Revision 6 – 01/25/2006

MEDIASOFT
TECHNOLOGIES

# eMedia Card Designer

## Plug-In SDK

The computer program this SDK refers to is intended to be run on any IBM PC or compatible computer, running Microsoft Windows 9x, NT, 2000, XP and 2003 Operating Systems.

This computer program is a COM/DCOM Server/Client and is strictly compliant with the COM/DCOM interfaces as described in the specification defined by Microsoft Corporation.

Please refer to the End-user license and the limited warranty terms for any questions that may concern the license usage granted by Mediasoft Technologies to the registered user.

Should you have any question regarding this document, its contents or the computer software "eMedia Card Designer", please contact us:

Web: http://www.emedia-cards.com

e-mail: support@emedia-cards.com

Postal address:

Mediasoft Technologies
228, rue de la Convention
75015 Paris – France

This document is part of the Plug-Ins SDK, available on the eMedia Card Designer web site as a downloadable package, and in the technical notes of the support section.

# CONTENTS

# REVISION INFORMATION

This SDK document is available for the build **582** and more of eMedia Card Designer. For any information in this document that may refers to/from a specific build number, this number will be described.

The part II of this document is only available for build **631** and more of eMedia Card Designer.

This document was last updated on **Wednesday, January 25, 2006 22:00 GMT**.

# SUMMARY

This document describes how to create your own plug-in that could be used in eMedia to obtain more functionalities of the software.

## About the plug-ins

Plug-ins can be implemented using Microsoft Visual Basic™, Microsoft Visual C++™ with MFC or ATL libraries or other development tool having the ability to create COM components.

Plug-ins are in-process or out-of-process COM components that expose through the standard interfaces methods and properties having a specific name. These methods and properties are called by the eMedia program when the end user:

- Prints out a card,
- Adds a record to the database,
- Deletes a record in the database,
- Modifies a record in the database,
- A record is found using CoolReader™,
- Creates a new card,
- Opens a card,
- Saves a card,
- Acquires a photo,
- Switches to design mode or operating mode,
- Selects the plug-in in the "Tools" menu.

Plug-ins can have a configuration interface to help the end user to customize it. The configuration interface is called through a specific method when the end user chooses the "Plug-ins" command in the "Tools" menu, then selects the plug-in name.

This document demonstrates the development of a custom plug-in, starting with the sample plug-in source files available in the Plug-Ins SDK provided in the download section of the internet site.

All the code examples shown below are written in Microsoft® Visual Basic™.

## About this document

To implement your own plug-in, you'll have to create an in-process COM component or an out-of-process COM component.

This component must contain a class with specific methods and properties names. These are detailed in the part I of this document.

Your component may have to retrieve the characteristics of the current card template opened in eMedia Card Designer. For this purpose, one of the methods of the component is called by eMedia and provides a class object. The properties and methods of this object are described in the part II of this document.

# WHAT IS NEEDED TO CREATE YOUR OWN PLUG-IN

In order to use the plug-in in eMedia, you must provide to the end user the following:

- The compiled plug-in file (.dll file or .exe file)
- The plug-in descriptor file (.emp file)

The end-user will then perform the three following tasks:

1. Copy the files into a folder (usually the eMedia installation folder),
2. Register the COM component,
3. Register the plug-in in eMedia.

The end user must be logged into the system with administrative rights, and the eMedia Card Designer software must not be running.

## Copying the files

You can provide to the end user the .exe or .dll file and the .emp file on a floppy disk or a CD-ROM. The user will then copy these files into a folder of the hard disk. Usually, the files are copied in the eMedia Card Designer installation folder.

You may provide to your end user an installation package which will copy and register the plug-in.

If your plug-in is developed using Microsoft® Visual Basic™, you will not have to provide neither the VB runtime, the ADO 2.5 libraries or the COM/DCOM files, as these libraries are already installed on the client computer by the eMedia Card Designer installation process.

## Registering the COM component (optional)

When a COM component is installed on a computer, it must be recognized by Windows. For that purpose, keys must be written in the system Registry. To do so:

- If the component is an out-of-process COM component (e.g. the file have the .exe extension), the user must launch it one time to perform the registration. The file can also be launched with the /REGSERVER command-line option.
- If the component is an in-process COM component (e.g. the file have the .dll extension), the user must launch the REGSVR32.EXE program with the .dll file name as a command-line parameter.

For instance: Your plug-in file is named MyPlugIn.dll. The user will start a command prompt window, browse to the folder containing the file, will then type:

```
REGSVR32 MyPlugIn.dll
```

This registration is optional if the user registers the plug-in by using the "Add plug-in" button in the "Select plug-in" dialog box (see below), as eMedia proceeds to the registration of the COM component by itself. But, if you plan to provide an installation package, you will have to register the COM component in the system registry by yourself.

## Registering the plug-in in eMedia

eMedia needs to recognize a COM component as one of its plug-ins. To do so, the user must register it in eMedia by adding and selecting it in the "Select plug-in" dialog box. To do so:

- Start eMedia Card Designer, Professional or Expert version (or standard version with the demonstration mode enabled).

- Open the "Tools" menu,
- Select the "Plug-Ins" command,
- Click on the "Select plug-ins" sub-command.
- In the "Select plug-ins" dialog box, click on the "Add plug-in" button.
- Select the plug-in descriptor file (.emp file) and click "Open".
- Click OK in the "Select plug-in" dialog box.

If you plan to provide the plug-in with an installation package, you can perform this operation by yourself: modify the system Registry in that way:

1. Create a registry key having your plug-in name in the following registry key:
   **`HKEY_LOCAL_MACHINE\Software\Mediasoft Technologies\eMedia\Plug-Ins`**
2. In this key, create the value **`ProgID`** of **`REG_SZ`** type. Populate the data of this value with the ProgID of your COM component.
3. Add a new value in the following registry key:
   **`HKEY_CURRENT_USER\Software\Mediasoft Technologies\eMedia\Plug-Ins`**
   Set the value data to the friendly name of your plug-in.

The **`HKEY_CURRENT_USER\...`** value specifies what plug-in are active for the current user. The data of these values are used to retrieve the **`HKEY_LOCAL_MACHINE\...`** key. In this key, the data of the **`ProgID`** value is used by the eMedia Card Designer program to instantiate the component. There may be more than one Active value in the **`HKEY_CURRENT_USER\...`** key, as more than one plug-in may be active at the same time. The values are named (by default) **`Active 1`**, **`Active 2`**, **`Active 3`**, etc

## The plug-in descriptor file

Once your plug-in implemented and compiled into a COM component (either as an ActiveX EXE or an ActiveX DLL), you must prepare the descriptor file.

This file is only needed if you don't plan to provide an installation package. This file will be selected by the end user when he will click on the "Add plug-in" button of the "Select plug-in" dialog box. If you plan to provide an installation package, you can avoid the descriptor file creation and perform the tasks described above.

The plug-in descriptor file contains the plug-in description, which would be read by eMedia Card Designer when the end user registers the plug-in into eMedia. It is a profile file, containing only one section named "Plug-In". This section contains the four following keys:

- **Type**: this key MUST contain the text: "eMedia Plug-In Definition File". This key is read by eMedia to assume that this file contains a plug-in description.
- **File name**: this key contains the plug-in file name. This is used by eMedia Card Designer to register the COM component into the system registry.
- **ProgID**: this key contains the ProgID of the COM component (i.e. the project name and the class name of your plug-in separated by a dot).
- **Title**: this key contains a friendly text representing your plug-in. eMedia will create this key in the system registry (see step #1 in the previous paragraph above), and will add the ProgID value containing the data found in the ProgID key of the descriptor file.

# OPENING THE SAMPLE PLUG-IN SOURCE CODE

The sample plug-in provided in the Plug-Ins SDK is a Microsoft® Visual Basic™ project. You can open it with Visual Studio 6, or by double-clicking on the .vbp file.

The sample plug-in consists of a VB class, called "Sample", containing the entry points of the component as public methods and functions. Currently, these methods and functions only display a message box.

You can modify the code of these methods and functions, change the class properties and the project properties, and then recompile the component.

This article will explain you how to modify these information.

## Class and project properties

The first things that you may change are the class and project names, as these information permit to build the ProgID of your component. The project name will be the left part of the ProgID, and the class name will be the right part of the ProgID:

In the sample plug-in, the project name is **`eMediaSample`**, and the class name is Sample. Then, the ProgID of the COM component will be **`eMediaSample.Sample`**.

