Tagging Guidelines for 1-Wire Sensors and Instruments

Revision 1.00
September 1, 1999

by David Smiczek

Table of Contents

Abstract

Each 1-Wire device that Dallas Semiconductor builds is assigned a 64 bit 1-Wire Network Address number. The number is laser engraved into the read-only memory of the device. Dallas Semiconductor manages this number pool of 10^{19} codes so that each device has a unique number.

Once these 1-Wire devices leave Dallas Semiconductor, they go to a service provider. A service provider usually associates these 1-Wire Network Address numbers with a physical object. The association of the number to the physical object is often placed in some kind of database. With the introduction of more complex 1-Wire devices that can perform sophisticated sensing and reading operations, this task becomes more critical. This is further complicated by the desire to group 1-Wire devices and sensors together into a cluster to perform a group function.

To easily describe these associations and operations, the following document will present a *1-Wire TAG* format. This *1-Wire TAG* can reside in a traditional database or it can even be placed in the memory of a 1-Wire device. By carrying the *1-Wire TAG* with the sensor cluster, it can be self-configuring when presented to a new master application. If it is impractical or not desired to keep the *1-Wire TAG* with the cluster, it can also reside on a home Internet site. The master application can then take one of the unique 1-Wire Network Address numbers found on the 1-Wire to request the appropriate *1-Wire TAG* from the hosting site.

Dallas Semiconductor plans on maintaining the standard for the *1-Wire TAG* and making it accessible at ‘www.1-Wire.net’. There are also plans under way to provide hosting services for the *1-Wire TAG* data on the same site.

While Dallas Semiconductor is promoting this format, the content and version control of the *1-Wire TAGs* will be under the authority of the 1-Wire service provider.

1-Wire TAG Specification

The *1-Wire TAG* file specification as defined here can reside in a file on the 1-Wire sensor itself (physical) or in another file system such as on the master (local) or in a database on a server (remote). When the file is located in the memory of a 1-Wire device in the 1-Wire Cluster (physical) it must be in the form of an *Extended File Structure* file. The file can reside in any normal memory 1-Wire device such as the DS1993, DS2406, or DS2433. The *Extended File Structure* is the format used by the TMEX software drivers. The specification for the Extended File Structure can be found in the Dallas Semiconductor Application Note 114, "TMEX Extended File Structure Revision 3.10".

The contents of the *1-Wire TAG* file can have two different formats. The first type is the *Basic* format and contains a predefined format with fields describing the manufacture and sensor type. The Basic type of *1-Wire TAG* format provides self registering capabilities to known sensor types and configurations. The second type is the *Descriptor* and contains the description in the *Object-Based Messaging* format. *Object-Based Messaging* was developed by Dallas Semiconductor for network administration of 1-Wire enabled controllers. It also works well as a general purpose and flexible tagging language. The specification for *Object-Based Messaging* or (OBM) can be obtained from Dallas Semiconductor. The *Descriptor* type *1-Wire TAG* provides self-registering capabilities to even unknown configurations of sensor clusters.

Whenever a memory 1-Wire device is found by a *1-Wire TAG* compatible application it is examined to see if it has a *Basic* tag file TAGB.000 or a *Descriptor* tag file TABD.000. The contents are then parsed to obtain the available sensor characteristics.

Basic TAG

Overview

The *Basic 1-Wire TAG* contains twenty-eight bytes of predefined fields in the *Extended File Structure* file TAGB.000. Note that since the data is only twenty-eight bytes, the entire contents fits in one standard page of a memory 1-Wire device. There are seven fields defined. The first two fields are four bytes each and the next five fields begin with a ASCII $. See **Figure 1** for a general field map for the *Basic* tag file. With the exception of the *Binary Serialization* field, all codes are in ASCII text.

Structure

Figure 1. Basic *1-Wire TAG* field map

Byte 0 - 3	Byte 4 - 7
pppp Part ID (4 bytes)	**ssss** Binary serialization (4 bytes)

Byte 8 - 12	Byte 13 - 16	Byte 17 - 19	Byte 20 - 22	Byte 23 - 27
$mmmm Manufacturer code (5 bytes)	**$fff** Module function code (4 bytes)	**$ii** Interface type (3 bytes)	**$rr** Hardware revision (3 bytes)	**$dddd** Date code (5 bytes)

Table 1 lists the currently defined *Part ID*'s (bytes 0 to 3). The *Part ID* will inform the host as to the function of the attached sensor.

