WinKit 6 Application Layer

# Basics

Application layer works with data objects that are

* Strongly typed (with some support for polymorphic types)
* Named
* Organized into a tree-like structures

Logical structure of a set of data is described in an AppInterfaceProfile object. Basically, a profile is a set of interfaces represented by AppInterface objects. An interface defines a tree-like structure of AppInterfaceObjects which represent individual data items. Each AppInterfaceObject defines the type of the object – AppType. The AppType is a base class for a family of derived classes describing particular details of each type category.

Each object in a tree hierarchy has a (short) *name*, identifying it in the scope of its parent object, and also each object has a *full name* identifying it in the tree structure as a whole. The full name has the form

<name1>.<name2>…<nameN>.

The terminal leafs of the object tree represent *simple* objects which usually have a simple value such as string, byte, or DateTime. Intermediate leafs in the object tree represent *composite* objects – they are thought of as consisting of all child objects located below in the tree. Composite objects are *groups* and *arrays*. A group is a sequence of child objects identified by an arbitrary name and having arbitrary type. An array is a sequence of child object having the same type and identified by a numeric index.

Actual data values of application objects are stored in AppObject and CompositeValue objects. AppObject directly corresponds to some AppInterfaceObject (actually, with the same full name). CompositeValue, on the other hand, represents a value of some application type not directly linked to any particular object defined in the profile. Both AppObject and CompositeValue store a value of a particular application type.

The data value contained in AppObject is accessible via the Value property.

AppObject usually is a part of a hierarchy of objects – it may have a parent and may have children. The class AppData represents the entire collection of AppObjects corresponding to a particular AppInterfaceProfile. Note that there may exist several instances of AppData for a single instance of AppInterfaceProfile. Smaller collections of AppObjects, not covering the entire profile, are provided by instances of the AppDataSubset class.

The values of data stored in AppObject and CompositeValue instances exist in the computer memory and are accessible immediately. Their values may be retrieved from external (with respect to PC memory) storage or, conversely, stored to the external storage, using one of the *application data providers*.

There are several types of application data providers. These are classes implementing the IAppDataProvider interface. The interface contains methods that allow to

* Read a data from external storage
* Write a data to external storage
* (Optionally) subscribe to notification on changes of data in external storage.

Read and write operations follow the IAsyncRequest pattern. They may be synchronous or asynchronous, using events or callbacks to report completion. They may be aborted at any time. There may be multiple requests executing at the same time.

Currently, there are three major groups of application providers:

* Based on a mapping of application objects and types to virtual memory. These providers require additional data structure - AppInterfaceProfileImplementation – which describes the mapping. Example: AppVirtMemDataProvider.
* Based on storing the data in an xml file of a special format. Example: XHDFileDataProvider.
* Based on a mapping of application objects to DNP protocol objects of some DNP device, described by XML DNP Profile.

In the future, more provider types may be added: e.g., a provider to store/retrieve data to/from relational database.

Also, there are some provider classes that extend a provider functionally. They have technical value, simplifying common programming tasks. For example, a PollingDataProvider adds polling capability to an existing provider, thus enabling notifications of changes when the existing provider does not have this capability.

# Libraries

The application layer functionality is implemented as a class library **SandC.WinKit6.Application.dll**. The classes are contained in the **SandC.WinKit6.Application** namespace. The class library has an accompanying documentation in the **AppLevel.ClassLibrary.chm** file.

# Tree Structures

Instances of several classes such as AppObject, AppInterfaceObject, CompositeValue, etc. form a tree-like structure. These classes derive from TreeNode class which provides the following properties and methods to help navigate the tree structure:

|  |  |
| --- | --- |
| Property/Method | Description |
| Name | The (short) name of the object in its parent’s scope. |
| FullName | The full name of the object starting from the root of the hierarchy. |
| Parent | Parent object, if any (null otherwise). |
| Children | A collection of child objects. May be empty collection. |
| GetSubObject | Given a long (dot-separated) name, returns corresponding child object. |
| this[string] indexer | Gets immediate child by its name |
| this[int] indexer | Gets immediate child by its index. Applicable only to objects of array type. |
| Next | Gets next sibling child of the same parent. |

# Short and Full Names

A short name of an object may contain only letters, digits, and underscore, and must begin with a letter. The short name of an object identifies the object in its parent.