To change the project name, open the "Project" menu and select the "Project properties" choice. In the "Project properties" dialog box, click on the "General" tab, then change the contents of the "Project name" text-box.

To change the class name, click on the class in the project explorer pane, then change the "Name" property of the class in the properties pane.

You can also customize the project information (such as the project type (select ActiveX EXE or ActiveX DLL), the startup object, the component description, the compatibility mode and the version information) in the "Project properties" dialog box.

## Plug-in source code

In the class code, you will find first the enumerations that are used for method parameters. Do not change these enumerations, as these values are transmitted from eMedia Card Designer.

The **`m_Operation`**, **`m_Active`**, **`m_Recordset`** and **`m_eMediaObject`** private variables are used internally by the class. All these variables may be deleted from the source code if you don't need them:

- **`m_Operation`** stores the current running operation (see **`EPI_Job`**),
- **`m_Active`** contains the activity flag of the plug-in (see **`Active`**),
- **`m_Recordset`** contains the copy of the ADO recordset currently used by eMedia Card Designer (see **`EPI_Database`**),
- **`m_eMediaObject`** contains the instance of the **`CardObjects`** class provided by eMedia Card Designer (see **`EPI_GetCardObject`**).

# PART I

# PLUG-IN COMPONENT REFERENCE

# REFERENCE

In the following pages, the properties and methods of the plug-in are described.

Basically, the plug-in you'll create must contain the following property and the following methods. If you don't implement these, your plug-in will not be able to execute code for the corresponding interfaces, but eMedia Card Designer will not crash.

If some of these properties and methods are not useful, implement them without any code in the corresponding functions, so you'll be able to modify your plug-in later without any need to refer to this documentation.

In the list below, the "Name" column contains the name of the property or method. The "Description" column gives some information about the purpose of the class member, and build describes the minimum eMedia Card Designer build number needed to implement this class member.

## Properties

| Name | Description | Build |
|---|---|---|
| **Active** | Specifies if the plug-in is enabled or not. | 582+ |

## Methods

| Name | Description | Build |
|---|---|---|
| **EPI_Initialize** | Plug-In initialization. | 582+ |
| **EPI_GetCardObject** | Provides to the plug-in the instance of the CardObjects class of eMedia Card Designer. The class members allow the plug-in to retrieve all the characteristics of the current opened card. | 630+ |
| **EPI_Terminate** | Plug-In termination. | 582+ |
| **EPI_Preferences** | Called when the user selects the plug-in name in the "Plug-ins" command of the "Tools" menu in eMedia Card Designer. | 582+ |
| **EPI_Supports** | Called to ask the plug-in if a specific operation is supported. | 582+ |
| **EPI_SetMode** | Called when the user switches between design and operating modes. | 582+ |
| **EPI_Job** | Informs the plug-in that an operation is about to begin and/or to end. | 582+ |
| **EPI_Database** | Provides the current ADO recordset used in eMedia Card Designer to the plug-in. | 582+ |
| **EPI_DataTransmit** | Provides to the plug-in one of the current eMedia Card Designer object name and value. | 582+ |

## Active

This property has a Boolean type. It can be set to FALSE (zero) or TRUE (all value different from zero, usually -1).

This property is set by your program if you want to give your user the ability to use or not your plug-in.

Typically, this will be done by a configuration dialog box, containing a check box that will reflect the value of the Active property. The configuration dialog box will be called by the `EPI_Preferences` method.

eMedia will check the `Active` property of your plug-in before every call to the `EPI_Job`, `EPI_Database` and `EPI_DataTransmit`. If your property is set to TRUE, the call is done. If your property is set to FALSE, no call is done.

If you don't plan to use this functionality, either set the `m_Active` private data to TRUE in the `EPI_Initialize` method, or replace the Active Property Let/Get by the following code:

**Current implementation**

```
'In the general declarations section
Private m_Active As Boolean

'In the class module code

Public Property Let Active( _
            ByVal vActive As Boolean)
  m_Active = vActive
  If vActive Then
    MsgBox "The sample plug-in " & _
           "has been activated"
  Else
    MsgBox "The sample plug-in " & _
           "has been deactivated"
  End If
End Property

Public Property Get Active() _
                       As Boolean
  Active = m_Active
End Property
```

**Replace by…**

```
'In the general declarations section
'Delete the declaration

'In the class module code

Public Property Get Active() _
                       As Boolean
  Active = True
End Property
```

Property declaration:

```
Public Property Let Active(ByVal vActive As Boolean)
Public Property Get Active() As Boolean
```

See also:

`EPI_Initialize` method

`EPI_Preferences` method

`EPI_Database` method

`EPI_DataTransmit` method

## EPI_Initialize

This method is called by eMedia Card Designer when it is necessary to initialize the plug-in (eMedia startup or plug-in activation). This method accepts one String parameter (LPSTR C/C++ type) which specifies the language used in the eMedia interface, as you may want to use the same language in your Plug-In.

This method is a function which must return FALSE (zero) or TRUE (-1). If this function returns false, eMedia considers that the plug-in cannot be activated. None of the methods of the plug-in will be then called anymore.

### Function declaration:

```
Public Function EPI_Initialize(ByVal CurrentLanguage As String) As Boolean
```

### Return value:

True (-1) if the plug-in can be called from eMedia Card Designer, false (0) otherwise.

### See also:

`EPI_GetCardObject` method

`EPI_Terminate` method

## EPI_GetCardObject

This method is called by eMedia Card Designer just after the EPI_Initialize method. The provided parameter of this call is an instance of the eMedia Card Designer CardObject class. Storing this instance allows the plug-in to retrieve all the characteristics of the current card opened in the software, for instance to retrieve the card properties, objects list and objects properties.

### Function declaration:

```
Public Sub EPI_GetCardObject (TheObject As Object)
```

### Return value:

No return value.

### Notes:

In the declaration of the function, the parameter type has been set to the generic type `Object`. You may transform this type to `eMedia.CardObject` to allow early binding, referencing the eMedia Card Designer component from the "References" command of the "Project" menu.

The class members of the `CardObject` class are detailed later in this document.

### See also:

`EPI_Initialize` method

`CardObject` class members

## EPI_Terminate

This method is called by eMedia Card Designer when it is necessary to terminate the plug-in processing (such as when eMedia stops or when the plug-in is deactivated). This method doesn't accept any parameter nor returns a value.

### Function declaration:

```
Public Sub EPI_Terminate()
```

See also:

`EPI_Initialize` method

## EPI_Preferences

This method is called by eMedia Card Designer when the user selects the plug-in name in the "Plug-ins" menu. You may implement in this method a call to a configuration dialog box, an about dialog box or any configuration code.

The parameter is the main eMedia form object and can be used in the Plug-in to retrieve the properties of the main form (such as its position, its hWnd, etc). The parameter is declared in the Plug-in as the `Object` generic type, but it is at running time an instance of the `Form` class of Microsoft® Visual Basic™ 6.0.

### Function declaration:

```
Public Sub EPI_Preferences(AppForm As Object)
```

### See also:

`EPI_Initialize` method

## EPI_Supports

This method is called by eMedia Card Designer to check if your plug-in supports the requested function. This method is called before any call to the `EPI_Job` method. If this method returns TRUE, then the `EPI_Job`, `EPI_Database` and `EPI_DataTransmit` methods will be called. Otherwise, none of them will be called. This method helps eMedia Card Designer to run faster, by avoiding unneeded calls.

The `EPI_Supports` method accepts one parameter of `enumPlugInOperations` type, which is the operation requested.

To implement this method, modify the contents of the `Select Case` statement, and set to True or False every return values, depending on the functionalities implemented by your plug-in.

**Function declaration:**

```
Public Function EPI_Supports( _
                           ByVal lngFunctionality As enumPlugInOperations _
                            ) As Boolean
```

**See also:**

`EPI_Job` method

`EPI_Database` method

`EPI_DataTransmit` method

## EPI_SetMode

This method is called by eMedia Card Designer when the user switches to the design mode or to the operating mode, to provide this information to the plug-in in order to perform some initialization tasks.

The `EPI_SetMode` method accepts one parameter of `enumPlugInMode` type, which contains the new current mode (0 = Design mode, 1 = Operating mode).

**Function declaration:**

```
Public Sub EPI_SetMode(ByVal lngNewMode As enumPlugInModes)
```

**See also:**

## EPI_Job

This method is called by eMedia when one of these predefined actions occurs:

- A card print-out is beginning,
- A record is added in the database,
- A record is deleted in the database,
- A record is modified in the database,
- An image acquisition was performed,
- CoolReader have selected a record,
- A card file has been opened,
- A card file has been saved,
- A new card has been created,
- A job is finishing.

