The objects required in a typical device are:

- Either a connection object or a connection manager object;
- An identity object;
- One or several network-specific link objects (depends on network);
- A Message router object (at least its function).

Further objects are added according to the functionality of the device. This enables scalability for each implementation so that simple devices, such as proximity sensors, are not burdened with unnecessary overhead. Developers typically use publicly defined objects (see above list), but can also create their own objects in the vendor-specific areas, e.g., Class ID 100 – 199. However, they are strongly encouraged to work with the Special Interest Groups (SIGs) of ODVA to create common definitions for additional objects instead of inventing private ones.

Out of the general use objects, several are described in more detail below.

2.5.1. Identity Object (Class ID: 0x01)

The identity object is described in greater detail because, being a relatively simple object, it can serve to illustrate the general principles of CIP objects. In addition, every device must have an identity object.

The vast majority of devices support only one instance of the identity object. Thus, typically there are no requirements for any class attributes that describe additional class details, e.g., how many instances exist in the device. Only instance attributes are required in most cases. These are as follows:

Mandatory Attributes:

- Vendor ID
- Device Type
- Product Code
- Revision
- Status
- Serial Number
- Product Name

Optional or Conditional Attributes:

- State
- Configuration Consistency Value
- Heartbeat Interval
- Active Language
- Supported Language List
- International Product Name
- Semaphore
- Assigned_Name
- Assigned_Description
- Geographic_Location
- Modbus Identity Info
- Protection Mode

Let us have a look at these attributes in more detail:

- The **Vendor ID** attribute identifies the vendor that markets the device. This UINT (Unsigned Integer) value (for Data Type descriptions see Section 2.9) is assigned to a specific vendor by ODVA. If a vendor intends to build products for more than one CIP Network, the same Vendor ID will generally be assigned for all networks, but they must be registered independently with ODVA prior to use.

- The **Device Type**, again a UINT value, specifies which profile has been used for this device. It must be one of the vendor IDs listed in Volume 1, Chapter 6 of the CIP Networks Library or a vendor-specific type (see Section 2.6).

- The **Product Code** is a UINT number defined by the vendor of the device. This code is used to distinguish multiple products of the same vendor ID from the same vendor. There is generally a loose association between a Catalog Number and aproduct code, but not necessarily.

- The **Revision** is split into two USINT (Unsigned Short Integer) values specifying a major revision and a minor revision. Any device change(s) that result in modifying the behavior of the device on the network must be reflected in a change to the minor revision at minimum. Any changes in the device's logical interface, e.g., additional attributes, modified/additional I/O assemblies etc., require a change to the major revision and this change must be reflected in a revised Electronic Data Sheet (EDS) (see Section 2.7). vendor ID, vendor ID,product code and major revision provide an unambiguous identification of an EDS for this device.

- The **Status** attribute provides information on the status of the device, e.g., whether it is owned (controlled by another device) or configured (to something different than the out-of-the-box default), and whether any major or minor faults have occurred.

- The **Serial Number** is used to uniquely identify individual devices in conjunction with the vendor ID, i.e., no two CIP devices from one vendor may carry the same serial number. The 32 bits of the serial number allow ample space for a subdivision into number ranges that can be used by different divisions of larger companies.

- The **Product Name** attribute allows the vendor to give a meaningful ASCII name string (up to 32 characters) to the device.

- The **State** attribute describes the state of a device in a single UINT value. It is less detailed than the Status attribute.

- The **Configuration Consistency Value** allows a distinction between a device that has been configured and one that has not, or between different configurations in a single device. This helps avoid unnecessary configuration downloads.

- The **Heartbeat Interval** enables the Device Heartbeat Message. This is an unconnected change-of-state message that has a settable background cyclic interval between one and 255 seconds. Currently, this option is only defined for use on DeviceNet.

- The **Supported Language List** and **International Product Name** attributes can be used to describe

the product in multiple languages and the **Active Language** attribute then specifies which of the supported languages is in use.

- The **Semaphore** attribute provides a semaphore for client access synchronization to the entire device.

- The **Assigned_Name, Assigned_Description** and **Geographical_Location** attributes can be used to individualize products by the user of the product.

- The **Modbus Identity Info** attribute can provide identity information in Modbus format to the extent the device supports it.

- The **Protection Mode** attribute is an indication of the present mode of protection for the device.

The services supported by the class and instance attributes are either Get_Attribute_Single (typically implemented in DeviceNet and CompoNet devices) or Get_Attributes_All (typically implemented in ControlNet and EtherNet/IP devices). The only attributes that can be set are: the Heartbeat Interval, the Active Language, the Semaphore, the Assigned_Name, Assigned_Description and Geographical_Location attributes. The only other service that typically is supported by the identity object is the Reset service. This Reset service comes with three different options that can let the device restart in three different ways.

The behavior of the identity object is described through a state transition diagram.

2.5.2. Parameter Object (Class ID: 0x0F)

This object is described in some detail since it is referred to in Section 2.7, Configuration and Electronic Data Sheets. When used, the parameter object comes in two types: a complete object and an abbreviated version (parameter object Stub). This abbreviated version is used primarily by DeviceNet and CompoNet devices that have only small amounts of memory available. The parameter object stub, in conjunction with the Electronic Data Sheet, has roughly the same functionality as the full object (see Section 2.7).

The purpose of the parameter object is to provide a general means of allowing access to many attributes of the various objects in the device without requiring a tool (such as a handheld terminal) to have any knowledge about specific objects in the device.

The class attributes of the parameter object contain information about how many instances exist in this device and a Class Descriptor, indicating, among other properties, whether a full or a stub version is supported. In addition, class attributes tell whether a configuration assembly is used and what language is used in the parameter object.