A full name has the form

<name1>.<name2>…<nameN>

A full name is used to identify an object in the tree relative to some another object, which may not be immediate parent. The GetSubObject method takes the full name and looks for its immediate child with name name1, then, in turn, takes its immediate child with name name2, and so on.

Child elements of objects having the array application type have names which are decimal representations of corresponding indices, e.g. “0”, “1”, …, “10”, “11”,… . For this case a special variant of syntax may be used: e.g., instead of “SomeArray.7” the string “SomeArray[7]” may be used to designate the eighth element in the (array) SomeArray.

# Profile and Implementation revisions

The classes AppInterfaceProfile and AppInterfaceProfileImplementation have two properties – Name and Revision. The AppInterface class has the Uri and Revision properties. These properties have special meaning and are used to control compatibility between profile descriptions, data files and physical controller data.

When an application data is written to a high-level data file, uri and revision of interfaces is written to the file. When the content of a high-level data file is loaded back, the uri and revision in the file is compared to that in the profile which is used to load data. If they do not match, the operation is aborted.

When an application data provider based on virtual memory provider is created, the name and revision of the AppInterfaceProfileImplementation is compared to the ident and revision attributes provided by the virtual memory provider (or directly read from virtual memory as specified by the CCP Addendum). If they does not match, the application data provider is not created (throws an exception).

# Compiled (serialized) profiles and implementations

AppInterfaceProfile and AppInterfaceProfileImplementation originate usually from definitions given in specially formatted xml files described in the *Application Layer File Formats* seciton. The definitions are parsed by

public AppInterfaceProfile AppProfileParser

.Main(string xmlFile, IInterfaceResolver resolver, XmlReader xmlReader = null);

public AppInterfaceProfileImplementation ProfileImplementationParser

.Load(string profileImplFName);

methods and their overloads.

However, the user usually does not need to call these methods directly. The AppInterfaceProfile and AppInterfaceProfileImplementation objects created by parsers may be serialized to a file(s) and subsequently de-serialized from them.

Compiled application interface profile files have .prfdef extension and complied interface profile implementation files have .prfimt extension. They are compressed and not human-readable.

The class ProfileDefinitionsBinaryStreamer provides methods for (de)serializing of interface profile:

void WriteProfileDefinitions(AppInterfaceProfile profile, Stream stream, bool closeStreamAfterWriting = false);

AppInterfaceProfile ReadProfileDefinitions(Stream stream, bool closeStreamAfterReading = false);

The class ProfileImplementationBinaryStreamer provides methods for (de)serializing of implementation:

void WriteProfileImplementation(AppInterfaceProfileImplementation implementation, Stream stream, bool closeStreamAfterWriting);

AppInterfaceProfileImplementation ReadProfileImplementation(Stream stream, bool closeStreamAfterReading);

There is a bsu.exe utility that compiles and serializes interface profiles and implementations. Thus, usually there is no need to call WriteXXX methods in user code. Instead, the programs like ILink load already compiled and serialized profiles and implementations using the ReadXXX methods.

# Read/Write Operations

An IAppDataProvider class implements the IAsyncRequest pattern. To perform a read or write operation, an AppDataConnection instance should be created by the method

AppDataConnection GetAppDataConnection();

The AppDataConnection class provides the following (synchronous) methods and their asynchronous variants.

Read

void Read(IEnumerable<AppObject> readData, out OPCompStatus st, IProgress<long> progress = null);

readData is a list of object to read. Upon completion of the operation, the st variable gets the result status of the operation (which may be OK, Aborted, or CommunicationError) and, if the status is OK, the AppObjects in readData are filled with data values retrieved from corresponding storage (depending on provider type). Each of the AppObjects in the list (and their children) will have the Status property set accordingly.

The progress argument, if supplied, may report progress notifications during a lengthy operation.

Write

For convenience, there are several overloaded variants of the Write method. The basic one is

void Write(IEnumerable<AppInterfaceObject> writeData, IEnumerable<CompositeValue> newValues, out OPCompStatus status, IProgress<long> progress = null);

The writeData argument provides a list of AppInterfaceObjects identifying application objects that have to be written to; newValues is a list of corresponding values to be written. Upon completion, the result of the operation is found in status argument.