This function accepts one numeric parameter of **`Long`** type (**`int`** type in C). This value is one of the **`enumPlugInOperations`** values and tells the plug-in what kind of operation is to begin. This method doesn't return anything.

The methods are called by eMedia Card Designer according to this scheme:

1. Call to the **`EPI_Supports`** method to ask the plug-in if it supports the operation currently running in eMedia Card Designer. eMedia holds execution until the function call is processed. If the function call returns true, the steps defined below are performed.
2. Call to the **`EPI_Job`** method, to provide information about the operation that eMedia Card Designer is performing. eMedia holds execution until the function call is processed.
3. Call to the **`EPI_Database`** method, to provide database information. eMedia holds execution until the function call is processed.
4. Call to the **`EPI_DataTransmit`** method for each object significant for the operation:
   - For printing, for adding, modifying and deleting records: all card objects names and values, and/or all database field names and values.
   - For an image acquisition: the name of the photo object and its value (the file name containing the picture taken). Beware of the fact that the picture file name is a temporary file. The file will be deleted when the **`EPI_Job`** will be called again with the **`PlugInEnd`** parameter.
   - For CoolReader™: the database linked field name and the value.
   - For a card opened: the file name of the card and the current mode of eMedia.
   - For a card saved: the file name of the card.
   - For a card creation: nothing.
5. Call to the **`EPI_Job`** method to inform the plug-in that the operation is finished.

**Example:**

1. The user clicks on the “Delete record” button in operation mode while a database record is displayed. eMedia reads the **`Active`** property to see if the plug-in is active. If the property returns false, eMedia jumps to step 8 below.
2. If the **`Active`** property is true, eMedia calls the **`EPI_Supports`** method:
   **`EPI_Supports(PlugInDBDelete)`**

3. If the call returned TRUE, eMedia calls the **`EPI_Job`** method:
   **`EPI_Job(PlugInDBDelete)`**
4. eMedia calls the **`EPI_Database`** method and provides the current recordset:
   **`EPI_Database(`*`CurrentRecordsetObject`*`)`**
5. eMedia calls the **`EPI_DataTransmit`** for the first database field:
   **`EPI_DataTransmit(PlugInDBField, "BIRTH_YEAR", 1950, 4)`**
6. eMedia calls the **`EPI_DataTransmit`** method for the second object on the card, for the third, etc
7. eMedia calls the **`EPI_Job`** method at the end to inform the plug-in that everything has been provided:
   **`EPI_Job(PlugInEnd)`**
8. eMedia deletes the record in the database.

A call to the **`EPI_Job`** method is followed by one call to the **`EPI_Database`** method, then zero, one or more calls to the **`EPI_DataTransmit`** method, then by another call to **`EPI_Job`** with the value **`PlugInEnd`**.

**Function declaration:**

```
Public Sub EPI_Job(ByVal op As enumPlugInOperations)
```

**See also:**

`Active` property

`EPI_Supports` method

`EPI_Database` method

`EPI_DataTransmit` method

## EPI_Database

This method is called by eMedia Card Designer just after the call to **EPI_Job**, and provides to your plug-in the current ADO recordset object.

This method accepts two parameters:

- The first parameter is an instance of the **ADODB.Recordset** class. This parameter is an ADO Recordset, and is really the recordset used in eMedia. So, you must not modify this recordset directly, nor move to another record.
- The second parameter is one of the values of the **enumDBEngine** enumeration, and specifies the type of the database (if you have to check its capabilities).

If you plan to modify data in the recordset, it will be better to clone it first, then to do your modifications on the cloned recordset.

If you don’t want to use the **EPI_Database** method nor the database capabilities, perform the following steps:

1. Remove the project reference to the Microsoft® ADO libraries: Open the "Project" menu in Microsoft® Visual Basic™ and click on the "References" command. In the references list, uncheck the "Microsoft ActiveX Data Objects 2.5 library" element, and press OK.
2. Remove the **m_Recordset** variable in the general declarations section of the class module.
3. Remove the **EPI_Database** method.

### Function declaration:

```
Public Sub EPI_Database(ByVal rsData As ADODB.Recordset, _
                        ByVal lngDatabaseEngine As enumDBEngine)
```

### See also:

**EPI_Supports** method

**EPI_Job** method

## EPI_DataTransmit

This method is called by eMedia Card Designer to provide an object value. The object can be a database field, a record number, a card object, the front side of the card, the back side of the card, the smart card chip value or the magnetic strips values.

This method accepts four parameters:

- One of the values of the `enumPlugInDataSource` enumeration. This value is a `Long` (`int` type in C). This parameter informs the plug-in about the object transmitted.
- A string which contains the object name (for objects on the card) or the field name (for database fields), or a special term enclosed with ":". This value is a `String` (`LPSTR` type in C).
- A variant which contains the value of the object. As it is of `Variant` type, you must retrieve the sub-type by yourself. You can use the `VarType()` function of Microsoft® Visual Basic™ for that purpose.
- The size of the third parameter in bytes (or in characters for string values). This value is a `Long` (`int` type in C).

In this method, you can store the values, the object names or perform directly an operation onto the object. Beware of the fact that the object name and value is passed by value, not by reference: acting on this name and/or this value doesn't affect the original object in eMedia Card Designer.

**Function declaration:**

```
Public Sub EPI_DataTransmit(ByVal parmSource As enumPlugInDataSource, _
                            ByVal parmName As String, _
                            ByVal parmValue As Variant, _
                            ByVal parmSize As Long)
```