Table 1. Defined Part ID's (Byte 0 - 3 in Basic *1-Wire TAG*)

Part ID	**Description**
VOLT	Voltage Probe
COND	Conductivity Probe
PHPH	PH probe
PRES	Pressure Probe
SCAL	Weigh Scale
FORC	Load-cell and strain-gage Probe
SKEY	Security KEY (keys can also exist in the serialization field of any DS2407 in the net)
GDIS	GAP LED display panel
7DIS	7(9) Segment Display
CNTR	Rain gage
DSWS	Dallas Semiconductor Weather station

The *Binary serialization* (bytes 4 to 7) is a four byte least-significant byte first binary number. It represents the device number of type *Port ID*.

The *Manufacture code* (bytes 8 to 12) represent the manufacturer of the sensor. A manufacture code beginning with '#' indicates a FCC Grantee Code.

The *Modular function type* specifies the type of module. A module function type of 'ASS' indicates that this module is associated with another module to form a complete sensor. Spaces (hex 20) indicate that this module is not associated with another module.

The *Interface type* indicate that type of interface that this sensor employs.

The *Hardware revision* is a two character hexadecimal ASCII value that indicates the hardware revision of the sensor.   The value $00 is the first revision up to $FF giving a total of 255 possible revisions.

The *Date code* (bytes 23 to 27) indicate the date the sensor was tagged. The data code is a simple four digit ASCI code where the first two digits indicate the year and the last two indicate the week number (1 to 52).   If the year is greater than or equal to 70 the century is 1900 and if it is less than 70 the century is 2000.   For example, a year code of '02' indicates 2002 and a year code of '98' indicates 1998.

If any of the ASCII fields are not used, place spaces (hex 20) in the field contents. For example a non-used *Interface type* field would be '$(space)(space)'.   If a *Part ID* or other field type is not defined by this specification and is required, please register this type with Dallas Semiconductor by emailing to iButton.Support1@dalsemi.com.

Any data supplied in addition to the *Basic* or *Descriptor 1-Wire TAG* files, such as in another file, or outside of the file structure is vendor specific and considered outside of this specification. Note that the *Descriptor* type file provides a richer avenue for supplying additional information.   Both *1-Wire TAG* files may be provided in the same memory 1-Wire device to meet the needs of all hosts.

Descriptor TAG

Overview

The *Descriptor 1-Wire TAG* file (TAGD.000) contains data in the OBM (*Object Based Messaging*) format. Please refer the to the OBM (*Object Based Messaging*) documentation from Dallas Semiconductor to get a detailed format description. In brief, OBM consists of ‘Group’ and ‘Data’ objects. These objects are in a format that can be easily parsed. ‘Group’ objects perform an action and ‘Data’ objects read or set the contents of a memory block (Message Information Block or MIB). The action performed due to a ‘Group’ object can be performed at the beginning of the object, at the end, or both. With this flexible format, a sensor reading application can read and interpret previously unknown sensor cluster architectures.

Table 2 lists the group objects that are defined for this specification. Each of the group objects have required and optional data objects. If the group is a (PRE) or (PRE and POST) execute type object then the data objects must be before the beginning of the group. If the group object is a POST execute type object then the data objects must be after the beginning and before the end of the group object. This ordering of the objects allows the 1-Wire cluster of sensor to be read while the *1-Wire TAG* is being parsed. An application could even use the *1-Wire TAG* as a pseudo-script.

Table 2. Summary of group objects for *Descriptor* TAG.

Group Object	**Description**	**Required Data Objects**	**Optional Data Objects**	**Pre / Post**
ParseData	Group designating OBM parse-able data	none	All	POST
OWCluster	Logical grouping of 1-Wire devices	ClusterNum ClusterRev	Description Manufacturer ManufacturerCode Enum SecondsSince1970 NetRegistration	PRE POST
OWBranch	1-Wire branch device for complex topologies	OWNetAddress AccessMethod ChannelMask ChannelState	Description	PRE POST
OWSensor	1-Wire sensor, can be read	OWNetAddress AccessMethod ChannelMask* ChannelState* InitState*	Description ScaleFactorM ScaleFactorD UnitName	POST
OWActuator	1-Wire actuator, can be set to control something	OWNetAddress AccessMethod ChannelMask* ChannelState* InitState*	Description	POST

***** Not required for all AccessMethods. See the description of the AccessMethods for details.

Table 3 lists the data objects that are defined for this specification. These objects are used to set memory locations in the Memory Information Block (MIB). Group objects use the data locations in the MIB when performing actions.

Table 3. Summary of data objects for *Descriptor* TAG.