The progress argument, if supplied, may report progress notifications during a lengthy operation.

Other overloaded variants of the Write method include:

1. Specifying AppObject instead of AppInterfaceObject. The AppInterfaceObject to which AppObject refers will be used.
2. Specifying Object instead of CompositeValue in newValues. This is obsolete and not recommended.
3. Specifying a single AppObject instead of a pair writeData-newValues. The current value of AppObject will be used as newValue. This is obsolete and not recommended. This method may lead to problems if the AppObject is simultaneously read by another request.

# Application Object Types

Types of application data are described by instances of classes derived from the base AppType class. The base class (and hence all its derived classes) provide the following properties and methods:

|  |  |
| --- | --- |
| Method/Property | Description |
| Category | Category of the type – see below. |
| Name | Name of the type. May be empty for auto-generated (anonymous) types. |
| FullName | Full name of type. It has the form <interfacename>.<typename> |
| DefaultValue | Default value for objects of this type. |
| DotNetType | .Net type of the values of objects of this type. |
| Description |  |
| Units |  |
| Validate | Validates whether the given object is a valid value of this type. |

The Category property of AppType is used to determine the actual derived type of AppType. Type categories are:

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Application Type | DotNetType | Description |
| Numeric | NumericType | double | Numbers |
| String | StringType | string | Strings |
| Enumeration | EnumerationType | string | Enumeration type. |
| Version | VersionType | string | Version string, e.g. “1.0” |
| TimeStamp | TimeStampType | DateTime | Timestamps |
| IPAddress | IPAddressType | IPAddress | IP addresses |
| Boolean | BooleanType | boolean | Booleans |
| GPSLatitude | GPSLatitudeType | Latitude | Class representing a latitude value |
| GPSLongitude | GPSLongitudeType | Longitude | Class representing a longitude value |
| TimerByte | TimerByteType | double | Legacy applications support. A time interval from 0 up to 12 hours (in seconds) |
| Array | ArrayType | List<object> | Array of objects having the same type. |
| Group | GroupType | Dictionary<string, object> | Analog of “struct” in C language. |
| Union | UnionType | null | Have no static value type. Actual type depends on data. |
| OctetString | OctetStringType | byte[] | Array of bytes (octets) |

# Individual Application Types Descriptions

An application type is an instance of one of the AppType-derived classes. Each instance has specific values of properties determining the application type. Thus, an application type is an *instance* of one of .NET classes below, not the .NET type itself.

In many cases, the Validate method of a type is all that is needed to work with data - e.g., test a value for correctness before a write operation. However, some applications require a more intimate knowledge of an application type. For example, GUI editor for object values uses information about particular type to adjust its interface accordingly.

NumericType

|  |  |
| --- | --- |
| Property | Description |
| SpecialValues | List of special values |
| Ranges | List of ranges |

Special value is a pair consisting of a name (string) and a value (double).

A range is a pair of boundaries – Min and Max and a Step.

A number belongs to a range if it is between the boundaries.

A number is a valid value for this type when it either belongs to one of the ranges or coincide with one of the special values.

StringType

|  |  |
| --- | --- |
| Property | Description |
| MaxLength | Maximal length of the string (in Unicode characters). |

EnumerationType

|  |  |
| --- | --- |
| Property | Description |
| Values | List of possible values (strings) for this enumeration. |

VersionType

|  |  |
| --- | --- |
| Property | Description |
| NumberCount | Count of numbers in the version strings. E.g., three in “1.0.0”. |
| NumberSize | 1, 2, or 4 to indicate byte, ushort, or uint respectively. |

TimeStampType

|  |  |
| --- | --- |
| Property | Description |
| From | The oldest timestamp supported. |
| To | The newest timestamp supported. |
| PrecisionCategory | Precision of timestamp - Seconds or Milliseconds. |

IPAddressType

Nothing special. There exist only one IPAddressType.

BooleanType

Nothing special. There exist only one BooleanType.