**See also:**

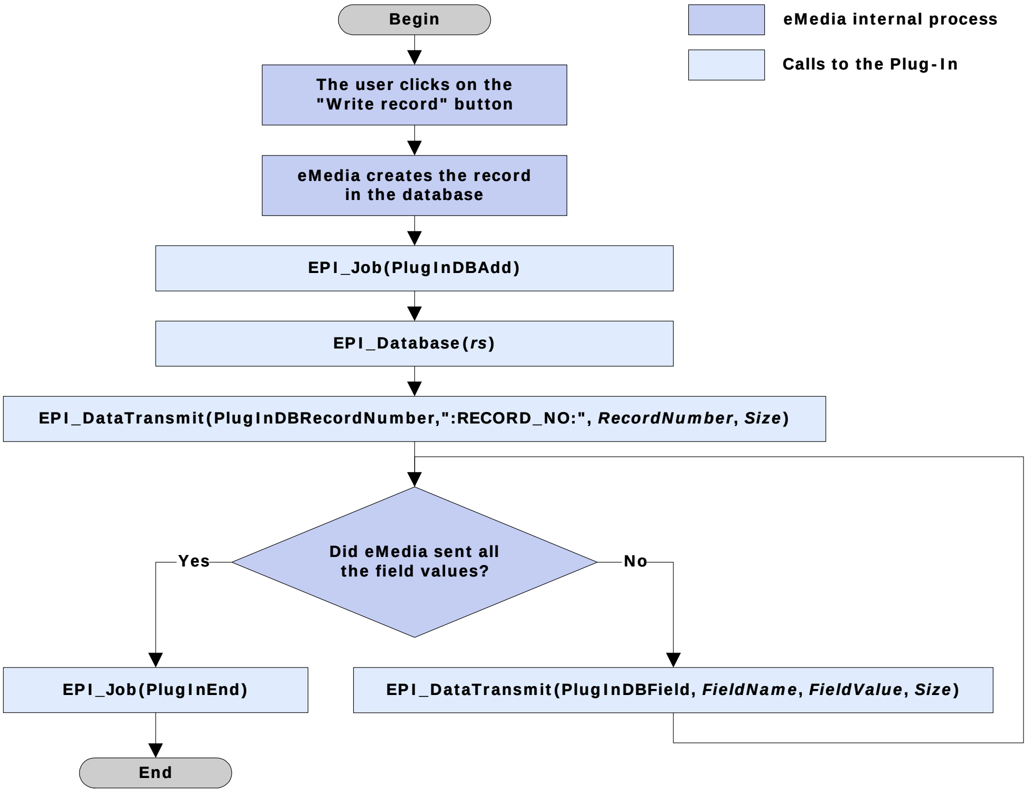
`EPI_Job` method

# FUNCTIONS CALLS

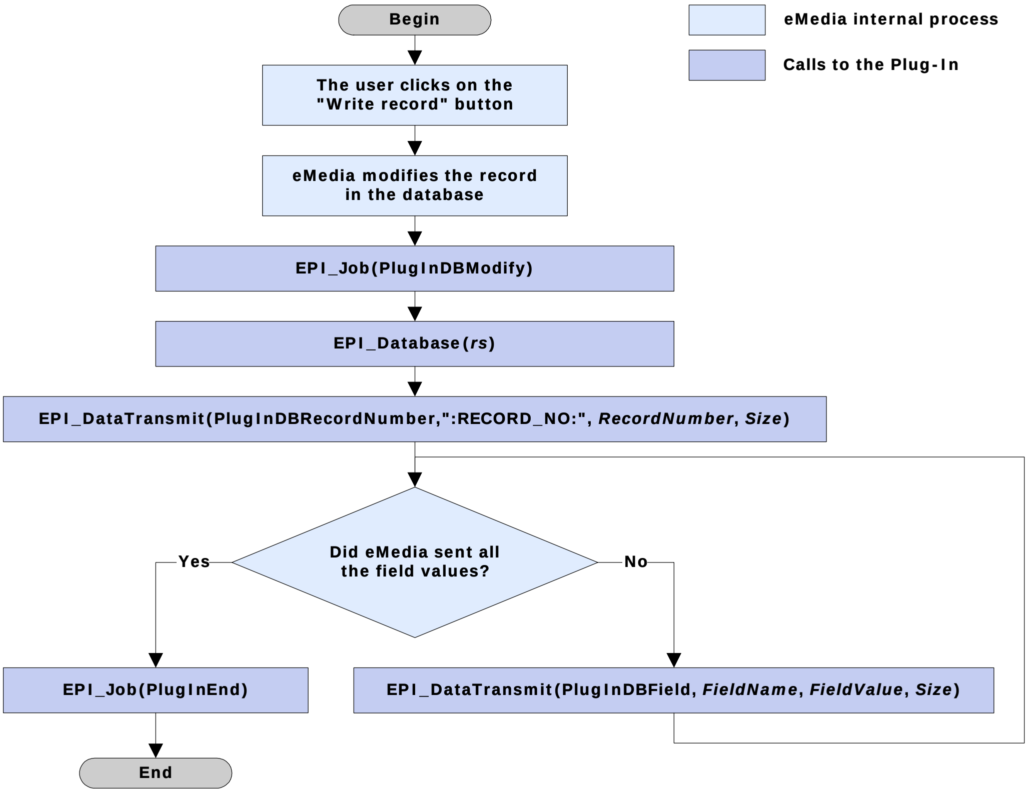
This section contains the calls scheme for any operation occurred in eMedia Card Designer notified to the plug-in.

All these calls are performed synchronously. eMedia Card Designer always hangs while the plug-in method or property is running.

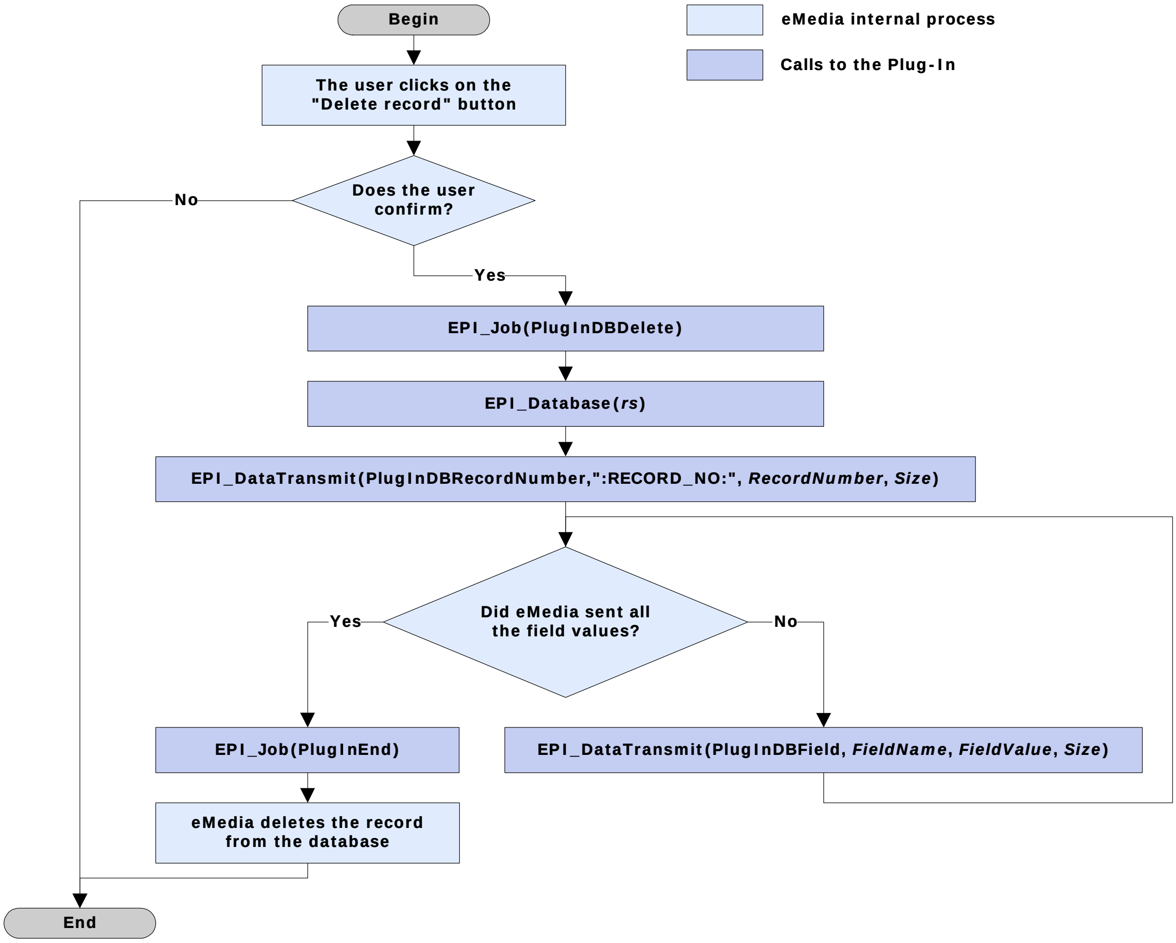
## Database record addition


Begin
The user clicks on the "Write record" button
eMedia creates the record in the database
EPI_Job(PlugInDBAdd)
EPI_Database(rs)
EPI_DataTransmit(PlugInDBRecordNumber,":RECORD_NO:", RecordNumber, Size)
Did eMedia sent all the field values?
Yes
No
EPI_Job(PlugInEnd)
EPI_DataTransmit(PlugInDBField, FieldName, FieldValue, Size)
End
eMedia internal process
Calls to the Plug-In

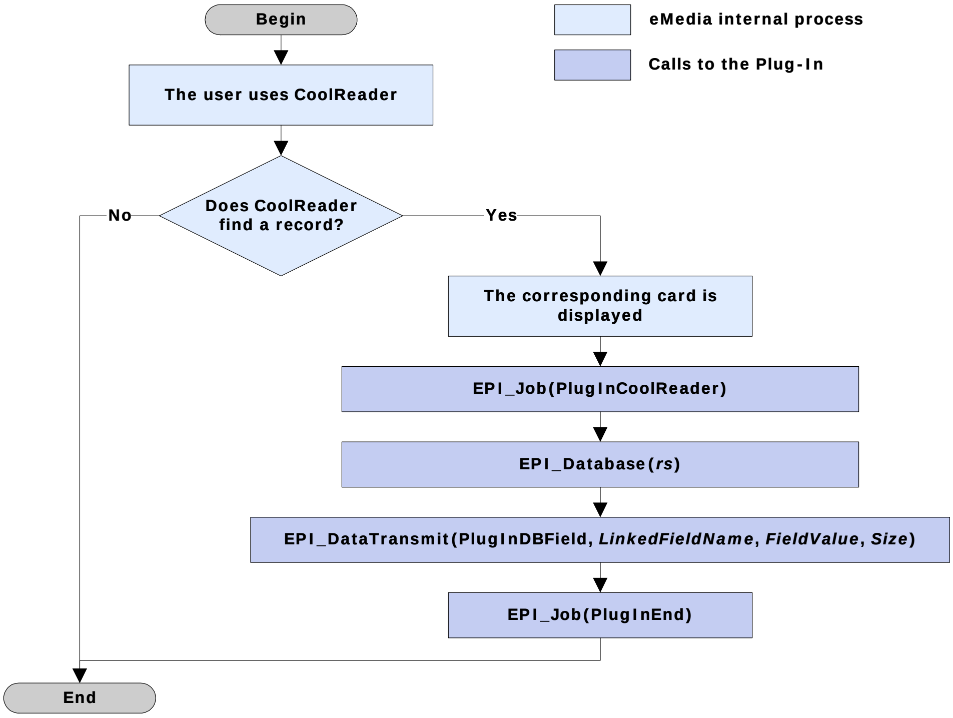
## Database record modification


Begin
The user clicks on the "Write record" button
eMedia modifies the record in the database
EPI_Job(PlugInDBModify)
EPI_Database(rs)
EPI_DataTransmit(PlugInDBRecordNumber,":RECORD_NO:", RecordNumber, Size)
Did eMedia sent all the field values?
Yes
No
EPI_Job(PlugInEnd)
EPI_DataTransmit(PlugInDBField, FieldName, FieldValue, Size)
End
eMedia internal process
Calls to the Plug-In