Group Object	**Description**	**Type**	**Max size (bytes)**
Description	General description	String	255
OWNetAddress	1-Wire Network node address	Binary Array	8
Manufacturer	Manufacturer Name	String	255
ManufacturerCode	Manufacturer Code	Binary Array	8
Enum	General Enumeration	Integer	4
SecondsSince1970	Time/date stamp expressed as seconds since January 1, 1970	Integer	4
AccessMethod	Access method to read/set 1-Wire sensor	Integer	4
ChannelMask	Channel mask for sensors with multiple channels	Integer	4
ChannelState	Desired channel state	Integer	4
ScaleFactorM	Multiplication scaling factor	Integer	4
ScaleFactorD	Division scaling factor	Integer	4
UnitName	General unit name	String	255
ClusterNum	1-Wire Cluster unique number	Binary Array	255
ClusterRev	1-Wire Cluster revision number	Integer	4
InitState	Initialization state for 1-Wire Sensor	Binary Array	255
NetRegistration	WWW network address for remote data services	String	255

By convention, when printing the OBM based tag in human-readable form, a 'Group' group is designated by pre-pending a 'G:' and a 'Data' object by a 'D:'. **Figure 2** displays a minimal *1-Wire TAG* describing a 1-Wire cluster with only one device.

Figure 2. Sample *Descriptor* TAG

```
RAW TAG: (size 35)
C2 CF 01 07 04 22 6B 8B 10 02 17 01 00 80 2D 80
2B 34 01 08 10 8B 6B 22 00 00 00 FF 02 16 01 01
FF FF FF

{G: ParseData}[
   {D: ClusterNum(4)} 22 6B 8B 10
   {D: ClusterRev(1)} 0
   {G: OWCluster}[
      {G: OWSensor}[
         {D: OWNetAddress(8)} FF000000226B8B10
         {D: AccessMethod(1)} 1
      ]
   ]
]
```

The *1-Wire TAG* in **Figure 2** is display in raw byte form and also translated into a human-readable form. The entire TAG is in OBM format and is encapsulated in a group called 'ParseData'. The 'ParseData' group instructs the reader of this file that it is indeed OBM parse-able information. The 'ParseData' group is required on all OBM type packets including *Descriptor* TAGs. The content of the ‘ParseData’ group is the description of sensors. The 1-Wire sensors are contained in the ‘OWCluster’ group. The required data objects ‘ClusterNum’ and ‘ClusterRev’ are before the beginning of the ‘OWCluster’ group since it is a PRE and POST execute type group. Note that the groups have brackets and indentation to indicate the beginning and end of the group. Also note that the data objects have a length in bytes displayed in parentheses. The only sensor in the cluster is a device uniquely identified by the ‘OWNetAddress’ data object. The method to read it is specified in the ‘AccessMethod’ data object. See a complete description of AccessMethod’s later in this document.

Table 4. Summary of Access Methods for 1-Wire Sensors, 1-Wire Actuators, and 1-Wire Branches.

Access Method	Description	Number
AM_ORDER_ROM_SWITCH_LIST	Order list of switches where closure is detected by the presense of a specified 1-Wire device	0
AM_TEMPERATURE_1820	Temperature read in DS1820/DS1920 format	1
AM_TEMPERATURE_18B20	Temperature read in DS18B20 format	2
AM_COUNT_VELOCITY_2423	Velocity calculation using 1-Wire counter DS2423	3
AM_COUNT_TOTAL_2423	Count using 1-Wire counter DS2423	4
AM_SWITCH_2406	Low side switch using DS2406 (or DS2407)	5
AM_LEVEL_2406	Level sensor (HIGH/LOW) using DS2406	6
AM_ACTIVITY_2406	Activity sensor using DS2406	7
AM_SWITCH_2409	High side switch using DS2409	8
AM_LEVEL_2409	Level sensor (HIGH/LOW) using DS2409	9
AM_ACTIVITY_2409	Activity sensor using DS2409	10
AM_VOLTAGE_2450	Voltage reading using DS2450	11