GPSLatitudeType

There exist only one GPSLatitudeType.   
The value of object of GPSLatitudeType has Latitude type. Latitude type is the helper class that keeps latitude value and could be converted to string using the following formats:  
“ISO” : ±DDMMSS.SSSS  
“D” : ±DD.DDDD  
“DM” : ±DDMM.MM  
“DMS”: ±DDMMSS.S  
Also, Latitude class has two constrictors:  
Latitude( float degrees ) and Latitude( iso\_string) where iso\_string is the latitude representation in ISO format: ±DDMMSS.SSSS

GPSLongitudeType

There exist only one GPSLongitudeType.  
The value of object of GPSLongitudeType has Longitude type. Longitude type is the helper class that keeps longitude value and could be converted to string using the following formats:

“ISO” : ±DDDMMSS.SSSS  
“D” : ±DDD.DDDD  
“DM” : ±DDDMM.MM  
“DMS”: ±DDDMMSS.S

Also, Longitude class has two constrictors:

Longitude( float degrees ) and Longitude( iso\_string) where iso\_string is the longitude representation in ISO format: ±DDDMMSS.SSSS

TimerByteType

Nothing special. There exist only one TimerByteType.

ArrayType

|  |  |
| --- | --- |
| Property | Description |
| Size | Length of the array |
| ElementType | AppType of the element of the array |

GroupType

|  |  |
| --- | --- |
| Property | Description |
| ListNames | A list of child elements (fields) names. |
| ItemTypes | A dictionary which maps a field name to field type. |
| ItemDescriptions | A dictionary which maps a field name to field description. |

UnionType

|  |  |
| --- | --- |
| Property | Description |
| Types | A dictionary that maps a selector value (string) to an application type for the actual type of this union. |

OctetStringType

|  |  |
| --- | --- |
| Property | Description |
| Size | The length of the octet string – the number of bytes. |

# Composite Values

The CompositeValue class derives from the TreeNode class, thus providing the usual tree navigation methods.

|  |  |
| --- | --- |
| Method/Property | Description |
| ObjectType | AppType of this CompositeValue |
| ActualType | Actual AppType for this CompositeValue. Coincide with ObjectType when ObjectType is not UnionType. When ObjectType is a UnionType, ActualType is either null, meaning that actual type is unknown, or one of AppTypes (other than UnionType). |
| Value | The value of this CompositeValue, when this CompositeValue have a simple type – that is, it is not actually composite and has no child elements. |
|  |  |

Comparison of AppObject.Value and CompositeValue

Both methods of accessing data values stored in AppObject are equivalent and provide similar capabilities.

The Value property of an AppObject contains the data value associated with this instance of AppObject. For simple objects (terminal leafs in the tree) the value usually has a simple type such as string or double. For composite objects, the value has a type List<object> for arrays and Dictionary<string,object> for groups. The CompositeValue property of an AppObject provides an instance of the CompositeValue type which essentially contains the same data, but in a form more suitable for navigation in the composite structure. Thus, the use the Value property for accessing really composite values is discouraged.

Note also that while a CompositeValue provide convenient means to navigate in its tree structure, the use of CompositeValue and plain AppObject.Value to access composite data is actually not recommended because of performance issues: these objects occupy additional memory and need processor resources to be created. It is recommended to navigate over AppObjects until objects of simple types are reached, and then access their values.

However, there is an exception: for objects of union types, if the union type includes group type variants, there is no AppObjects corresponding to the members of the inner group. The only way to access the data of the inner group is to use CompositeValue or Value of the entire union.

# Unions

An object of union type may have different actual types depending of the value of selector object. Both selector object and union type object are members of the same group.

When there is no particular value of the selector object (not read or assigned, communication error, invalid data in controller), the union type object have no actual type and no particular value – its ActualType property is null.

An AppObject having a union type have no child elements.

The union type object may be written only together with its selector object.

When a union type object is been read, the library implicitly reads the corresponding selector object to determine the actual type of the union object.

Assignment to the Value property of a selector object changes the ActualType property of the union object and destroys its value (if it previously had one).

If a definition of a union type contains composite (not simple) types (array types, group types, other union types), members of that inner composite types can only be accessed via CompositeValue property of the object. The corresponding AppObject (of the union type) has no child elements, because they might be different depending on particular value of selector. However, the CompositeValue contains at any given moment a particular value of the selector which explicitly determines the ActualType of the union object and thus all its possible child elements.