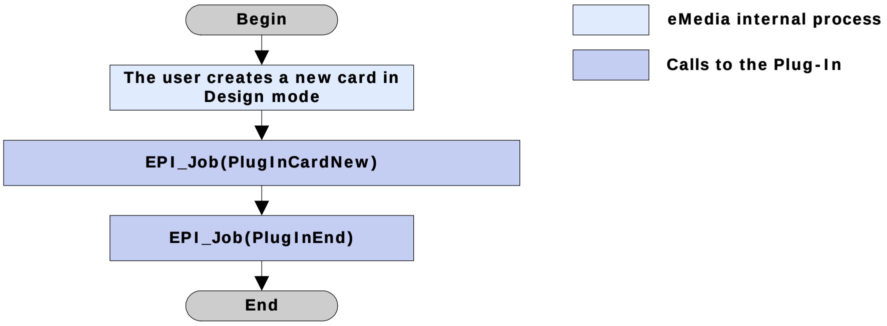
# Database record deletion


Begin
eMedia internal process
Calls to the Plug-In
The user clicks on the "Delete record" button
Does the user confirm?
No
Yes
EPI_Job(PlugInDBDelete)
EPI_Database(*rs*)
EPI_DataTransmit(PlugInDBRecordNumber,":RECORD_NO:", *RecordNumber*, *Size*)
Did eMedia sent all the field values?
Yes
No
EPI_Job(PlugInEnd)
EPI_DataTransmit(PlugInDBField, *FieldName*, *FieldValue*, *Size*)
eMedia deletes the record from the database
End

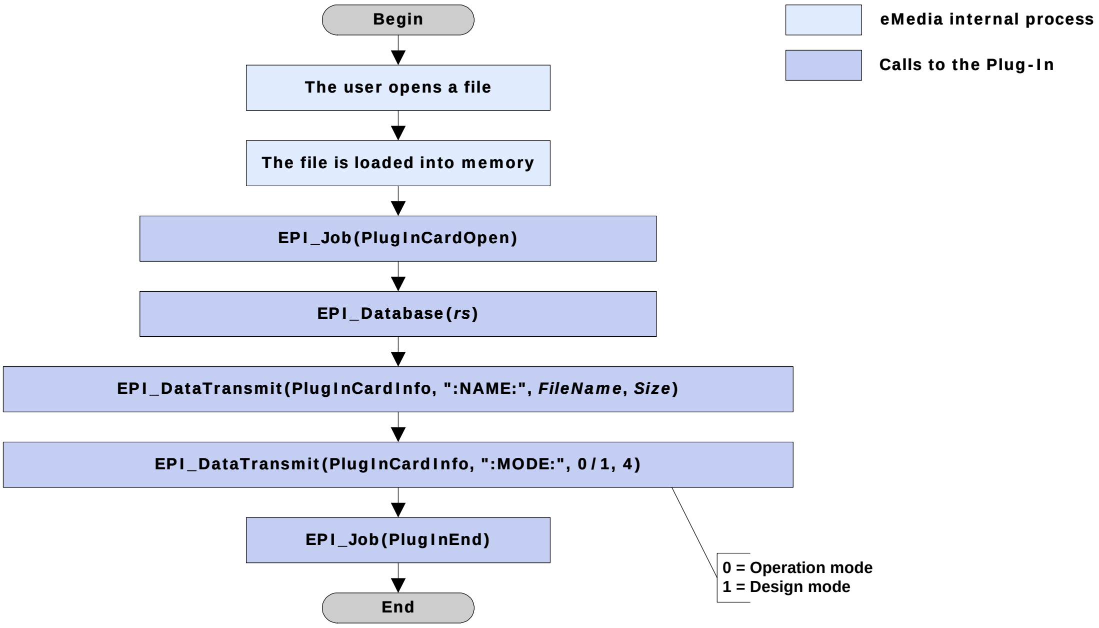
# CoolReader finds a record


Begin
eMedia internal process
Calls to the Plug-In
The user uses CoolReader
Does CoolReader find a record?
No
Yes
The corresponding card is displayed
EPI_Job(PlugInCoolReader)
EPI_Database(*rs*)
EPI_DataTransmit(PlugInDBField, *LinkedFieldName*, *FieldValue*, *Size*)
EPI_Job(PlugInEnd)
End

