KB Name	eWON binary data files format		
Type	Binary data format		
Since revision	NA		
KB Number	KB-0043-0	Build	14
Mod date	25. May. 2011		

Knowledge Base Information

eWON binary data files format

In eWON, you can export some files in binary format.

ircall.bin Historical recording of tags

inst_val.bin Instant values of tags

IMPORTANT **FIRMWARE VERSION:**

Pay attention that data formats changed a little with the eWON firmware 6 due to the new TagType (Integer and DWord).

see Backward Compatibility chapter.

1 ircall.bin

The ***ircall.bin*** file contains the binary values of all recorded Tags defined in the eWON. This file is an image of the Tag memory of the eWON.

This file is present on every eWON type, but this file is empty if the eWON cannot record data (eWON500, eWON2xxx).

IMPORTANT

The processor of the eWON uses the Big Endian format for memory access (most significant byte first) like Motorola processor. Thus, this access method is also applicable for the *ircall.bin* file. In PC world (Intel processor), the access method is Little Endian! → We need to reverse all bytes read (and words if necessary) from files in order to correctly interpret it in a PC program.

1.1 Header structure

The IrcAll.bin file begins by a Header structure.

Firmware Major #16	Firmware Minor #16
unused #16	Record Size #16

This structure contains 4 short integers (16bits): the firmware version of the eWON (Major and Minor), an unused data (Dummy) and the length of the record structure (RecordSize).

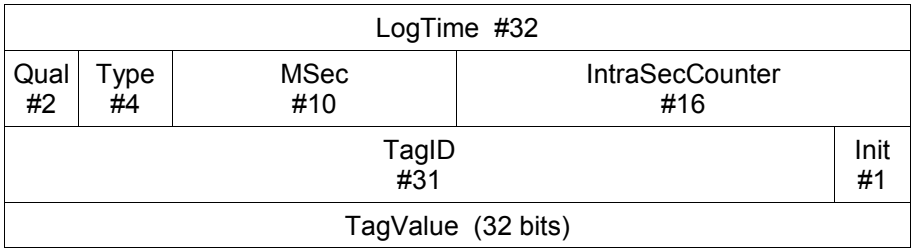
Example of implementation in C++:

```
typedef struct
{
   unsigned short VersionHi;
   unsigned short VersionLo;
   unsigned short Dummy;
   unsigned short StructLen;
}
HistoricalHeader_t;
```

If your eWON runs firmware version 6.1, the Header will be 6 1 0 16.

1.2 Record structure (since firmware 6)

After the 8 bytes of the Header, the eWON data can be found. Each record is encoded in a 16 bytes structure defined as follows:



Field name	Field description
LogTime (32 bits)	Absolute time in second (since 1970) for the record
Quality (2 bits)	Quality of the Tag (0:Bad 1:Uncertain 3:Good)
TagType (4 bits)	Tag type (0:Boolean 1:Float32 2:Integer32 3:unsignedInteger32)
MSec (10 bits)	Set to 0. *deprecated: This was the number of MSec to add to LogTime in order to get the complete timestamp of the record.*
IntraSecCounter (16 bits)	This value is incremented for each point logged during the same second (incremented even if TagId is different).
TagID (31 bits)	Unique Id of the Tag (== value in Var_lst.txt), never the same for 2 tags, even if tag deleted
InitValue (1 bits)	TRUE if the point was log due to a restart of the system
TagValue (32 bits)	Actual value logged at the beginning of the interval Coded as Float32, as Integer32 or as Unsigned32.

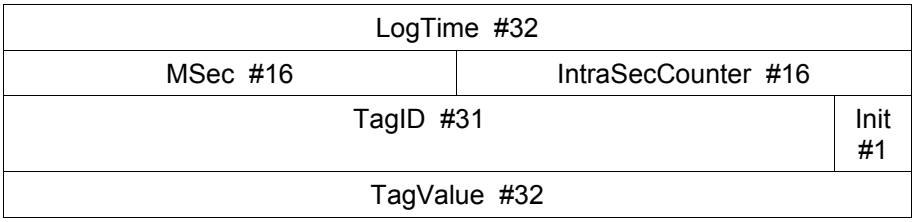
Example of implementation in C++:

```
typedef struct
{
  time_t              LogTime;
  unsigned short      IntraSecCounter;
  unsigned short      MSec:10;
  unsigned short      Type:4;
  unsigned short      IrcQuality:2;
  unsigned int        InitValue:1;
  unsigned int        TagId:31;
  float               TagValue;
}
HistoricalRecordV6_t;
```

Note: Due to the BigEndian to LittleEndian swap of bytes *and Words*, the structure elements are in reverse order.

