibreriaOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR que contiene el nombre de la librería. Este valor se extrae de un array de etiquetas por esa razón habrá que indicarle el índice del array mediante un int.

- **NombreOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

  Devuelve un BSTR que contiene el nombre del ciudadano.

- **Apellido1Out**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene el primer apellido del ciudadano.

- **Apellido2Out**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene el segundo apellido del ciudadano.

- **NumeroDNIOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene el DNI del ciudadano.

- **FechaNacimientoOut**
  Tipo dato: BSTR.

Tipo acceso: Salida.

Devuelve un BSTR que contiene la fecha de nacimiento del ciudadano.

- **NacionalidadOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la nacionalidad del ciudadano.

- **SexoOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene el sexto del ciudadano.

- **FechaExpiracionOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene de la fecha de expiración del DNI del ciudadano.

- **LocalidadNacimientoOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la localidad de nacimiento del ciudadano.

- **ProvinciaNacimientoOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la provincia de nacimiento del ciudadano.

- **NombrePadresOut**
  Tipo dato: BSTR.

Tipo acceso: Salida.

Devuelve un BSTR que contiene los nombres de los padres del ciudadano.

- **DomicilioOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene el domicilio del ciudadano.

- **LocalidadDomicilioOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la localidad del domicilio del ciudadano.

- **ProvinciaDomicilioOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la provincia de la localidad del ciudadano.

- **ImagenFacialOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la imagen facial del ciudadano.
Se trata de un jpeg2000.

- **ImagenFirmaOut**
  Tipo dato: BSTR.
  Tipo acceso: Salida.

Devuelve un BSTR que contiene la imagen de la firma del ciudadano.
Se trata de un jpeg2000.

Nota: Todos los datos de **entra** y **salida** de las propiedades estarán en **binario**.

Descripción de los parámetros de entrada y salida de los metodos:

A continuación, se va a proceder a dar una explicación mas amplia de los parámetros de entrada y salida de los métodos anteriormente vistos.

Gestión de Pin:

- **PresentarPin**
  Este método necesita de un parámetro de entrada. Este parámetro, que será el valor del pin, se inicializará previamente mediante la propiedad **PinIn**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Una vez invocado el método, si el pin es incorrecto, podremos llamar a la propiedad **DatosOut** para comprobar el número de intentos que nos quedan antes de bloquearse la tarjeta.

  Podemos obtener el número de intentos que nos quedan antes de bloquear la tarjeta invocando al método sin inicializar la propiedad **PinIn**. El número de intentos estará disponible en la propiedad **DatosOut**.

  Ejemplo:

```
lObjActiveX.PinIn = pin;
lObjActiveX.PresentarPin();
rv = lObjActiveX.lastresultOut;
```

Gestión de Conexión:

- **IniciarConexion**
  Este método no precisa de ningún parámetro de entrada.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Será necesario que sea el primer método que se lance.

- **FinalizarConexion**
  Este método no precisa de ningún parámetro de entrada.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**. Tenemos que tener en cuenta que esta función se va a encargar de liberar todos los recursos y por tanto tirar todos los canales seguros establecidos hasta ese momento tanto, por esta razón hay que hacer un buen uso de este método, sabiendo en todo momento lo que supone finalizar la conexión.
  Es importante que antes de cerrar la aplicación o página web que hace uso de la librería, es necesario llamar a este método, ya que internamente posee un thread que se encarga de detectar la extracción e inserción del DNIe.

- **ObtenerTipoDocumento**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el tipo de documento que hay introducido en el lector leyendo la propiedad **DatosOut.** Los valores devueltos corresponderán a 0x00 si se trata de un **DNIe actual**, 0x01 corresponderá a un **DNIe 3.0,** 0x02 si se tratara de un **Pasaporte** y 0x6 si fuera una tarjeta permiso residencia con can. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:
  ```
  lObjActiveX.ObtenerTipoDocumento()
  TipoDocumento = lObjActiveX.DatosOut;
  rv = lObjActiveX.lastresultOut;
  ```

- **ObtenerNumeroSerie**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener la longitud del número de serie mediante la propiedad **ulNumeroSerieLen** y el valor mediante la propiedad **NumeroSerieOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Ejemplo:

```
lObjActiveX.ObtenerNumeroSerie()
LenNumeroSerie = lObjActiveX.ulNumeroSerieLen;
NumeroSerie = lObjActiveX. NumeroSerieOut;
rv = lObjActiveX.lastresultOut;
```

- **ObtenerNumeroSerie_Soporte**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener la longitud del número de serie mediante la propiedad **ulNumeroSerieLen** y el valor mediante la propiedad **DatosOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:

```
lObjActiveX.ObtenerNumeroSerie_Soporte()
LenNumeroSerie = lObjActiveX.ulNumeroSerieLen;
NumeroSerie = lObjActiveX.DatosOut;
rv = lObjActiveX.lastresultOut;
```

- **ObtenerVersionDNIe**
  Este método se encarga de obtener de la tarjeta la versión del DNIe.

