

TITLE:
SIEIODEV.DLL
DESCRIPTION
AND APPLICATION NOTES

Author: Bertocci Michele

DOCUMENT CODE:		
Dep: UAS	Project: Win+Drive	File: SIEIODEV.DOC Disk:
Author(s): Bertocci	Doc. Vers.: 01.000	Source: WORD
Approved:	Author:	Department manager:
		Date: 6/10/2002

Ind. Mod.	Date	Modify Description
da V..... a V.....	. . // ..	

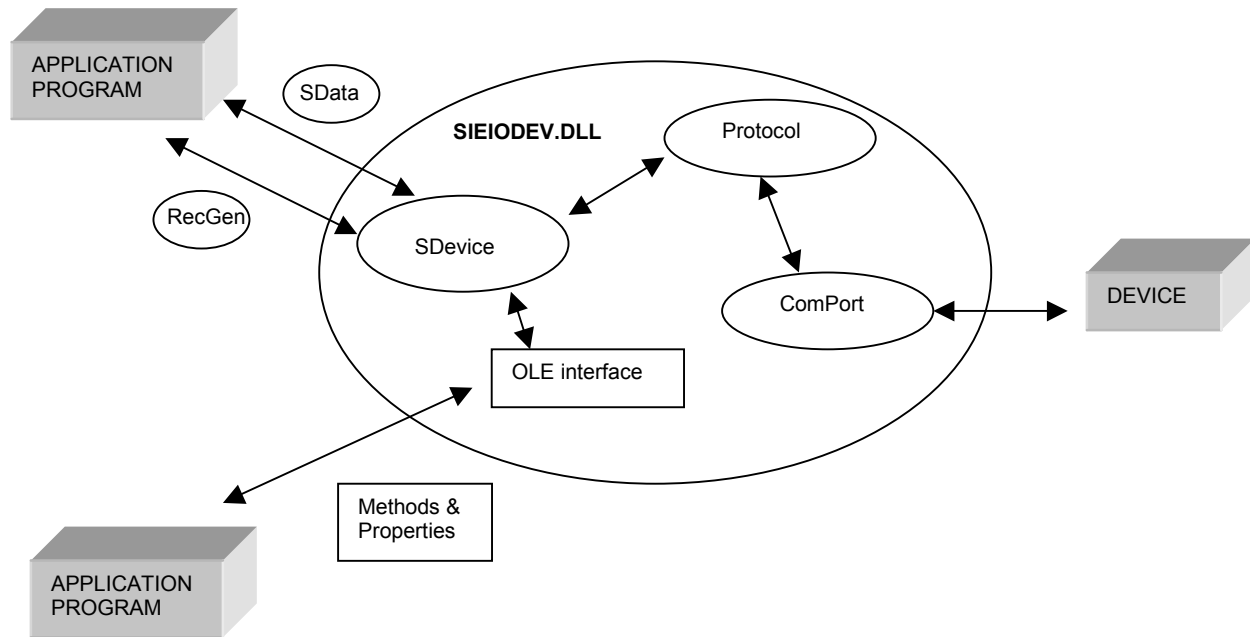
SIEIODEV.DLL

DESCRIPTION AND APPLICATION NOTES

1. GENERAL DESCRIPTION

SIEIODEV.DLL is a Win32 Dynamic Link Library (can be used on Windows NT and Windows 95 systems). It contains an object oriented C++ Application Program Interface (API) that makes possible to work and to communicate with the FlexMax drive. It also contains an OLE Automation interface that allows applications that support automation (Microsoft Office applications, Visual Basic and many others) to exchange data with the FlexMax drive.

The architecture of the DLL is the following:



2. C++ INTERFACE

The main objects involved in the API are **SDevice**, **Protocol**, **ComPort**, **Sdata**, and **RecUnkn**.

The application program talks with **SDevice** (which represents a FlexMax drive) and exchanges two types of data with it: **SData** and **RecUnkn**.

The class **SData** defines the basic data type exchanged with the FlexMax drive; an object of this class encapsulates the value, the FlexMax data type and the conversion routine to/from the C++ language data types.

The class **RecUnkn** is the base class for device data records. It manages the main characteristics of all types of a data record: identifier, index, type and data buffer.