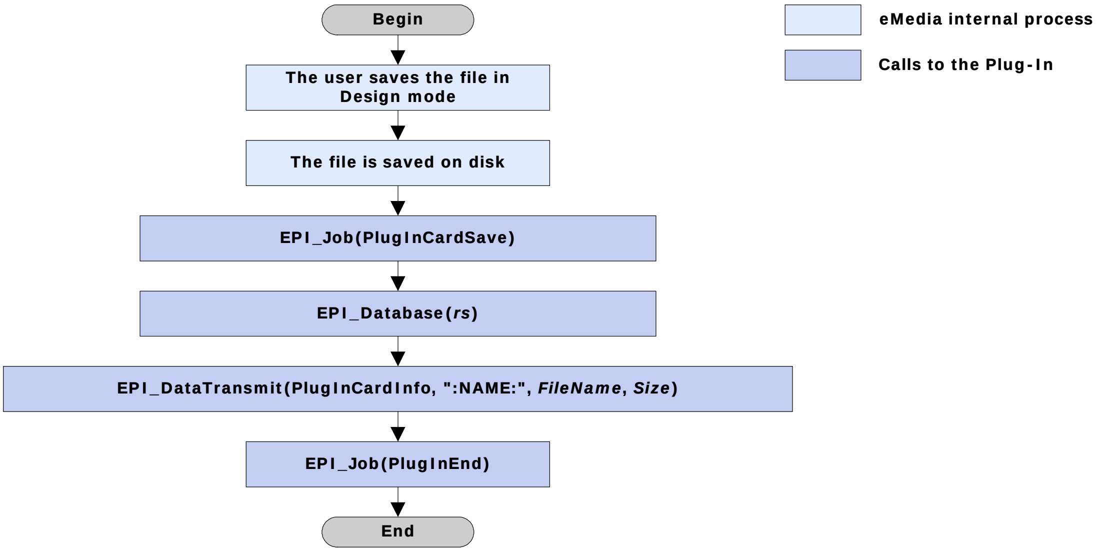
# New card template created



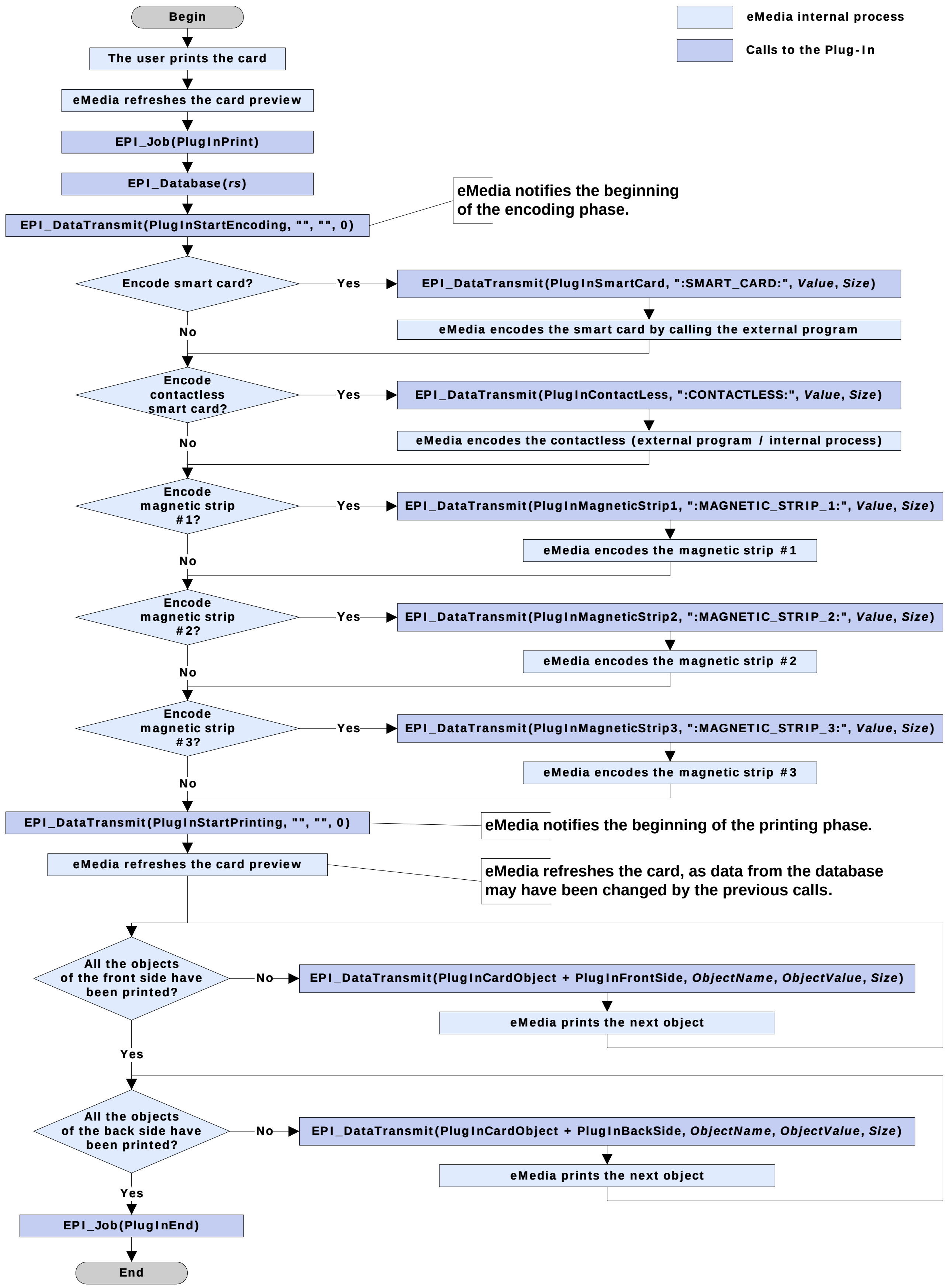
# Card template opened



# Card template saved


Begin
The user saves the file in Design mode
The file is saved on disk
EPI_Job(PlugInCardSave)
EPI_Database(rs)
EPI_DataTransmit(PlugInCardInfo, ":NAME:", FileName, Size)
EPI_Job(PlugInEnd)
End
eMedia internal process
Calls to the Plug-In

# Card printout

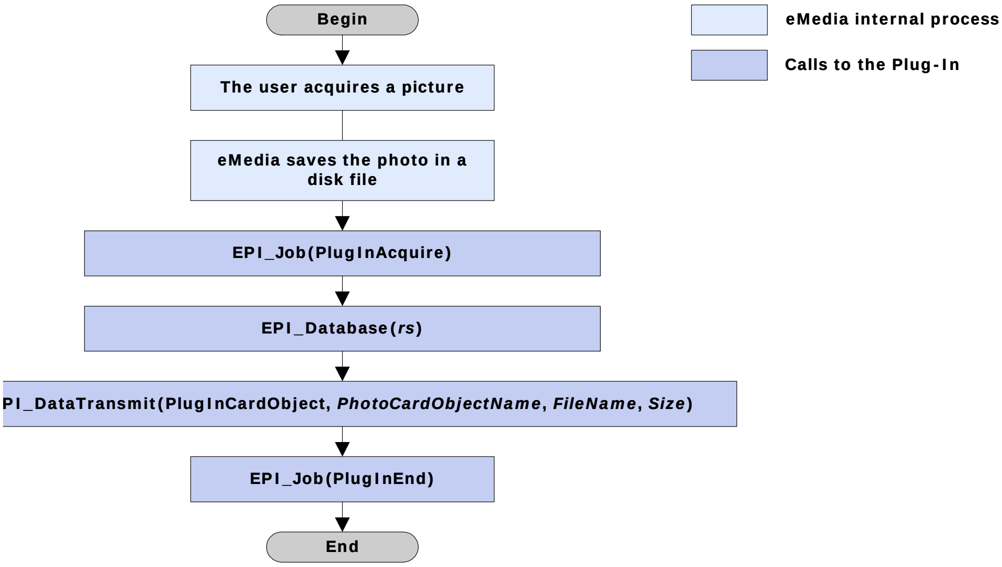

eMedia internal process
Calls to the Plug-In
Begin
The user prints the card
eMedia refreshes the card preview
EPI_Job(PlugInPrint)
EPI_Database(*rs*)
EPI_DataTransmit(PlugInStartEncoding, "", "", 0)
Encode smart card?
Yes
EPI_DataTransmit(PlugInSmartCard, ":SMART_CARD:", *Value*, *Size*)
eMedia encodes the smart card by calling the external program
No
Encode contactless smart card?
Yes
EPI_DataTransmit(PlugInContactLess, ":CONTACTLESS:", *Value*, *Size*)
eMedia encodes the contactless (external program / internal process)
No
Encode magnetic strip #1?
Yes
EPI_DataTransmit(PlugInMagneticStrip1, ":MAGNETIC_STRIP_1:", *Value*, *Size*)
eMedia encodes the magnetic strip #1
No
Encode magnetic strip #2?
Yes
EPI_DataTransmit(PlugInMagneticStrip2, ":MAGNETIC_STRIP_2:", *Value*, *Size*)
eMedia encodes the magnetic strip #2
No
Encode magnetic strip #3?
Yes
EPI_DataTransmit(PlugInMagneticStrip3, ":MAGNETIC_STRIP_3:", *Value*, *Size*)
eMedia encodes the magnetic strip #3
No
EPI_DataTransmit(PlugInStartPrinting, "", "", 0)
eMedia refreshes the card preview
All the objects of the front side have been printed?
No
EPI_DataTransmit(PlugInCardObject + PlugInFrontSide, *ObjectName*, *ObjectValue*, *Size*)
eMedia prints the next object
Yes
All the objects of the back side have been printed?
No
EPI_DataTransmit(PlugInCardObject + PlugInBackSide, *ObjectName*, *ObjectValue*, *Size*)
eMedia prints the next object
Yes
EPI_Job(PlugInEnd)
End

eMedia notifies the beginning of the encoding phase.

eMedia notifies the beginning of the printing phase.

eMedia refreshes the card, as data from the database may have been changed by the previous calls.

# Photo acquisition


Begin
The user acquires a picture
eMedia saves the photo in a disk file
EPI_Job(PlugInAcquire)
EPI_Database(rs)
PI_DataTransmit(PlugInCardObject, PhotoCardObjectName, FileName, Size)
EPI_Job(PlugInEnd)
End
eMedia internal process
Calls to the Plug-In

# PART II

# THE CARDOBJECTS CLASS REFERENCE

# REFERENCE

The `CardObjects` class is a public non-creatable class of eMedia Card Designer. As a public class, a client program or a server program can use its members, properties and methods; but, as a non-creatable class, no client program or server program is able to create a new instance of this class.

Your own software component (either a client program or a server program) can retrieve one instance of this class:

- In client programs (applications that take control of eMedia Card Designer through the `eMedia.Application` component), the `CardObjects` property of the `Application` class returns one instance of this class.
- In server programs (applications called by eMedia Card Designer when events occur – also known as plug-ins), the `EPI_GetCardObject` method is called by eMedia, providing as parameter one instance of this class.

Once the instance retrieved, you may use the methods of this class (no properties are available yet), applied to the retrieved object to obtain in-depth information on the currently opened card template opened in the software.