# Providers

All providers implement IAppDataProvider interface and can be used to read/write application object value. They differ in the external storage to which the values are written and additional service provided.

When a provider is created, it is passed an optional boolean isReadOnly parameter which defaults to false. If isReadOnly is true, the created provider supports read operations but attempting a write operation causes an InvalidOperationException.

AppVirtMemDataProvider

AppVirtMemDataProvider is created by a static Create method with parameters:

* AppInterfaceProfile – interface profile
* AppInterfaceProfileImplementation – implementation of the interface profile
* IVirtMemProvider – the source of virtual memory

Read and write operations use implementation to translate object values into virtual memory content and then use IVirtMemProvider to read/write the memory.

XHDFileDataProvider

XHDFileDataProvider is created by a static Create method with parameters:

* AppInterfaceProfile – interface profile
* A high-level data file name.

The content of the file is read into provider internal data structures and is used as initial state of application data values. Write operations change object values in internal data structures but does not immediately modify the file. The Flush method re-writes the file with new object values.

AppCombinedDataProvider

AppCombinedDataProvider creates new provider from existing ones. It is created by a static Create method with parameters:

* List of IAppDataProvider s
* Name for the newly created profile
* Revision for the newly created profile

The newly created provider supports an interface profile which is constructed by combining all interfaces which are contained in all the profiles of providers in the list. The method return null if the provider could not be created due to duplicate interface names in providers.

Read and write requests are performed as follows: all objects in the request are sorted into groups depending on provider from with the object’s interface originated. Then separate requests are made to original providers.

PollingDataProvider

This provider adds change notifications capability based on periodic polling for data to an existing provider. It is created by a static Create method which takes an existing provider as an argument.

WriteBlockingDataProvider

This provider is a wrapper over an existing provider. It helps detect write collisions on objects. It is created by a static Create method taking existing provider as an argument.

The WriteBlockingDataProvider maintains a list of objects that are currently being written to. When an attempt is made to write to an object that is already being written to, the write operation throws InvalidOperationException.

AppDataProvider

AppDataProvider is a stacked combination of three providers: PollingDataProvider on top of WriteBlockingDataProvider on top of AppVirtMemDataProvider. It is created by a static Create method with the same arguments as AppVirtMemDataProvider. Essentially, this is a shortcut for creating a stack or providers.

Other application data providers may be created in other object libraries different from SandC.WinKit6.Application. A notable example is AppDnpDataProvider which maps application objects to native objects of the DNP protocol (other than virtual memory).

# Subscriptions

An IAppDataProvider has a property SupportsSubscription indicating whether it supports subscriptions. If a provider does not support subscriptions inherently, subscription functionality may be added by stacking PollingDataProvider above it:

var pollingProvider = new PollingDataProvider(nonPollingProvider);

The PollingDataProvider uses the underlying data provider supplied as an argument in the constructor to perform read/write access to data, maintains a cache of already read values, and regularly polls for new values. When changes are detected, change notification on subscription is fired.

A subscription is created by the method

public ISubscription<IEnumerable<AppObject>> CreateSubscription(IEnumerable<AppObject> appObjects)

The appObjects lists the objects in which the user is interested.

There may be several subscriptions simultaneously.

To receive change notification, IObserver pattern is used (see MSDN documentation for detailed explanation). The user should create the IObserver<IEnumerable<AppObject>> and pass it to Subscribe method of ISubscription (inherited from IObservable).

Some providers generate change notification by themselves – e.g., based on receiving an unsolicited message from a device.

The PollingDataProvider actually polls for data at some time interval to detect changes.

This provider has a more specialized method to create subscriptions:

public AppSubscription CreateSubscription(IEnumerable<AppObject> appObjects, int pollingTime);

The pollingTime argument gives the time interval for polling. The subscription class AppSubscription derives from ISubscription<IEnumerable<AppObject>> but also provides some specific functionality:

|  |  |
| --- | --- |
| Method/Property | Description |
| Period | Polling interval |
| IsStarted | Polling is active |
| Start | Starts polling |
| Stop | Stops polling |
| Poll | Perform immediate, out-of-schedule, poll |

# High-Level Data Files

There are two types of high-level data files:

* Data files containing values of particular objects in application interfaces – data snapshots (.xdss) and settings files (.xspt)
* Data files containing a value of a particular application type (.xdat)

The files have simple xml format. In both types of files, top xml elements specify the interfaces particular data objects belong to. In .xdss and .xspt files inner xml elements specify a name of an object in the interface and the value of the object. In .xdat files a name of an application type is specified.