Figure 3. Sample *Descriptor* TAG for 1-Wire Weather Station

```
RAW TAG: (size 300)
C2 CF 03 05 16 31 2D 57 69 72 65 20 57 65 61 74
68 65 72 20 53 74 61 74 69 6F 6E 01 07 04 06 BE
69 12 02 17 01 00 03 07 14 44 61 6C 6C 61 73 20
53 65 6D 69 63 6F 6E 64 75 63 74 6F 72 01 08 04
00 00 00 00 02 19 01 14 02 18 04 53 AD CA 37 80
2D 03 05 10 49 73 6F 6C 61 74 69 6F 6E 20 42 72
61 6E 63 68 34 01 08 12 69 BE 06 00 00 00 30 02
14 01 02 02 15 01 02 02 16 01 05 80 2A 80 2B 03
05 0E 57 69 6E 64 20 44 69 72 65 63 74 69 6F 6E
02 16 01 00 01 06 40 01 D4 57 00 02 00 00 AE 01
D0 57 00 02 00 00 72 01 01 79 73 02 00 00 FB 01
34 AB 73 02 00 00 20 01 3D 82 73 02 00 00 BD 01
26 82 73 02 00 00 1E 01 6F B1 73 02 00 00 37 01
CC 57 00 02 00 00 54 FF FF 80 2B 03 05 09 45 6E
63 6C 6F 73 75 72 65 34 01 08 10 7C 8E 15 00 00
00 CE 02 16 01 01 FF 80 2B 03 05 0A 57 69 6E 64
20 53 70 65 65 64 34 01 08 1D CF 33 00 00 00 00
B2 03 06 03 4D 50 48 02 12 02 E9 2F 02 13 02 10
27 01 06 01 0F 02 16 01 03 FF FF FF

{G: ParseData}[
   {D: Description(22)} 1-Wire Weather Station
   {D: ClusterNum(4)} 06 BE 69 12
   {D: ClusterRev(1)} 0
   {D: Manufacturer(20)} Dallas Semiconductor
   {D: ManufacturerCode(4)} 00 00 00 00
   {D: Enum(1)} 20
   {D: SecondsSince1970(4)} 936029523
   {G: OWCluster}[
      {D: Description(16)} Isolation Branch
      {D: OWNetAddress(8)} 3000000006BE6912
      {D: ChannelMask(1)} 2
      {D: ChannelState(1)} 2
      {D: AccessMethod(1)} 5
      {G: OWBranch}[
         {G: OWSensor}[
            {D: Description(14)} Wind Direction
            {D: AccessMethod(1)} 0
            {D: InitState(64)}
                01 D4 57 00 02 00 00 AE
                01 D0 57 00 02 00 00 72
                01 01 79 73 02 00 00 FB
                01 34 AB 73 02 00 00 20
                01 3D 82 73 02 00 00 BD
                01 26 82 73 02 00 00 1E
                01 6F B1 73 02 00 00 37
                01 CC 57 00 02 00 00 54
         ]
      ]
      {G: OWSensor}[
         {D: Description(9)} Enclosure
         {D: OWNetAddress(8)} CE000000158E7C10
         {D: AccessMethod(1)} 1
      ]
      {G: OWSensor}[
         {D: Description(10)} Wind Speed
         {D: OWNetAddress(8)} B20000000033CF1D
```

```
          {D: UnitName(3)} MPH
          {D: ScaleFactorM(2)} 12265
          {D: ScaleFactorD(2)} 10000
          {D: InitState(1)} 0F
          {D: AccessMethod(1)} 3
       ]
    ]
 ]
```

Group Objects

ParseData

The ‘ParseData’ group indicates that the enclosed data is OBM parse-able. This is required on all OBM packets and prevents inadvertent parsing of non-OBM data.

Execution Mode:

POST – no required objects

Required Data objects:

none

Optional Data objects:

all

Example:

```
{G: ParseData}[

       (OBM format data here)
]
```

OWCluster

The 'OWCluster' provides a logical grouping of separate 1-Wire sensors. The example of this is the 1-Wire Weather Station. The standard weather station contains three different sensors employing eleven 1-Wire devices.

Execution Mode:

PRE / POST – The OWCluster action is performed at the beginning and end of the group during parsing. The data objects must be before the beginning of the group.

Required Data objects:

ClusterNum – Provide a unique identifying number. It is recommended that this number be generated by selecting one of the 1-Wire Net devices in the cluster and extracting the unique portion. The ClusterNum is used by the reading application to uniquely identify the cluster.

ClusterRev – Provide a unique revision number. Each time the *1-Wire TAG* cluster is updated, this revision number should be increased. This number is used by the reading application to select the correct version if more then one cluster has the same ClusterNum.

Optional Data objects:

Description – Text description of the 1-Wire cluster.

Manufacturer – Text name of manufacturer.

ManufacturerCode – Binary array representing manufacturer information. The format for this has yet to be defined.

Enum – Integer representing an enumeration of the 1-Wire Cluster's of this type. This can be thought of a manufacturer's serialization number. No two cluster's of the same type should have the same enumeration number.

SecondsSince1970 – Time / Date stamp of the creation of this cluster. This is represented as an integer count of seconds from January 1, 1970.

NetRegistration – This provides the World-Wide-Web address where the result of reading this cluster can be posted. This site can then provide advanced data services.

Example:

```
{D: Description(22)} 1-Wire Weather Station
{D: ClusterNum(4)} 06 BE 69 12
{D: ClusterRev(1)} 0
{D: Manufacturer(20)} Dallas Semiconductor
{D: ManufacturerCode(4)} 00 00 00 00
{D: Enum(1)} 20
{D: SecondsSince1970(4)} 936029523
{G: OWCluster}[

       (sensor information here)

]
```

OWBranch

The ‘OWBranch’ provides a physical grouping of 1-Wire devices by creating a ‘branch’. A branch is separate from the main 1-Wire Network and must be switched on to access. This type of arrangement is used to provide physical location information and isolation of 1-Wire devices.