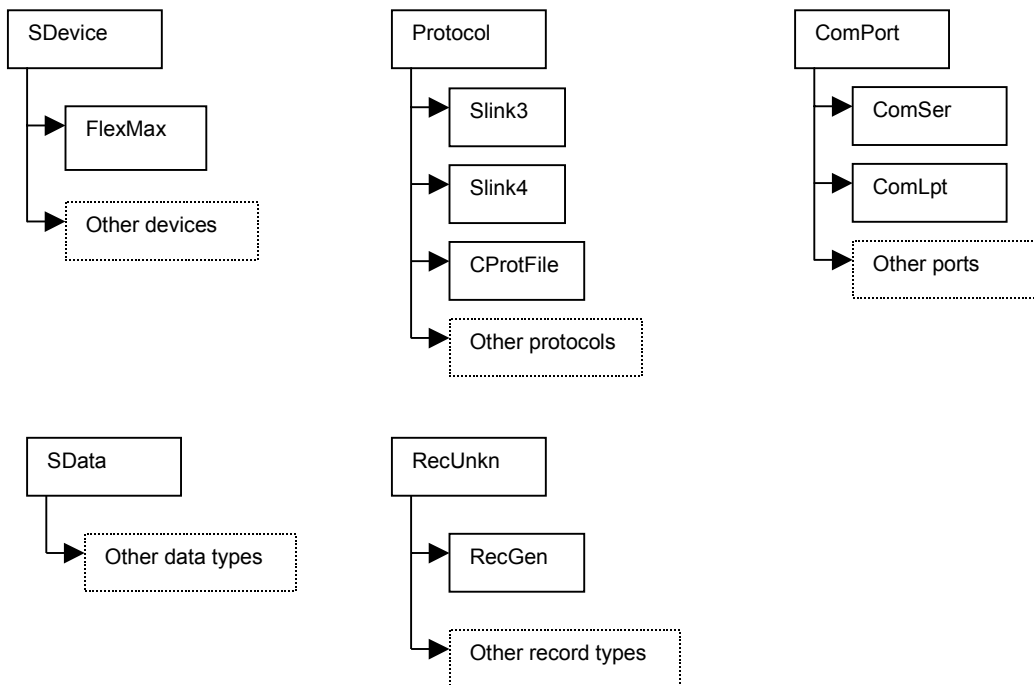
The class **RecGen** (derived from **RecUnkn**) represents the data record stored into the database of FlexMax drives. It is formed by several fields of type **SData** (value, minimum, maximum etc.) and some attribute fields (write permissions, record type, access level etc).

The communication between an object of class **SDevice** and the physical device is made through objects of class **Protocol** and **ComPort**.

The class **Protocol** defines a generic the FlexMax protocol with the basic communication functions. **ComPort** defines a generic communication port which can be used to link with the physical device.

These main classes are the root classes from which are derived more specialized classes needed to manage specific devices, protocols and communication ports.

The hierarchy is shown in the figure below.



2.1 CLASS DESCRIPTION

2.1.1 Class SData

The class **SData** defines the basic data type exchanged with a FlexMax drive. It contains two informations: the data type and its value.

The class members can be divided into the following groups: initialization, assignment and conversions.

The initialization is performed by the constructors; several constructors are available (one for every C simple data type). Every constructor assigns the value and the corresponding FlexMax type to the object.

For example the following code:

```
float x = 123.4F;
SData d (x);
```

assigns to the object **d** the value of 123.4 and the FlexMax data type of TD_FLOAT. A copy constructor is also available to perform a correct member initialization.

```
SData d1 (d);
```

Assignments can be performed using a two-step procedure:

```
float x = 123.4F;
SData d;

// d will have a value of 123.4 and the TD_FLOAT data type.
// The data type must be assigned before its value
d.SetType (TD_FLOAT);
d.SetSData (&x);
```

or using a direct assignment using the overload operators:

```
float x = 123.4F
SData d1;
SData d2;

// d (above example), d1 and d2 have the same value and type
d1 = x;
d2 = (float) 123.4;
```

To retrieve the value from an object of type SData, it's available the function **GetData** with all its overloads:

```
SData d1 (123.4F);
float y;
int c;

// y will have the value of 123.4
d1.GetData (&y);
// c will have the value of 123
d1.GetData (&c);
```

The function **SetType** can also be used to convert the actual data type of an object:

```
// d1 (example above) will have its type changed in TD_BYTE and  
// its value converted to 123.  
d1.SetType (TD_BYTE);
```

String assignments and conversion are also available:

```
SData d;
char str[10];

// d will have a value of 123.4 and the TD_FLOAT data type.
d.SetType (TD_FLOAT);
d.SetData ("123.4");

// str will contain the string "123"
d.SetType (TD_CHAR);
d.GetData (str);
```

2.1.2 Class RecUnkn

The class **RecUnkn** manages all record types that can be exchanged with a FlexMax. It's the base class for all devices records and has the following main characteristics:

lpa:	numeric record identifier
ldb:	index of the record in the device database
Trec:	record type
GetBuffer:	the function used to get the unformatted record data
LenBuffer:	the length of the record buffer

All the record exchange with the devices is performed using this base class. The access to the data fields of various record types is performed through the derived record classes.