High-level data files are used to store application data. Their content may be accessed in two ways:

* Save/Load methods
* Using a special application data provider

# Save/Restore Groups

Save/Restore groups are defined in .xsptdef files. The format of the file is described in SaveRestoreGroupsDefinition\_1.1.doc document.

An application profile definition file (.xprfdef) includes a reference to a file with definitions for save/restore pertaining to this profile.

At run-time, the save/restore groups are represented by the class AppSaveRestoreGroups and is accessible via the SaveRestoreGroups property of AppInterfaceProfile.

Each save/restore group is represented by an instance of SaveRestoreGroup. Save/restore groups are organized into tree-like hierarchy. Navigation over the hierarchy is supported by TreeNode methods (and SaveRestoreGroup inherits from TreeNode).

The property SRGroupsTree of AppSaveRestoreGroups contains the list of top-level groups in the hierarchy.

Application objects belonging to a SaveRestoreGroup are given by the ChildItems property. It is assumed that a save/restore group represented by an instance of SaveRestoreGroup contains application objects listed in ChildItems of this SaveRestoreGroup and ChildItems lists of all SaveRestoreGroup down in the hierarchy.

The object data of a particular save/restore group may be stored to a file or retrieved from a file by the Save and Load helper methods. In these methods, an application data provider and a file name are given as parameters. The Load method retrieves data values from high-level data files and writes it to provider. The Save method retrieves the values of objects belonging to a save/restore groups from the provider and stores them into a file.

Special support is provided for GUI end-user programs like ILink. The SaveRestoreGroup class has a Boolean property Cheched which may be set/unset in the user program. The method SaveAllChecked is similar to the Save method but it does not have the parameter specifying the group to be saved. Instead, it saves all groups that that have Cheched property set to true.

# Special Applications of Virtual Memory Mappings

In general, programmer does not deal with type mapping (to virtual memory) at the API level. When application object values are read or written, the library uses type mapping definitions given in the xprfimp file to translate object values to/from virtual memory values, and this is transparent to the user.

However, there are special cases where the virtual memory mapping API is used explicitly. One of such cases is decoding of event logs stored in compact flash (CF) files.

A particular type mapping of interest may be obtained by a method of AppInterfaceProfileImplementation

AppTypeMapping GetTypeMappingByName(string fullTypeName, string mappingName);

The obtained AppTypeMapping may be used, e.g., to convert the content of a block of virtual memory into object value by using a method

CompositeValue RestoreCompositeValue(VirtMemBitBlock virtMemBitBlock);

or using the convenience function

CompositeValue RestoreCompositeValue(byte[] buffer);

Note that the length of the buffer in both cases should be at least StorageSizeInBytes, as specified by that AppTypeMapping property.

# Supplement: IAsyncRequest pattern

Access to major functionality of the libraries is provided via API that allows asynchronous access, simultaneous access, and the possibility to abort the outgoing request. This is achieved uniformly in all libraries via IAsyncRequest pattern (similar in various respects to IAsyncResult pattern of .NET).

Note: IAsyncRequest pattern was developed before the release of .NET 4 which provides similar functionality, allowing aborting a call, in CancellationToken and CancellationSource classes. In the future, internal workings of the libraries may be implemented through the use of these classes, and additional APIs may be added to provide the framework suggested by these classes.

A general pattern is as follows: usually, there is a class which provides some services - e.g., a device to which requests can be made. Each request executes on a *Request Control Block (RCB)* class, specific to the service. Each request is executed on its own instance of RCB. Each RCB is capable of executing only one request at a time. There may be several simultaneously running RCB to the same device. After RCB completes execution of a request, it may be re-used to execute another request.