Execution Mode:

PRE / POST – The OWBranch action is performed at the beginning and end of the group during parsing. The data objects must be before the beginning of the group. When first called that action is to open the switch with the available information. If the *1-Wire TAG* is being used as a script, this information must be saved is some way (branch stack) so that the branch can be reopened when the end of the group is found.

Required Data objects:

OWNetAddress – The ‘OWNetAddress’ provides the 1-Wire Network node address for the 1-Wire device controlling the branch. Each node address is guaranteed to be unique.
AccessMethod – The ‘AccessMethod’ provides the method to read or access the 1-Wire device. The ‘AccessMethod’ must be known by the application in order to read the sensor.
ChannelMask – The ‘ChannelMask’ provides a bit-mask indicating which channels on the 1-Wire switch need to be set. A ‘1’ in the mask indicates the branch value needs to be set and a ‘0’ if it does not.
ChannelState – The ‘ChannelState’ provides a bit-mask indicating the desired state of the channels to open the ‘branch’. Note that bit values outside of the ‘ChannelMask’ will be ignored. A ‘1’ in the ‘ChannelState’ indicates conducting or closed and a ‘0’ indicates open.

Optional Data objects:

Description – Text description of the 1-Wire branch.

Example:

```
{D: Description(16)} Isolation Branch
{D: OWNetAddress(8)} 3000000006BE6912
{D: ChannelMask(1)} 2
{D: ChannelState(1)} 2
{D: AccessMethod(1)} 5
{G: OWBranch}[

       (information about devices on branch)

]
```

OWSensor

The ‘OWSensor’ is a group describing a device that can be read to provide a numerical value. The key to reading the sensor described in the ‘OWSensor’ is the ‘AccessMethod’ data object. The value of this object will instruct the application on what method to use to retrieve this ‘numerical value’.

Execution Mode:

POST – The OWSensor action is performed at the end of the group during parsing. The data objects must be ‘inside’ the group. Depending on the value of the ‘AccessMethod’ data object there may be other data objects required.

Required Data objects:

OWNetAddress – The ‘OWNetAddress’ provides the 1-Wire Network node address for the 1-Wire device controlling the sensor. Each node address is guaranteed to be unique.
AccessMethod – The ‘AccessMethod’ provides the method to read or access the 1-Wire device. The ‘AccessMethod’ must be known by the application in order to read the sensor.
ChannelMask – The ‘ChannelMask’ provides a bit-mask indicating which channels on the 1-Wire switch need to be set. A ‘1’ in the mask indicates which channel needs to be set. This data object is not required for all access methods.
ChannelState – The ‘ChannelState’ provides a bit-mask indicating the desired state of the channels. Note that bit values outside of the ‘ChannelMask’ will be ignored. A ‘1’ in the ‘ChannelState’ indicates conducting or closed and a ‘0’ indicates open. This data object is not required for all access methods.
InitState – The ‘InitState’ provides the initialization state for the sensor. The format of this depends on the ‘AccessMethod’.

Optional Data objects:

Description – Text description of the 1-Wire sensor.
ScaleFactorM – Scale multiplication factor. This data object is optional in some ‘AccessMethods’.
ScaleFactorD – Scale division factor. This data object is optional in some ‘AccessMethods’.
UnitName – Provide a unit name for the results from some ‘AccessMethods’.

Example:

```
{G: OWSensor}[
   {D: Description(10)} Wind Speed
   {D: OWNetAddress(8)} B20000000033CF1D
   {D: UnitName(3)} MPH
   {D: ScaleFactorM(2)} 12265
   {D: ScaleFactorD(2)} 10000
   {D: InitState(1)} 0F
   {D: AccessMethod(1)} 3
]
```

OWActuator

The 'OWActuator' is a group describing a device that can be set to affect the state of device. The key to settings the actuator described in the 'OWActuator' is the 'AccessMethod' data object. The value of this object will instruct the application on what method to use.

Execution Mode:

POST – The OWActuator action is performed at the end of the group during parsing. The data objects must be 'inside' the group. Depending on the value of the 'AccessMethod' data object there may be other data objects required.

Required Data objects:

OWNetAddress – The 'OWNetAddress' provides the 1-Wire Network node address for the 1-Wire device controlling the actuator. Each node address is guaranteed to be unique.

AccessMethod – The 'AccessMethod' provides the method to set the 1-Wire device. The 'AccessMethod' must be known by the application in order to read the sensor.

ChannelMask – The 'ChannelMask' provides a bit-mask indicating which channels on the 1-Wire switch need to be set. A '1' in the mask indicates the branch values can be changed. This data object is not required for all access methods.