  Si se trata de una tarjeta que no tiene dicha información porque se trate de una tarjeta sin fichero de versión se tratará de construirla.

  Ejemplo del contenido:
  Salida de la FNMT:
  DNIe 01.90 A11 H 4C34

  Label, nombre del producto: DNIe.
  Versión del sistema operativo: 01.90
  Tipo de Biometría: A (Sagem). B(Siemens).
  Personalización: 11
  Identificador del chip: H 4C34.
  Tarjeta expedida:
  DNIe 01.90 A11 H 4C34 **EXP 1-1**

  Label: EXP
  Mapa de ficheros expedido y sus condiciones de acceso: 1-1

- **ObtenerVersionesLibrerias**

Este método no necesita de ningún parámetro. Se encarga de extraer las versiones de las librerías DLL_ADMINISTRATIVA, PKCS11 y librerías biométricas.
Una vez invocado el método, podremos obtener el número de librerías disponibles mediante la propiedad **ArrayVersionLibreriasLenOut** y el valor del nombre y versión mediante las propiedades **NombreLibreriaOut** y **VersionLibreriaOut** que recibirá como parámetro un índice.
Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Gestión de Datos:

- **ObtenerDatoPublico**
  Este método necesita de un parámetro de entrada, corresponderán con la etiqueta del dato. Este parámetro se inicializará previamente mediante la propiedad **EtiquetaDatoIn**. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:

```
lObjActiveX.EtiquetaDatoIn = etiqueta;
lObjActiveX.ObtenerDatoPublico();
rv= lObjActiveX.lastresultOut;
Datos =lObjActiveX.DatosOut;
```

- **ObtenerDatoPrivado**
  Este método necesita de un parámetro de entrada, corresponderán con la etiqueta del dato. Este parámetro se inicializará previamente mediante la propiedad **EtiquetaDatoIn**. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.EtiquetaDatoIn = etiqueta;
lObjActiveX.ObtenerDatoPrivado();
rv= lObjActiveX.lastresultOut;
Datos =lObjActiveX.DatosOut;
```

- **ObtenerCAN**
  Este método no necesita de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:
```
lObjActiveX.ObtenerCAN();
rv= lObjActiveX.lastresultOut;
CAN =lObjActiveX.DatosOut;
```

- **ObtenerMRZ**
  Este método no necesita de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:
```
lObjActiveX.ObtenerMRZ();
rv= lObjActiveX.lastresultOut;
MRZ =lObjActiveX.DatosOut;
```

- **ObtenerDisplayMessage**
  Este método no necesita de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**. Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:
```
lObjActiveX.ObtenerDisplayMessage();
rv= lObjActiveX.lastresultOut;
DisplayMessage =lObjActiveX.DatosOut;
```

- **ObtenerEtiquetasDatosPublicos**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el número de etiquetas disponibles mediante la propiedad **ArrayEtiquetasLenOut** y el valor de la etiqueta mediante la propiedad **EtiquetaOut** que recibirá como parámetro un índice.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

- **ObtenerEtiquetasDatosPrivados**

Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el número de etiquetas disponibles mediante la propiedad **ArrayEtiquetasLenOut** y el valor de la etiqueta mediante la propiedad **EtiquetaOut** que recibirá como parámetro un índice.
Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
Ejemplo para obtener las etiquetas de cualquier tipo de datos (públicos, privados):

```
var NumEtiquetas = 0;


//Inicialmente invocamos el método ObtenerEtiquetasDatos Publicos.
rv= lObjActiveX.ObtenerEtiquetasDatosPublicos();

//Comprobamos que la operación ha ido bien.
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
    alert("Se he producido el error  " +  rv);
    return;
}
//Obtenemos el número de etiquetas disponibles..
NumEtiquetas =lObjActiveX.ArrayEtiquetasLenOut;
if (NumEtiquetas == 0)
{
      alert("No hay datos públicos en la tarjeta");
      return;
}
var etiquetas=new Array(NumEtiquetas);
var option;