Request control block class implements IAsyncRequest interface. The interface has the following members:

public interface IAsyncRequest

{

bool CompletedSynchronously {get;}

bool IsCompleted {get;}

WaitHandle AsyncWaitHandle {get;}

bool IsAborted {get;}

Action<IAsyncRequest> Callback {get; set;}

object UserToken {get; set;}

void Abort();

void Close();

}

|  |  |
| --- | --- |
| CompletedSynchronously | The request completed synchronously. The result is available immediately. |
| IsComleted | The request has completed (synchronously or asynchronously). |
| AsyncWaitHandle | A WaitHandle which becomes signaled when request is completed. |
| IsAborted | True, if the request was aborted. |
| IsClosed | The request control block was disposed and cannot be used anymore. |
| Callback | A delegate which is called when asynchronous execution of the request completes. |
| UserToken | Arbitrary object that may be set or get by the user at any time. |
| Abort() | Aborts the execution of request that is being executed on this RCB. |
| Close() | Disposes the resources associated with this RCB. |

Members of the IAsyncRequest interface provide properties and methods common to all RCBs. Each type of particular RCB has its own specific methods to initiate a request and to get the results of the request. These methods follow the pattern:

If synchronous form of a request is

public int Operation(TIn inParam, ref TRef refParam, out TOut outParam);

then there are methods

public void OperationAsync(TIn inParam, ref TRef refParam);

public int GetOperationResult(ref TRef refParam, out TOut outParam);

If synchronous form of a request is

public void Operation(TIn inParam, ref TRef refParam, out TOut outParam);

then there are methods

public void OperationAsync(TIn inParam, ref TRef refParam);

public bool GetOperationResult(ref TRef refParam, out TOut outParam);

The method OperationAsync initiates a request. The method GetOperationResult may be called when request is completed.

Rules:

If after OperationAsync the value of CompletedSychronously is true, IsCompleted is true and AsyncWaitHandle is in signaled state.

If IsCompleted is true or AsyncWaitHandle is signaled, request has completed the execution and GetOperationResult may be called.

If request completes synchronously, the Callback is not called.

If request was executed asynchronously and completed the execution, the Callback is called (if Callback was set).

The library never sets or gets the UserToken object.

If Abort() or Close() were executed, the RCB cannot be used for further requests.

If Abort() was called, the Callback may or may not be called, depending on the state of request execution at which the Abort() was called.

Close() executes Abort() internally, so that any running request is terminated.

Close() should be called eventually for every RCB so that resources associated with the RCB could be released.

Interface Profile Xml Files Format

An interface profile is defined in the following three types of files:

|  |  |  |
| --- | --- | --- |
| File extension | Description | Schema |
| xprfdef | Main interface profile definition file; contains references to xintdef and xsptdef files. | InterfaceProfile\_1.1.xsd |
| xintdef | Contains a definition of a single interface. | Interface\_1.1.xsd |
| xsptdef | Contains definitions of setpoint save/restore groups. | SaveRestoreGroups\_1.1.xsd |

# Interface Profile

An interface profile is defined in a file with extension xprfdef. It contains the root element <InterfaceProfile> with the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| formatRev | Yes |  | Should be 1.1 – currently supported format revision. |
| name | Yes |  | Name of the interface profile. |
| revision | Yes |  | Revision of the interface profile. |

The <InterfaceProfile> element may have the following child elements:

* <Interfaces> - a section containing references to xintdef files defining individual interfaces of this profile.
* <SaveRestoreGroups> - contains a reference to xsptdef file containing definitions of save restore groups.
* <EventRecordType> - contains the name of the event record application type.

## EventRecordType attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| file | Yes |  | The name of a file with xsptdef exptension containing definitions of save/restore groups. |

The <Interfaces> element contains a list of <Interface> elements.

## Interface attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| uri | Yes |  | Uri of the interface – see below. |
| revision | Yes |  | Revision of the interface |
| name | Yes |  | The name of an interface to be used in this profile as an alias for the uri. |

Example.