ChannelState – The 'ChannelState' provides a bit-mask indicating the state to turn 'on' the actuator. Note that bit values outside of the 'ChannelMask' will be ignored. A '1' in the 'ChannelState' indicates conducting or closed and a '0' indicates open. This data object is not required for all access methods.

InitState – The 'InitState' provides the initialization state for the actuator. The format of this depends on the 'AccessMethod'.

Optional Data objects:

Description – Text description of the 1-Wire sensor.

Example:

```
{G: OWActuator}[
   {D: Description(11)} Garage door
   {D: OWNetAddress(8)} 3000000006BE6912
   {D: ChannelMask(1)} 2
   {D: ChannelState(1)} 2
   {D: AccessMethod(1)} 5
]
```

Data Objects

Description

The ‘Description’ data object is a simple text string to describe the associated group.

Format:

ASCII printable text

Max size in bytes:

255

Example:

```
{D: Description(22)} 1-Wire Weather Station
```

OWNetAddress

The ‘OWNetAddress’ data object is binary array containing a valid 1-Wire network address. This can also go by the following names (ROM Number, ROM ID, Serial Number, Registration Number).

Format:

Binary array containing a valid 1-Wire Network address.

Max size in bytes:

8 (must be 8 bytes)

Example:

```
D: OWNetAddress(8)} FF000000226B8B10
```

Manufacturer

The ‘Manufacturer’ data object is a simple text string to describe the manufacturer usually associated with a 1-Wire cluster group.

Format:

ASCII printable text

Max size in bytes:

255

Example:

```
{D: Manufacturer(20)} Dallas Semiconductor
```

ManufacturerCode

The ‘ManufacturerCode’ data object is a binary array containing bit fields to indicate Manufacturer. The format for the bit fields has not been set.

Format:

Binary array

Max size in bytes:

8

Example:

```
{D: ManufacturerCode(4)} 00 00 00 00
```

Enum

The ‘Enum’ data object is an integer indicating a simple enumeration. This value can be used to provide a serialization number for manufactured clusters.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: Enum(1)} 20
```

SecondsSince1970

The ‘SecondsSince1970’ data object is an integer indicating the number of seconds since 12:00a.m., January 1, 1970. This can be converted to a Time/Date value. For example, a value of 936106086 can be converted to a Time/Date of August 31, 1999 at 1:28:06p.m.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: SecondsSince1970(4)} 936106086
```

AccessMethod

The ‘AccessMethod’ data object is an integer indicating the method to access a particular branch, sensor, or actuator. The different ‘AccessMethods’ are described in detail later in this document.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: AccessMethod(1)} 1
```

ChannelMask

The ‘ChannelMask’ data object is an integer that indicates which channels to access for a particular branch, sensor, or actuator. This integer is a bit-mask where a ‘1’ indicates a channel to be considered.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: ChannelMask(1)} 2
```

ChannelState

The ‘ChannelState’ data object is an integer that indicates the desired state of a device with multiple channels. Only the channels indicated in the ‘ChannelMask’ will be set with the state from the ‘ChannelState’. A ‘1’ in the ‘ChannelState’ indicates an ON or conducting state.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: ChannelState(1)} 2
```

ScaleFactorM

The ‘ScaleFactorM’ data object is an integer that indicates what to multiply a resulting value by to get a correctly scaled result.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: ScaleFactorM(2)} 12265
```

ScaleFactorD

The ‘ScaleFactorD’ data object is an integer that indicates what to divide a resulting value by to get a correctly scaled result.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: ScaleFactorD(2)} 10000
```

UnitName

The ‘UnitName’ data object is a simple text string to describe unit being expressed in the result of a reading, usually a sensor.

Format:

ASCII printable text

Max size in bytes:

255

Example:

```
{D: UnitName(3)} MPH
```

ClusterNum

The ‘ClusterNum’ data object is binary array containing a unique identifier. This data object is used to identify the 1-Wire Cluster. It can easily be made unique by using a 1-Wire network address from the cluster itself. For brevity, the leading zero’s and CRC can be left off. In most cases this is result in a 4 to 5 byte array.

Format:

Binary array containing a unique identifier

Max size in bytes:

8

Example:

```
{D: ClusterNum(4)} 22 6B 8B 10
```

ClusterRev

The ‘Cluster’ data object is an integer that indicates the revision number of a 1-Wire cluster. This in will prevent any conflicts if there is a new version of the *1-Wire TAG*.

Format:

Integer, least significant byte first

Max size in bytes:

4

Example:

```
{D: ClusterRev(1)} 0
```

InitState

The ‘InitState’ data object is binary array containing initialization data. The contents of this data object depend on the current group and access method.

Format:

Binary array

Max size in bytes:

255

Example:

```
{D: InitState(1)} 0F
```

NetRegistration

The ‘NetRegistration’ data object is a simple text string representing the World-Wide-Web address for this *1-Wire TAG* to go for remote data services.

Format:

ASCII printable text

Max size in bytes:

255

Example:

```
{D: NetRegistration(14)} www.1-Wire.net
```

Access Methods

The access method designates how to access a sensor, branch, or actuator. The access method is simply a number that is known by all to represent a specific technique. New access methods can be added as new sensor types are created. If an older application does know this access method, it will simply skip that sensor or prompt for more information.

AM_ORDER_ROM_SWITCH_LIST

The 'AM_ORDER_ROM_SWITCH_LIST' access method is simply an ordered list of contact sensors. Contact is made when a 1-Wire device is in contact with the 1-Wire Network. This access method applies to the 'OWSensor' group.

Required Data objects:

InitState – Ordered list of 1-Wire Network addresses. Each 1-Wire Network Address is an 8 byte array. For example if 'InitStat' has a length of 16 bytes and a data payload of '01 D4 57 00 02 00 00 AE 01 D0 57 00 02 00 00 72' then there are two ROM contact switches in this ordered list.

Operation:

1. Start at the beginning of the list of 1-Wire Network Addresses.
2. Check to see if the 1-Wire device is in contact to the 1-Wire Network by doing a targeted search.
3. Report if device is in contact and it's order number.
4. Go to the next 1-Wire Network Address
5. Check if done with all devices
6. Go to 2.

Result:

Report which of the 1-Wire contacts are 'on' and their order number. For example, if the third and fourth 1-Wire devices were in contact, then the result would be (3,4).

AM_TEMPERATURE_1820

The 'AM_TEMPERATURE_1820' access method reads the current temperature from a DS1820/DS1920. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS1820 to be read.

Operation:

1. Verify the DS1820 is present on the 1-Wire Network
2. Recall the EEPROM calibration
3. Start a temperature conversion and apply power delivery
4. Delay 500ms
5. Read the temperature

Result:

Report the result temperature reading.

AM_TEMPERATURE_18B20

The 'AM_TEMPERATURE_18B20' access method reads the current temperature from a DS18B20. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS1820 to be read.
InitState – Single byte containing the number of bits to be used it the reading the temperature. Valid values are 9, 10, 11, and 12.

Operation:

1. Verify the DS1820 is present on the 1-Wire Network
2. Set the number of bit to used in the conversion
3. Recall the EEPROM calibration
4. Start a temperature conversion and apply power delivery
5. Delay 800ms
6. Read the temperature

Result:

Report the result temperature reading.

AM_COUNT_VELOCITY_2423

The 'AM_COUNT_VELOCITY_2423' access method calculates a velocity measurement by reading a DS2423 counter twice with a delay in between. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2423 to be read.
InitState – Single byte containing the page number of the counter to be used for velocity measurement. Valid values are 12, 13, 14, and 15.

Optional Data objects:

ScaleFactorD – The Count/Second value from the velocity reading is divided by this data object value. If this data object is not present then a value of '1' is used.
ScaleFactorM – The Count/Second value from the velocity reading is multiplied by this data object value. If this data object is not present then a value of '1' is used.
UnitName – String that is presented after the resulting scaled velocity value. This string can be a unit name such as 'MPH'.

Operation:

1. Verify the DS2423 is present on the 1-Wire Network
2. Read the counter associated with the page given in 'InitState'
3. Delay 500ms
4. Read the counter again
5. Calculate a count per second value
6. Multiply result by 'ScaleFactorM'
7. Divide result by 'ScaleFactorD'
8. Present the result with the 'UnitName' if present

Result:

Report the result velocity reading with optional 'UnitName'.

AM_COUNT_TOTAL_2423

The 'AM_COUNT_TOTAL_2423' access method reports a total count measurement by reading a DS2423 counter. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2423 to be read.
InitState – Single byte containing the page number of the counter to be used for velocity measurement. Valid values are 12, 13, 14, and 15.

Optional Data objects:

UnitName – String that is presented after the resulting scaled velocity value. This string can be a unit name such as ‘MPH’.

Operation:

1. Verify the DS2423 is present on the 1-Wire Network
2. Read the counter associated with the page given in ‘InitState’
3. Present the result with the ‘UnitName’ if present

Result:

Report the result count reading with optional ‘UnitName’.

AM_SWITCH_2406

The ‘AM_SWITCH_2406’ access method sets the output latches of a DS2406 to a specified state. This access method applies to the ‘ ‘OWBranch’ and ‘OWActuator’ groups.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2406.
ChannelMask – Bit mask of channels to operate. Since the DS2406 only has 2 channels, the only valid values are (in binary) 01 (Channel A only), 10 (Channel B only), 11 (Channel A and B).