This section of the document describes these methods:

## Methods

| Name | Description | Build |
|---|---|---|
| **GetCardFilename** | Returns the current card template file name. | 630+ |
| **GetSides** | Returns the sides of the current card template. | 630+ |
| **GetCardBackgroundFile** | Returns the file name of the picture in the card background. | 630+ |
| **GetCardBackgroundPrint** | Returns if the background must be printed or not. | 630+ |
| **SaveBitmap** | Saves the picture in a Photo object or in an Image object on disk. | 630+ |
| **GetObjectName** | Returns the name of a card object. | 630+ |
| **GetObjectType** | Returns the type of a card object. | 630+ |
| **GetObjectProperty** | Returns the value of a property of a card object. | 630+ |
| **Request** | Reserved for future usage | 630+ |

## GetCardFilename

This method returns the current card template file name (.emc file). This name contains the drive (DOS notation) or the server name and the resource name (UNC notation), followed by the path name, the file name and the file extension (typically ".emc").

### Function declaration:

```
Public Function GetCardFilename() As String
```

### Parameters

None.

### Return value type:

```
String
```

### See also:

## GetSides

This method returns information about the sides of the current card template opened in eMedia Card Designer. The returned value may be one of the following values:

| Value | Description |
|---|---|
| 1 | The current card template only contains a front side. |
| 2 | The current card template only contains a back side. |
| 3 | The current card template is two-sided. |

### Function declaration:

```
Public Function GetSides() As Long
```

### Parameters

None.

### Return value type:

`Long`

### See also:

## GetCardBackgroundFile

This method returns the file name of the background picture. This name contains the drive (DOS notation) or the server name and the resource name (UNC notation), followed by the path name, the file name and the file extension.

### Function declaration:

```
Public Function GetCardBackgroundFile(ByVal Side As Long) As String
```

### Parameters

The `Side` parameter must contain the side of the card template for which the background picture file name is to be retrieved. The value may be 1 for front side or 2 for back side.

### Return value type:

`String`

### See also:

`GetCardBackgroundPrint` method

## GetCardBackgroundPrint

This method returns TRUE if the background picture must be printed, FALSE if it will not be printed.

### Function declaration:

```
Public Function GetCardBackgroundPrint(ByVal Side As Long) As Boolean
```

### Parameters

The `Side` parameter must contain the side of the card template for which the background picture file name is to be retrieved. The value may be 1 for front side or 2 for back side.

### Return value type:

`Boolean`

### See also:

`GetCardBackgroundFile` method

## SaveBitmap

This method saves in a disk file the picture contained in a Photo object, an Image object or in the card background. The parameters specify the saving options. If the picture cannot be saved, the function fires an error.

### Function declaration:

```
Public Sub SaveBitmap(ByVal ObjectSide As Long, _
                      ByVal ObjectName As String, _
                      ByVal BitmapWidth As Long, _
                      ByVal BitmapHeight As Long, _
                      ByVal FilePath As String, _
                      ByVal FileType As Long, _
                      ByVal FileBpp As Long, _
                      ByVal FileQuality As Long)
```

### Parameters

| | |
|---|---|
| `ObjectSide` | This parameter must contain the side of the card template containing the object to save. This value must be 1 for front side or 2 for back side. |
| `ObjectName` | This parameter must contain the name of the Photo object or Image object containing the picture. Specify "*" to save the card background. |
| `BitmapWidth` | Width of the resulting picture (in pixels). The class instance in eMedia will resize the picture (keeping the ratio) to this width before saving. Notice that the original picture in eMedia will not be changed. |
| `BitmapHeight` | Height of the resulting picture (in pixels). The class instance in eMedia will resize the picture (keeping the ratio) to this height before saving. Notice that the original picture in eMedia will not be changed. |
| `FilePath` | File name and path in which the picture will be saved. |
| `FileType` | Picture format. This value must be 1 to save as a Windows Bitmap file (.bmp) or 2 to save as a JPEG file (.jpg) |
| `FileBpp` | Color depth of the picture to save. This value must be one of the following:<br>FileType = 1 (Windows bitmap)<br>1......... Black & White<br>4......... 16 colors<br>8......... 256 colors<br>16....... Hi-color (65536 colors)<br>24....... True color<br>32....... True color 32 bits per pixel.<br>FileType = 2 (JPEG)<br>8......... Grayscale<br>24....... True color |
| `FileQuality` | Quality factor for JPEG pictures. This value is ignored if the picture is saved as a Windows bitmap. The value must be between 0 (worst quality) and 100 (best quality) |

### Return value type:

None.

If the function isn't able to save the picture, an error is fired.

### Examples:

In this example, the contents of the pIdentity object located on the front of the card will be saved in the c:\Pictures\Id.jpg file as a JPEG picture, true color, 85% quality. This file will contain a 300x400 pixels picture:

```
x.SaveBitmap 1, "pIdentity", 400, 300, "c:\Pictures\Id.jpg", 2, 24, 85
```

In this example, the background picture of the back side of the card will be saved in the C:\Pictures\Back1.bmp file as a Windows bitmap picture, true color. The bitmap file will have a size of 1016x648 pixels:

```
x.SaveBitmap 2, "*", 1016, 648, "c:\Pictures\Back1.bmp", 1, 24, 100
```

**See also:**

`GetCardBackgroundFile` method

## GetObjectName

This method returns the name (Name property) of an object on the card, from its position in the z-order (the position of the object in the z axis). The x and y axis locate the object from the top left corner of the card. The z axis represents the position of the object relatively to the other card objects. On your card, each object is on the top and/or on the back of the other ones. The objects on the top cover the objects on the back. You may change this order in eMedia Card Designer using the "Send to back" and "Bring to front" icons at the bottom of the left toolbar.

### Function declaration:

```
Public Function GetObjectName(ByVal Side As Long, _
                              ByVal ZOrder As Long) As String
```

### Parameters

The `Side` parameter must contain the side of the card template for which the background picture file name is to be retrieved. The value may be 1 for front side or 2 for back side.

The `ZOrder` parameter must contain the z-order of the object to retrieve, from 0 to the total count of card objects minus one.

### Return value type:

`String`: the name of the card object.

If no object can be found and returned, the function fires the error number 0x080050006: "No such object on card".

### See also:

`GetObjectType` method

## GetObjectType

This method returns the type of an object on the card as a string.

### Function declaration:

```
Public Function GetObjectType(ByVal ObjectName As String) As String
```

### Parameters

The `ObjectName` parameter must contain the name of the object.

### Return value type:

`String`: the type of the object:

Text
BarCode
Image
Photo
Rectangle
Circle
Line

If the object doesn't exist, the method fires the error number 0x080050006: "No such object on card".

### See also:

`GetObjectName` method

## GetObjectProperty

This function returns the value of one property of any object. This value is always returned as a string.

### Function declaration:

```
Public Function GetObjectProperty(ByVal ObjectName As String, _
                                  ByVal PropertyName As String) As String
```

### Parameters

The `ObjectName` parameter must contain the name of the object.

The `PropertyName` parameter must contain the name of the property for which you want to retrieve the value. See below for the full list of properties, object type per object type.

### Return value type:

`String`: the value of the property.

If the object doesn't exist, the method fires the error number 0x080050006: "No such object on card".

If the object property can't be found, the method fires the error number 0x080050008: "No such property: "*PropertyName*" for the object "*ObjectName*"".

### See also:

`GetObjectName` method

`GetObjectType` method

Objects properties

## Request

This function is reserved for future usage. Currently, it fires an error number 0x080050002: "Subfunction not available".

### Function declaration:

```
Public Function Request(ByVal lFunction As Long, _
                        ByVal sParams As String) As String
```

### Parameters

The `lFunction` parameter must contain a subfunction number.

The `sParams` parameter must contain a string for the subfunction.

### Return value type:

`String`: the return value of the subfunction.

Currently, the method always returns an error.