2.6. Device Profiles

It would be possible to design products using only the definitions of communication networks and objects, but this could easily result in similar products having quite different data structures and behavior. To overcome this situation and to make the application of CIP devices much easier, devices of similar functionality have been grouped into vendor IDs with associated profiles. Such a CIP profile contains the full description of the object structure and behavior. The following vendor IDs and associated profiles are defined in Volume 1 (profile numbers in parentheses):

- AC Drives (0x02)
- CIP Modbus Device (0x28)
- CIP Modbus Translator (0x29)
- CIP Motion Drive (0x25)
- CIP Motion Encoder (0x2F)
- CIP Motion I/O (0x31)
- CIP Motion Safety Drive Device (0x2D)
- Communications Adapter (0x0C)
- CompoNet Repeater (0x26)
- Contactor (0x15)
- ControlNet Physical Layer Component (0x32)
- DC Drives (0x13)
- DC Power Generator (0x1F)
- Embedded Component (0xC8)
- Encoder (0x22)
- Enhanced Mass Flow Controller (0x27)
- Fluid Flow Controller (0x24)
- General Purpose Discrete I/O (0x07)
- Generic Device, keyable (0x2B)
- Human Machine Interface (HMI) (0x18)
- Inductive Proximity Switch (0x05)
- Limit Switch (0x04)
- Managed Ethernet Switch (0x2C)
- Mass Flow Controller (0x1A)
- Mass Flow Controller, Enhanced (0x27)
- Motor Overload Device (0x03)
- Motor Starter (0x16)
- Photoelectric Sensor (0x06)
- Pneumatic Valve(s) (0x1B)
- Position Controller (0x10)
- Process Control Valve (0x1D)
- Programmable Logic Controller (0x0E)
- Residual Gas Analyzer (0x1E)
- Resolver (0x09)
- RF Power Generator (0x20)
- Safety Analog I/O Device (0x2A)
- Safety Drive Device (0x2E)
- Safety Discrete I/O Device (0x23)
- Softstart Starter (0x17)
- Turbomolecular Vacuum Pump (0x21)
- Vacuum/Pressure Gauge (0x1C)

Device developers must use a vendor ID to uniquely identify their product. Any device that does not fall into the scope of one of the specialized Device Profiles must use the Generic Device profile (0x2B) or a vendor-specific profile. What profile is used and which parts of it are implemented must be described in the user's device documentation.

Every profile consists of a set of objects – some required, some optional – and a behavior associated with that particular type of device. Most profiles also define one or more I/O data formats (assemblies) that define the makeup of the I/O data. In addition to the commonly-defined objects and I/O data assemblies, vendors can add objects and assemblies of their own if their devices provide additional functionality. Furthermore, vendors can create profiles within the vendor-specific profile range. They are then free to define whatever behavior and objects are required for their device as long as they adhere to the general rules for profiles outlined in Chapter 6 of Volume 1 of the CIP Networks Library. Whenever additional functionality is needed by multiple vendors, ODVA encourages coordinating these new features through Special Interest Groups (SIGs), which can then create new profiles or additions to existing profiles for everybody's use and for the benefit of the device users.

All open (ODVA defined) profiles carry numbers in the 0x00 through 0x63 or 0x0100 through 0x02FF ranges, while vendor-specific profiles carry numbers in the 0x64 through 0xC7 or 0x0300 through 0x02FF ranges. All other profile numbers are reserved by CIP.

2.7. Configuration and Electronic Data Sheets

CIP provides several options for configuring devices:

- A printed data sheet;
- Parameter objects and parameter object stubs;
- An Electronic Data Sheet (EDS);
- A combination of an EDS and parameter object stubs;
- A configuration assembly combined with any of the above methods.

When using configuration information collected on a printed data sheet, configuration tools can only provide prompts for service, class, instance and attribute data and relay this information to a device. While this procedure can do the job, it is the least desirable solution since it does not determine the context, content or format of the data.

Parameter objects, on the other hand, provide a full description of all configurable data for a device. Since the device itself provides all the necessary information, a configuration tool can gain access to all parameters and maintain a user-friendly interface. However, this method imposes a burden on a device with full parameter information that may be excessive for a small device with limited internal resources. Therefore, an abbreviated version of the parameter object, called a parameter object stub, may be used (see Section 2.5.2). This option still allows access to the parameter data, but it does not describe any meaning to the data. parameter object stubs in conjunction with a printed data sheet are usable, but certainly not optimal. On the other hand, an EDS supplies all of the information that a full parameter object contains, in addition to I/O connection information, so the EDS provides the full functionality and ease of use of the parameter object without imposing an excessive burden on the individual device. In addition, an EDS provides a means for tools to perform offline configuration and to download configuration data to the device at a later time.

An EDS is a simple ASCII text file that can be generated on any ASCII editor. Since the CIP Specification provides a set of rules for the overall design and syntax of an EDS, specialized EDS editing tools, such as ODVA's EZ-EDS, can simplify the creation of EDS files. The main purpose of the EDS is to give information on several aspects of the device's capabilities, the most important ones being the I/O connections it supports and what parameters for display or configuration exist within the device. It is highly recommended that an EDS describe all supported I/O connections, as this makes the application of a device in a control system much easier. When it comes to parameters, EDS files should contain the attributes of application objects so that software can provide user access for monitoring and/or configuration purposes.

Let's look at some details of the EDS. First, an EDS is structured into sections, each of which starts with a section name in square brackets [ ]. The first two sections are mandatory for all EDS files.

- [File]: Describes the contents and revision of the file;

- [Device]: Is equivalent to the identity object information and is used to match an EDS to a device;

- [Device Classification]: Describes what network the device can be connected to. This section is optional for DeviceNet, required for ControlNet, EtherNet/IP and CompoNet;