//Obtenemos todas las etiquetas.
for (i=0;i<NumEtiquetas;i++)
{
  etiquetas[i] = lObjActiveX.EtiquetaOut(i);
  option = new Option(etiquetas[i]);
  inForm.EtiDatosPublicos.options[i]=option;
}
```

Gestión de la generación de Hash:

- **Hash_Init**
  Este método no precisa de ningún parámetro de entrada.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

- **Hash_Update**

Este método necesita de un parámetro de entrada. Este parámetro, será el valor del dato, se inicializará previamente mediante la propiedad **DatosIn**.
Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Ejemplo:

```
lObjActiveX.DatosIn = dato;
lObjActiveX.Hash_Update();
rv = lObjActiveX.lastresultOut;
```

- **Hash_Final**
  Este método no precisa de ningún parámetro de entrada. Para obtener el valor del hash deberemos invocar la propiedad **HashOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.Hash_Final();
rv = lObjActiveX.lastresultOut;
Hash = lObjActiveX.HashOut;
```

- **Hash**
  Este método necesita de un parámetro de entrada. Este parámetro, será el valor del dato, se inicializará previamente mediante la propiedad **DatosIn**. Para obtener el valor del hash calculado, deberemos invocar la propiedad **HashOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.DatosIn = dato;
lObjActiveX.Hash();
rv = lObjActiveX.lastresultOut;
Hash = lObjActiveX.HashOut;
```

Gestión de Firma:

- **Firmar**
  Este método necesita de dos parámetros de entrada, corresponderán con la etiqueta de la clave privada con la que queremos firmar los datos y el valor del dato a firmar. Estos parámetros se inicializarán previamente mediante las

propiedades **EtiquetaKeyPrivIn** y **DatosIn**. Para obtener el valor de la firma, deberemos invocar la propiedad **FirmaOut**.
Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

El algoritmo que se sigue para realizar la firmar digital es un RSA de 2048.
Es necesario haber presentado el pin justamente antes de llamar a firmar. Se trata de una condición de seguridad que tiene el DNIe. Cada vez que se quiera realizar una firma es necesario presentar el pin.

Ejemplo:

```
lObjActiveX.EtiquetaKeyIn = EtiquetaKey;
lObjActiveX.DatosIn = Datos;
lObjActiveX.Firmar();
rv = lObjActiveX.lastresultOut;
Firma = lObjActiveX.FirmaOut;
```

- **Verificar**
  Este método necesita de tres parámetros de entrada, corresponderán con la etiqueta de la clave privada con la que queremos verificar los datos, el valor del dato y la firma del dato. Estos parámetros se inicializarán previamente mediante las propiedades **EtiquetaKeyPubIn, DatosIn** y **FirmaIn**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.EtiquetaKeyIn = EtiquetaKey;
lObjActiveX.DatosIn = Datos;
lObjActiveX.FirmaIn = Datos;
lObjActiveX.Verificar();
rv = lObjActiveX.lastresultOut;
```

Gestión de la Biometria:

- **ObtenerInfoBiometrica**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener la información biométrica mediante la propiedad DatosOut.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Este método nos va a devolver el número de huellas almacenadas y el BIT.

Ejemplo:

```
lObjActiveX.ObtenerInfoBiometrica();
rv= lObjActiveX.lastresultOut;
Infobio = lObjActiveX.DatosOut;
```

02 01 02
7F6024830121060660864801650 3A1178101088201068702010188020005B10781020C3882010D
7F6024830122060660864801650 3A1178101088201058702010188020005B10781020C3882010D

Donde 02 es un tag, 01 nos indica que el siguiente byte corresponde con el número de huella que se encuentran almacenadas en la tarjeta y el resto son los BIT de cada huella. En este ejemplo tendríamos dos huellas almacenadas y a continuación tendríamos el BIT de cada huella.

Gestión de claves:

- **ObtenerEtiquetasClavesPublicas**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el número de etiquetas disponibles mediante la propiedad **ArrayEtiquetasLenOut** y el valor de la etiqueta mediante la propiedad **EtiquetaOut** que recibirá como parámetro un índice.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
var NumEtiquetas = 0;
rv= lObjActiveX.ObtenerEtiquetasClavesPublicas ();
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
        alert("Se he producido el error  " +  rv);
        return;
}
NumEtiquetas =lObjActiveX.ArrayEtiquetasLenOut;
if (NumEtiquetas == 0)
{
   alert("No hay claves públicas en la tarjeta");
   return;
```