### See also:

# OBJECTS PROPERTIES

The tables below list, object type per object type the property names you can request from the `GetObjectProperty` method:

All objects

| Property name | Description |
|---|---|
| **Height** | Height of the object in mm. |
| **Left** | Left position in mm. |
| **Locked** | 0 if the object is unlocked, 1 if it's locked. |
| **Name** | Name of the object. |
| **Side** | Side of the card for the object: 1=front, 2=back. |
| **Top** | Top position in mm. |
| **Width** | Width of the object in mm. |

Text objects

| Property name | Description |
|---|---|
| **Alignment** | Text alignment (0 = left, 1 = right, 2 = center). |
| **ChoiceList** | Choice list if Source=6. |
| **Data** | Contents of the object. |
| **DBField** | Database field if Source=5 or 7. |
| **Expression** | Expression if Source=4. |
| **FontCharset** | Charset used. |
| **FontColor** | Color of the text. |
| **FontName** | Name of the font. |
| **FontSize** | Size of the font in pt. |
| **FontStyle** | Bit mask of font characteristics (b0 set for bold, b1: italic, b2: strikethrough, b3: underlined). |
| **Format** | Format string. |
| **Legend** | Caption text in the "Input" window. |
| **MultiLine** | 1 if multilined object, 0 otherwise. |
| **Rotation** | Angle of rotation in degrees. |
| **Source** | Source of datas:<br>0 ..... Fixed data<br>2 ..... User input<br>3 ..... Print counter<br>4 ..... Expression<br>5 ..... Database field<br>6 ..... Choice list<br>7 ..... Database choice. |
| **Type** | Always returns "Text" |
| **ToolTip** | Text of the tooltip for the "Input" window. |
| **Visible** | 1 if the object is visible, 0 otherwise. |
| **ZOrder** | Position on the z-axis. |

Barcode objects

| Property name | Description |
|---|---|
| **Alignment** | Alignment of the barcode (0 = left, 1 = right, 2 = center). |
| **BackColor** | Color of the barcode background. |
| **BarWidth** | Width of the smallest bars. |
| **ChoiceList** | Choice list if Source=6. |
| **Data** | Contents of the object. |
| **DBField** | Database field if Source=5 or 7. |
| **Direction** | Direction of the barcode:<br>0 .....Left to right<br>1 .....Right to left<br>2 .....Top to bottom<br>3 .....Bottom to top. |
| **Expression** | Expression if Source=4. |
| **Format** | Format string. |
| **Legend** | Caption text in the "Input" window |
| **Notches** | 1 if notches are present, 0 otherwise. |
| **Source** | Source of datas:<br>0 .....Fixed data<br>2 .....User input<br>3 .....Print counter<br>4 .....Expression<br>5 .....Database field<br>6 .....Choice list<br>7 .....Database choice. |
| **Style** | Style of the barcode:<br>0 .....None<br>1 .....2 of 5 interleaved<br>3 .....3 of 9<br>4 .....Codabar<br>5 .....3 of 9 extended<br>6 .....Code128-A<br>7 .....Code128-B<br>8 .....Code128-C<br>9 .....UPC-A<br>10 ...MSI Code<br>11 ...Code 93<br>12 ...Code 93 extended<br>13 ...EAN-13<br>14 ...EAN-8<br>16 ...ANSI 3 of 9<br>17 ...ANSI 3 of 9 extended<br>18 ...Code128<br>19 ...EAN-128<br>20 ...UPC-E<br>22 ...MSI Code (checksum)<br>51 ...2 of 5 (checksum)<br>53 ...3 of 9 (checksum). |
| **ToolTip** | Text of the tooltip for the "Input" window. |

Barcode objects

| Property name | Description |
|---|---|
| **Type** | Always returns "BarCode". |
| **ValueFontCharset** | Charset used for the value text. |
| **ValueFontName** | Name of the font of the value text. |
| **ValueFontSize** | Size of the font of the value text in pt. |
| **ValueFontStyle** | Bit mask of font characteristics (b0 set for bold, b1 set for italic, b2 set for strikethrough, b3 set for underlined). |
| **ValueVisible** | 1 if the value text is visible, 0 otherwise. |
| **ZOrder** | Position on the z-axis. |

Image & Photo objects

| Property name | Description |
|---|---|
| **AutoZoom** | 1 if the auto zoom feature is turned on, 0 otherwise. |
| **BorderStyle** | 1 if a frame is drawn around the picture, 0 otherwise. |
| **CropLeft** | Distance in pixels between the left border of the picture and the left border of the container. |
| **CropTop** | Distance in pixels between the top border of the picture and the top border of the container. |
| **FileNaming** | File naming scheme:<br>1 ..... Automatic<br>2 ..... Manual<br>3 ..... Rule |
| **FileRule** | Expression to use when saving the picture. |
| **Legend** | Caption text in the "Input" window |
| **Picture** | Picture file name. |
| **Rotation** | Rotation angle in degrees. |
| **Source** | Always returns 1 for a Photo object and 0 for an Image object. |
| **TwainSourceName** | Source of the acquisition. |
| **Type** | Always returns "Image" for Image objects or "Photo" for Photo objects. |
| **Zoom** | Zoom level in percent. |
| **ZOrder** | Position on the z-axis. |

Shape objects

| Property name | Description |
|---|---|
| **BackColor** | Color of the shape background. |
| **BackStyle** | 1 if the background is opaque, 0 if transparent. |
| **BorderColor** | Color of the shape border. |
| **BorderWidth** | Size of the border in dots. |
| **Rotation** | Rotation angle in degrees. |

Shape objects

| Property name | Description |
|---|---|
| **`Source`** | Always returns 0. |
| **`Type`** | Always returns "Rectangle" for Rectangle objects, "Circle" for Oval objects and "Line" for Line objects. |
| **`ZOrder`** | Position on the z-axis. |

# CODE SAMPLES

The example below stores in the dynamic array `TheObjects` all the object names on the current card template:

```
Sub RetrieveAllObjects(eMediaCardObjects As eMedia.CardObjects)
Dim lngSide As Long
Dim lngOrder As Long
Dim strObjectName As String
  On Error Resume Next
  Redim TheObjects(0)
  'Loop on the two sides of the card
  For lngSide = 1 To 2
    'Begin with lowest z-order
    lngOrder = 0
    Do Until Err.Number <> 0
      'Retrieve the next object name
      strObjectName = eMediaCardObject.GetObjectName(lngSide, lngOrder)
      If Err.Number = 0 Then
        'No error: object found, store it at the end of the array
        Redim Preserve TheObjects(UBound(TheObjects) + 1)
        TheObjects(UBound(TheObjects)) = strObjectName
      End If
      'Next object
      lngOrder = lngOrder + 1
    Loop
    Err.Clear
  Next lngSide
End Sub
```

The example below checks the contents of the text object on the card named tDepartment, then compares it to a list of possible departments. If the value is found, it saves the picture in the pIdentity photo object on the card in the folder having this department name, using the card holder name (stored in the tName text object on the card) as the file name:

```
Function StorePictureInDept(eMediaCardObjects As eMedia.CardObjects) As String
Dim strDeptType As String
Dim strDeptValue As String
Dim strPictType As String
Dim lngPictSide As Long
Dim strHolder As String
Dim strPath As String
Const DEPARTMENTS = "marketing,sales,management,production,"
  On Error Resume Next
  'Check if the object exists and get the type
  strDeptType = eMediaCardObjects.GetObjectType("tDepartment")
  If Err.Number <> 0 Then
    'Error: exit
    StorePictureInDept = "Cannot find the tDepartment object"
    Exit Function
  End If
  'Check object type
  If strDeptType <> "Text" Then
    StorePictureInDept = "tDepartment is not a Text object"
    Exit Function
  End If
  'Type if ok, get contents
  strDeptValue = eMediaCardObjects.GetObjectProperty("tDepartment", "Data")
  'Check contents
  If InStr(1, DEPARTMENTS, strDeptValue & ",", vbTextCompare) = 0 Then
    StorePictureInDept = "tDepartment doesn't contain a valid value"
    Exit Function
  End If
  'Department OK. Check if pIdentity exist, get its type and card side.
  strPictType = eMediaCardObjects.GetObjectType("pIdentity")
  If Err.Number <> 0 Then
    StorePictureInDept = "Cannot find the pIdentity object"
    Exit Function
  End If
  If strPictType <> "Photo" Then
    StorePictureInDept = "pIdentity is not a Photo object"
    Exit Function
  End If
  lngPictSide = CLng(eMediaCardObjects.GetObjectProperty("pIdentity", "Side"))
  'Get the holder's name
  strHolder = eMediaCardObjects.GetObjectProperty("tName", "Data")
  If Err.Number <> 0 Then
    StorePictureInDept = "Cannot find the tName object"
    Exit Function
  End If
  'Avoid an empty name
  If Len(strHolder) = 0 Then strHolder = "Unknown"
  'build the path for the picture to save
  strPath = "C:\Pictures\" & strDeptValue & "\" & strHolder & ".jpg"
  'Then save the picture
  eMediaCardObjects.SavePicture lngSide,"pIdentity",300,400,strPath,2,24,85
  'Finished
  StorePictureInDept = "Ok, picture is saved in """ & strPath & """"
  Err.Clear
End Sub
```