1.3 Record structure (before firmware 6)

After the 8 bytes of the Header, the eWON data can be found. Each record is encoded in a 16 bytes structure defined as follows:



Field name	Field description
LogTime (32 bits)	Absolute time in second (since 1970) for the record
MSec (16 bits)	Set to 0. *deprecated: This was the number of MSec to add to LogTime in order to get the complete timestamp of the record.*
IntraSecCounter (16 bits)	This value is incremented for each point logged during the same second (incremented even if TagId is different).
TagID (31 bits)	Unique Id of the Tag (== value in Var_lst.txt), never the same for 2 tags, even if tag deleted
InitValue (1 bits)	TRUE if the point was log due to a restart of the system
TagValue (32 bits)	Actual value logged at the beginning of the interval Coded as Float32, as Integer32 or as Unsigned32.

Example of implementation in C++:

```
typedef struct
{
  time_t              LogTime;
  unsigned short      IntraSecCounter;
  unsigned short      MSec;
  unsigned int        InitValue:1;
  unsigned int        TagId:31;
  float               TagValue;
}
HistoricalRecord_t;
```

Note: Due to the BigEndian to LittleEndian swap of bytes *and Words*, the structure elements are in reverse order.

2 inst_val.bin

The inst_val.bin file contains the current values of all tags defined in the eWON.

2.1 Header

It will begin with a header of 20 bytes

Revision #32
RecordSize #32
NumberOfTag #32
RecFlag #32
Reserved #32

Field name	Field description
Revision (32 bits)	Revision of the inst_val file 1: before firmware6 2 and above: since firmware6
RecordSize (32 bits)	Size of the Record structure representing each tags information
NumberOfTags (32 bits)	Number of tags recorded in inst_val file
RecFlag (32 bits)	Internal use
Reserved (32 bits)	Internal use

Example of implementation in C++:

```
typedef struct
{
  int   Rev;
  int   RecSize;          //Record size
  int   NbTag;            //Number of tag exported
  int   RecFlag;
  int   Reserved;
}
InstantValueHeader_t;
```

2.2 Record

After the header, each eWON tags information will be represented in the following structure:

TagId #32	
TagValue #32	
AlarmStatus #32	
AlarmType #32	
TagType #16	TagQuality #16

Field name	Field description
TagId (32 bits)	ID of the tag
TagValue (32 bits)	Value of the tag. Coded as Float32, as Integer32 or as Unsigned32 depending of the TagType
AlarmStatus (32 bits)	Status of the alarm of the tag 0: none 1: pretrigger 2: alarm 3: acknowledged 4: return to normal
AlarmType (32 bits)	type of the alarm 0: none 1: high 2: low 3: level 4: high_high 5: low_low
TagType (16 bits)	Tag type 0: Boolean 1: Float32 2: Integer32 3: UnsignedInteger32
TagQuality (16 bits)	The quality used in the eWON is based on the quality defined by the OPC Foundation. It consists of a 16-bit value where • Bits 15-8 are vendor-specific quality information • **Bits 7-6** represent the **major quality (0:*bad* / *1:uncertain* / *3:good*)** • Bits 5-2 represent the sub-status • Bits 1-2 represent the limit status

note: On eWON before firmware6, the last 32bits of the structure were *Unused*.

Example of implementation in C++:

```
typedef struct
{
  int               TagId;
  float             TagValue;
  int               AlStatus;
  int               AlType;
  unsigned short    Quality;
  unsigned short    Type;
}
InstantValueRecord_t;
```

Note: Due to the BigEndian to LittleEndian swap of bytes and Words, the structure elements composing a doouble word (32bits) are in reverse order.

3 Backward Compatibility

For backward compatibility, eWON with firmware 6 (and above) can be force to produce files in the same format as before FW6.

For that, set the PreRev6Compat parameter to 1 in the config.txt file.

```
PreRev6Compat:1
```

This parameter applies on files defined by the ExportBlocDescriptor:

- $dtHL $ftB
- $dtHL $ftT
- $dtIV $ftT