```
}
var etiquetas=new Array(NumEtiquetas);
var option;

for (i=0;i<NumEtiquetas;i++)
{
   etiquetas[i] = lObjActiveX.EtiquetaOut(i);
   option = new Option(etiquetas[i]);
   inForm.EtiKeyPublicas .options[i]=option;
}
```

- **ObtenerEtiquetasClavesPrivadas**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el número de etiquetas disponibles mediante la propiedad **ArrayEtiquetasLenOut** y el valor de la etiqueta mediante la propiedad **EtiquetaOut** que recibirá como parámetro un índice.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

- **ObtenerClavePublicaRSA**
  Este método necesita de un parámetro de entrada, el cual corresponderá con la etiqueta de la clave pública. Este parámetro se inicializará previamente mediante la propiedad **EtiquetaKeyPubIn**. Para obtener el valor de los componentes de la clave (módulo y exponente), deberemos invocar las propiedades **ModuloOut** y **ExponenteOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.EtiquetaKeyIn= EtiquetaKey;
lObjActiveX.ObtenerClavePublicaRSA();
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
        alert("Se he producido el error  " +  rv);
        return;
}
var Modulo =lObjActiveX.ModuloOut;
var Exponente = lObjActiveX.ExponenteOut;

if ((Modulo.length)&&(Exponente.length))
{
    document.getElementById("ModuloSalida").value = Modulo;
   document.getElementById("ExponenteSalida").value = Exponente;
 }

else
   alert("Clave no existe");

return;
```

- **ObtenerClavePublicaRSAFirmada**
  Este método necesita de un parámetro de entrada, el cual corresponderá con la etiqueta de la clave pública. Este parámetro se inicializará previamente mediante la propiedad **EtiquetaKeyPubIn**. Para obtener el valor del cerificado de componente deberemos invocar la propiedad **CertComponenteOut** y para obtener la clave pública firmada deberemos invocar la propiedad **DatosOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.
  Ejemplo:

```
lObjActiveX.EtiquetaKeyPubIn= EtiquetaKeyPub;
lObjActiveX.ObtenerClavePublicaRSAFirmada();
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
        alert("Se he producido el error  " +  rv);
        return;
}
ClaveRSAFirmada = lObjActiveX.DatosOut
CertComponente = lObjActiveX.CertComponenteOut
```

- **ObtenerClavePublicaEC**
  Este método no necesita de ningún parámetro de entrada. Para obtener el valor, deberemos invocar la propiedad **DatosOut.**
  El valor devuelto corresponde a la clave pública EC de la tarjeta devolviendo todos los parámetros de la curva con su tag y longitud correspondiente. Los tag vendrán siguiendo el siguiente orden 0x81,0x82,0x83,0x84,0x85,0x87,0x86.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.ObtenerClavePublicaEC();
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
        alert("Se he producido el error  " +  rv);
        return;
}
Clave = lObjActiveX.DatosOut;
```

- **ObtenerClavePubliceECFirmada**
  Este método no necesita de ningún parámetro de entrada. Este método no necesita de ningún parámetro de entrada. Para obtener el valor del cerificado de componente deberemos

invocar la propiedad **CertComponenteOut** y para obtener la clave pública firmada deberemos invocar la propiedad **DatosOut**.
Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

Ejemplo:

```
lObjActiveX.ObtenerClavePublicaECFirmada();
rv= lObjActiveX.lastresultOut;
if (rv != 0)
{
        alert("Se he producido el error  " +  rv);
        return;
}
ClaveECFirmada = lObjActiveX.DatosOut
CertComponente = lObjActiveX.CertComponenteOut
```

Gestión de certificado:

- **ObtenerCertificado**
  Este método necesita de un parámetro de entrada, corresponderán con la etiqueta del certificado. Este parámetro se inicializará previamente mediante la propiedad **EtiquetaCertificadoIn**. Una vez invocado el método, podremos obtener el valor del dato mediante la propiedad **DatosOut**.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.

  Ejemplo:

```
lObjActiveX.EtiquetaCertificadoIn = etiqueta;
lObjActiveX.ObtenerCertificado();
rv = lObjActiveX.lastresultOut;
Certificado = lObjActiveX.DatosOut;
```

- **ObtenerEtiquetasTodosCertificados**
  Este método no precisa de ningún parámetro de entrada. Una vez invocado el método, podremos obtener el número de etiquetas disponibles mediante la propiedad **ArrayEtiquetasLenOut** y el valor de la etiqueta mediante la propiedad **EtiquetaOut** que recibirá como parámetro un índice.
  Para comprobar si la operación es correcta llamaremos a la propiedad **lastresultOut**.