2.1.3 Class RecGen

The class **RecGen** represents the parameter definition record stored into the database of FlexMax drives. Its main fields are:

Value:	actual parameter value
Min:	minimum value allowed for the parameter
Max:	maximum value allowed for the parameter
Def:	default (initialization) value for the parameter
Description information:	a readable parameter identifier and its measure unit
Access information:	a collection of information related to the operation allowed on the parameter (password, write permission, device status needed etc.).

A complete field description can be found on the specific device documentation.

2.1.4 Class SDevice

The class **SDevice** represents the generic FlexMax drive.

Objects of this class can be used to perform operations and data exchange common to all FlexMax drives. These operations are:

Parameter exchange by means of **RdPar** and **WrPar** member functions that can read and write device parameters in the form of **SData** objects.

Record exchange by means of **RdRec**, **RdRecRL** and **WrRec** member functions that can read and write device records in the form of **RecUnkn** objects.

Device identification by means of the **GetSlaveID**.

Protocol functions that allow assigning a particular protocol (**Protocol** objects and its derivations) to the device.

2.1.5 Class Protocol

The class **Protocol** represents the generic FlexMax communication protocol. It allows performing basic information exchange with FlexMax drives.

Typical **Protocol** functions are those that allows parameters and records exchange.

The functions **SetCom** and **GetCom** are to be used to assign the particular communication port needed to perform link with the device.

2.1.6 Protocol derived classes

The classes **Slink3** and **Slink4** manage the corresponding FlexMax serial protocols.

The class **CProtFile** makes possible to emulate a device using parameters files.

2.1.7 Class ComPort

The class **ComPort** defines the basic I/O functions of a communication port. Single byte and buffer I/O are available.

The class derived from **ComPort** (such as **ComSer**) performs also specific port settings, typical of each particular device.

2.2 TYPICAL C++ APPLICATION

The following example shows a typical use of the FlexMax communication API.

```
// The following lines of code make the initialization of
// a generic FlexMax drive in order to prepare it for the data
// exchange

    // Serial port object instance
ComSer ser;

    // Attach to COM1 serial port
ser.setcom (SER_COM1);

    // Line settings needed for a FlexMax drives: 9600,N,8,1
ser.setline (CBR_9600, 8, NOPARITY, ONESTOPBIT);

    // Slink3 protocol creation with port assignment
Slink3 sl3 (&ser);

    // Generic FlexMax drive creation with protocol assignement
SDevice device (&sl3);

//
// The following lines of code make a parameter reading
// and a parameter writing.
//

    // Application internal variable
float value;

    // SData object needed to exchange parameters
    // with the device
SData par;

    // Setting data type to TD_VOID allow to read the
    // parameter with the device original format
par.SetType (TD_VOID);

    // Reading of parameter 21102 of the device
device.RdPar (21102, par);

    // Storing of the parameter value in the floating point variable
    // If the parameter type was different from TD_FLOAT, an automatic
    // conversion is performed.
par.GetData (&value);

    // Perform some operations on the value and assign it
    // to the SData object as float
par = (float) (value * 1.4 - offset);

    // Changing of the parameter on the device
device.WrPar (21102, par);

    // Storing of the same value in a integer parameter on the device
```

```
par.SetType (TD_WORD);

// Storing the parameter on the device
device.WrPar (20186, par);
```

The following example shows a typical use of records classes.

```
float    val,
         min,
         max;

RecGen   rg;

// Read the record with identifier 20000.
// A RecGen object can be used with RdRec because it's derived
// from RecUnkn.
device.RdRec (20000, rg);

// Retrieve and print out data values
rg.Val.GetData (val);
rg.Min.GetData (min);
rg.Max.GetData (max);

printf ("Values:  actual %f  minimum %f  maximum %f\n", val, min, max);
printf ("Description %s", rg.Descr);
```

2.3 FILES NEEDED FOR C++ INTERFACE

The complete set of files necessary to work with SIEIODEV.DLL is the following:

SIEIODEV.DLL	is the dynamic link library that features all the FlexMax API for the devices. It must reside in the Windows system directory or in the application-working directory.
SIEIODEV.LIB	is the static library file that must be linked to the application program to interface with SIEIODEV.DLL.
SIEIODEV.TLB	is the OLE type library. It can be used only by C++ developers which wants to use the DLL as an In-Process OLE automation server.
SDATA.H	contains the definition of the class SData .
RECUNKN.H	contains the definition of the class RecUnkn .
RECGEN.H	contains the definition of the class RecGen .
SDEVICE.H	contains the definition of the class SDevice .
PROTOCOL.H	contains the definition of the class Protocol .
COMPORT.H	contains the definition of the class ComPort .
ERRO.H	contains the definitions of all error codes of FlexMax drives.
DGFC.H	contains the definition of the class Dgfc , a derivation of SDevice .
DGFCSTAT.H	contains the definitions of status, alarms etc. of the Dgfc device.
SLINK3.H	contains the definition of the class Slink3 , a derivation of Protocol .
SLINK4.H	contains the definition of the class Slink4 , a derivation of Protocol .
PROTFILE.H	contains the definition of the class CProtFile , a derivation of Protocol .
FILEPAR.H	contains the definition of the class CFilePar , the class that manages the parameter's files.
SIEICOM.EXE	is an application program that manages the serial communications ports. It allows sharing the access to a serial port between several applications. It's automatically invoked by SIEDISP.DLL when the access to a serial port is needed. SIEICOM.EXE automatically terminates when no more application programs need to use the serial port it manages.

SIEIODEV.DLL also needs that the two MFC (Microsoft Foundation Classes) DLLs: MFC42.DLL and MSVCRT.DLL are available to the system (in the Windows system directories or in the working directory).

3. OLE INTERFACE

SIEIODEV.DLL is an OLE Automation server. This feature allows all programs that support automation to communicate with FlexMax drives.

A typical example of these programs are the applications contained in Microsoft Office or object-based languages such as Microsoft Visual Basic.

The DLL implements an object named **SieiDev** that manages a simple communication with the FlexMax drives. Through its methods is possible to configure the communication port and the protocol and then to exchange parameters with the devices.

The object interface identifier is "SieiODev.SieiDev" and has the following characteristics:

Methods: Init
 Config
 EditPar
 ShowMsg

Properties: ParInt
 ParFloat
 MsgOn
 Success
 ErrorCode

3.1 Methods description

Init

The **Init** method must be invoked to initialize the **SieiDev** object before performing any communication procedure.

Syntax:

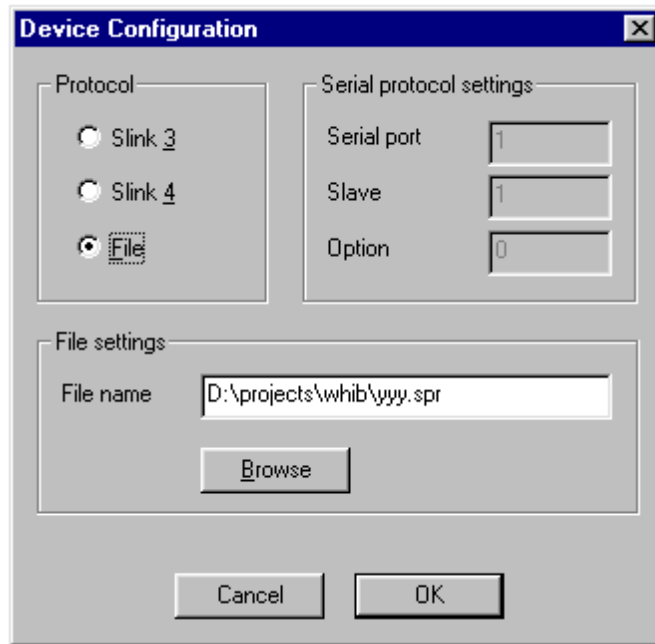
object.Init [filename]

filename is an optional string that indicates the pathname of the file where all configuration data are to be stored. The configuration data are automatically managed by the **Config** method.

Config

The **Config** method displays the configuration dialog-box shown below. All settings made in the dialog-box are stored in the configuration file (if present) specified with the **Init** method.