ChannelState – Bit mask of desired state of channels for ‘OWBranch’ or default state for ‘OWActuator’. Note for the state to be set, the corresponding bit must be set in the ‘ChannelMask’. The states are ‘0’ for non-conducting and ‘1’ for conducting.

Operation:

(with OWBranch, PRE execution, or OWActuator on first execution [set default state])

1. Verify the DS2406 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state specified in ‘ChannelState’ using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

(with OWBranch, POST execution)

1. Verify the DS2406 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state to non-conducting using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

(with OWActuator, after user has selected desired state on available channel(s))

1. Verify the DS2406 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state to the what the user selected using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

Result:

Report the result of attempting to set the desired switch state.

AM_LEVEL_2406

The 'AM_LEVEL_2406' access method reads the level values from the specified channels of a DS2406. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2406 to be read.
ChannelMask – Bit mask of channels to read the level. Since the DS2406 only has 2 channels, the only valid values are (in binary) 01 (Channel A only), 10 (Channel B only), 11 (Channel A and B).

Operation:

1. Verify the DS2406 is present on the 1-Wire Network
2. Read the current level state.
3. Return the level state of the channels in 'ChannelMask'

Result:

Report the level state, HIGH or LOW for each specified channel.

AM_ACTIVITY_2406

The 'AM_ACTIVITY_2406' access method reads the activity state from the specified channels of a DS2406. This access method applies to the 'OWSensor' group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2406 to be read.
ChannelMask – Bit mask of channels to read the activity. Since the DS2406 only has 2 channels, the only valid values are (in binary) 01 (Channel A only), 10 (Channel B only), 11 (Channel A and B).

Operation:

1. Verify the DS2406 is present on the 1-Wire Network
2. Read the current activity state.
3. Return the activity state of the channels in 'ChannelMask'

Result:

Report the activity state, YES or NO for each specified channel.

AM_SWITCH_2409

The ‘AM_SWITCH_2409’ access method sets the output latches of a DS2409 to a specified state. This access method applies to the ‘ ‘OWBranch’ and ‘OWActuator’ groups. The main and auxiliary channels of the DS2409 cannot both be turned on (conducting) at the same time.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2409.

ChannelMask – Bit mask of channels to operate. Since the DS2409 only has 2 channels, the only valid values are (in binary) 01 (Main Channel only), 10 (Auxiliary Channel only), 11 (Main and Auxiliary Channels).

ChannelState – Bit mask of desired state of channels for ‘OWBranch’ or default state for ‘OWActuator’. Note for the state to be set, the corresponding bit must be set in the ‘ChannelMask’. The states are ‘0’ for non-conducting and ‘1’ for conducting.

Operation:

(with OWBranch, PRE execution, or OWActuator on first execution [set default state])

1. Verify the DS2409 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state specified in ‘ChannelState’ using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

(with OWBranch, POST execution)

1. Verify the DS2409 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state to non-conducting using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

(with OWActuator, after user has selected desired state on available channel(s))

1. Verify the DS2409 is present on the 1-Wire Network
2. Read the current switch state so that when writing the state the non-masked channels remain the same.
3. Set the switch state to the what the user selected using the ‘ChannelMask’ and the current state from 2.
4. Return the result of setting

Result:

Report the result of attempting to set the desired switch state.

AM_LEVEL_2409

The ‘AM_LEVEL_2409’ access method reads the level values from the specified channels of a DS2409. This access method applies to the ‘OWSensor’ group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2409 to be read.
ChannelMask – Bit mask of channels to read the level. Since the DS2409 only has 2 channels, the only valid values are (in binary) 01 (Main Channel only), 10 (Auxiliary Channel only), 11 (Main and Auxiliary Channels).

Operation:

1. Verify the DS2409 is present on the 1-Wire Network
2. Read the current level state.
3. Return the level state of the channels in ‘ChannelMask’

Result:

Report the level state, HIGH or LOW for each specified channel.

AM_ACTIVITY_2409

The ‘AM_ACTIVITY_2409’ access method reads the activity state from the specified channels of a DS2409. This access method applies to the ‘OWSensor’ group.

Required Data objects:

OWNetAddress –1-Wire Network Address of the DS2409 to be read.
ChannelMask – Bit mask of channels to read the activity. Since the DS2409 only has 2 channels, the only valid values are (in binary) 01 (Main Channel only), 10 (Auxiliary Channel only), 11 (Main and Auxiliary Channels).

Operation:

1. Verify the DS2409 is present on the 1-Wire Network
2. Read the current activity state.
3. Return the activity state of the channels in ‘ChannelMask’

Result:

Report the activity state, YES or NO for each specified channel.