Syntax:
object.Config

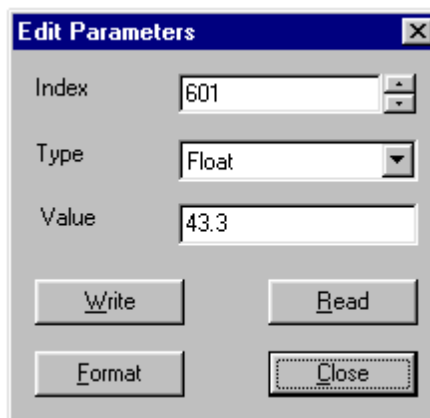


EditPar

The **EditPar** method displays the parameter dialog-box shown below. This dialog-box allows to read and write device parameters.

Syntax:
object.EditPar lpa

The *lpa* parameter indicates the identifier of the parameter to edit.



ShowMsg

The **ShowMsg** method displays a Windows message-box containing the description of the last error occurred during the communication.

Syntax:

object.**ShowMsg**

3.2 Properties description

ParInt

The **ParInt** property allows to read or set the integer value of a parameter.

Syntax:

object.**ParInt** (*lpa*) [= *value*]

The *lpa* parameter indicates the identifier of the parameter.

ParFloat

The **ParFloat** property allows to read or set the integer value of a parameter.

Syntax:

object.**ParFloat** (*lpa*) [= *value*]

The *lpa* parameter indicates the identifier of the parameter.

MsgOn

MsgOn is a BOOLEAN property that enables/disables the automatic issuing of error messages.

Syntax:

object.**MsgOn** = *value*

Success

Success is a BOOLEAN property that indicates the presence of an error during the last communication command executed.

Syntax:

object.**Success**

ErrorCode

ErrorCode is a numeric property that indicates the error code of last communication command executed.

Syntax:

object.**ErrorCode**

3.3 TYPICAL AUTOMATION MACRO

The following example shows the text of several macros written for Microsoft Excel in its macro language.

```
`
` Declaration of a variable used for
` device communication
`

Dim Dgfc

`
` This is a macro used to initialize the object SieiDev
`

Sub Init()

    ` Creation of the object
    Set Device = CreateObject("SieiODev.SieiDev")

    ` Initialization of object with configuration file
    ` XLSDEV.CNF
    Dgfc.Init "XlsDev.cnf"

    ` Activation of configuration dialog-box
    Dgfc.Config

    ` Enable error messages
    Dgfc.MsgOn = 1

End Sub

`
` This macro reads the parameter 21102 and puts its value
` in the active Excel cell.
` If the communication fails, a default string is stored in the cell
`

Sub GetPar21102()

    ` Read the parameter
    par = Dgfc.ParFloat(21102)

    ` Test if the communication failed
    If Dgfc.Success Then
        ` Cell with the value read
        ActiveCell.Value = par
    Else
        ` Cell with the error value
        ActiveCell.Value = "#####"
    End If

End Sub
```

```
`  
` This macro puts active cell value in the parameter 21102 of the device.  
`
```

```
Sub SetPar21102()  
  
    Dgfc.ParFloat(21102) = ActiveCell.Value  
  
End Sub
```

```
`  
` This macro activate the edit parameter dialog-box.  
` The parameter index is retrieved from the value of the active cell.  
`
```

```
Sub EditPar()  
  
    Dgfc.EditPar ActiveCell.Value  
  
End Sub
```

```
`  
` This macro reads 15 parameters starting from parameter 30108 and  
` puts their values in the selected column.  
`
```

```
Sub GetPars()  
  
    ` Number of the column  
    col = ActiveCell.Column  
  
    ` Reading loop  
    For i = 1 To 16  
        Cells(i, col).Value = Dgfc.ParFloat(30108 + i)  
    Next  
  
End Sub
```

3.4 FILES NEEDED FOR OLE INTERFACE

The complete set of files necessary to work with SIEIODEV.DLL is the following:

- SIEIODEV.DLL** is the dynamic link library that features all the SIEI API for the devices. It must reside in the Windows system directory or in the application-working directory.
- SIEICOM.EXE** is an application program that manages the serial communications ports. It allows sharing the access to a serial port between several applications.
It's automatically invoked by SIEDISP.DLL when the access to a serial port is needed.
SIEICOM.EXE automatically terminates when no more application programs need to use the serial port it manages.
- SIEIDF.STR** is the file containing the set of strings used to show the messages and the status of the devices in the dialog-boxes of the OLE interface.

SIEIODEV.DLL also needs that the two MFC (Microsoft Foundation Classes) DLLs: MFC42.DLL and MSVCRT.DLL are available to the system (in the Windows system directories or in the working directory).