<InterfaceProfile revision="3.3" formatRev="1.1" name="IntelliRupter" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="file:///C:/Program%20Files/S&amp;C%20Electric/WinKit%206/Schemas/InterfaceProfile\_1.1.xsd">

<Interfaces>

<Interface revision="1.0" uri="CommonTypes" name="CommonTypes"/>

<Interface revision="1.0" uri="Communications\DNP" name="DNP"/>

<Interface revision="1.0" uri="Logging\Logging\_IR" name="Logging"/>

…

</Interfaces>

<SaveRestoreGroups file="IntelliRupter.xsptdef"/>

<EventRecordType>Logging.HistEventRecordCF</EventRecordType>

</InterfaceProfile>

# Interface Libraries and Uries

The application library identifies interfaces by an uri and revision. A special module (IInterfaceResolver) is used to retrieve an interface, given its uri and revision. There may be several different implementations of IInterfaceResolver – based on file system folders and files, compressed (zip) file, database, or other storages.

An interface is identified by its uri and revision. The uri is a global (up to interface resolver/library) identifier of an interface. Its format is not defined by this specification and may be an arbitrary string, depending on the implementation of interface resolver.

WinKit 6 currently supports interface resolvers based on directory structure and compressed (zip) file.

A group of interfaces is organized into folders (in file system or inside a compressed file) and is called an interface library. There may be several interface libraries. A particular interface resolver is initialized by a library – its root directory in file system or compressed file path. The uri of an interface is a relative path under the root directory of the library.

The directory-based interface resolvers use the following convention to store an interface: for uri=”name1\name2\...\nameN” and revision=”x.y.z.t” the interface file relative path is

Name1\name2\...\nameN\x.y.z.t\nameN.xintdef

Usually, nameN would be a name of the interface in the interface profile. To avoid conflicts with other interfaces, a different name may be chosen.

# Save/Restore Groups Xml Files Format

Save/restore groups are defined in an xsptdef file. The root element is <SaveRestoreGroups> which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| formatRev | Yes |  | Format of the file – should be 1.1. |
| revision | Yes |  | Currently not used. |
| name | Yes |  | Currently not used. |

The <SaveRestoreGroups> element contains a tree-like structure formed by its child <SRGroup> elements, each defining an individual save/restore group. An <SRGroup> element may have the following child elements:

* <SRGroup> - child leafs in the tree of SRGroup elements.
* <Item> - defines an application object belonging to this SRGroup.

The <SRGroup> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | Yes |  | Name of the SRGroup. This name is shown to the user on Save/Restore dialog unless overridden by TextMapping in transformation layer. |
| admin | No | false | Indicates that the group is not selected by “Select All” button on Save/Restore dialog. |

The single <Item> element attribute is

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | Yes |  | Full name of an application object in the interface profile. |

Example.

<SaveRestoreGroups xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="IntelliRupterSetpointSrgDef" revision="1.0" formatRev="1.1" xsi:noNamespaceSchemaLocation="file:///C:/Program%20Files/S&amp;C%20Electric/WinKit%206/Schemas/SaveRestoreGroups\_1.1.xsd">

<SRGroup name="General">

<SRGroup name="IntelliRupter">

<Item name="IR\_General.SPT"/>

</SRGroup>

<SRGroup name="UserCommands">

<Item name="UserCommands.SPT"/>

</SRGroup>

</SRGroup>

<SRGroup name="Protection">

<SRGroup name="GeneralProfile1">

<SRGroup name="Direction1">

<Item name="ProtectionProfiles.SPT.GeneralProfile1.Direction1"/>

</SRGroup>

<SRGroup name="Direction2">

<Item name="ProtectionProfiles.SPT.GeneralProfile1.Direction2"/>

</SRGroup>

</SRGroup>

</SRGroup>

<SRGroup name="Other">

<Item name="IntelliTeam.SPT.ExternalLoading"/>

<Item name="IntelliTeam.SPT.PeerToPeer"/>

</SRGroup>

</SaveRestoreGroups>

# Interface Xml File Format

An application interface is defined in an xindef file. It has three major sections, defining

* Type definitions borrowed from other interfaces,
* Types defined in this interface,
* Definitions of interface objects,

Which are given in corresponding child elements of the root <Interface> element:

* <ExternalTypes>
* <Types>
* <Objects>.

<Interface> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| uri | Yes |  | Uri of the interface defined in this file. |
| revision | Yes |  | Revision of the interface defined in this file. |
| formatRev | Yes |  | Should be 1.1. |

Interfaces containing no interface objects definitions (<Objects> section) are called *type libraries.* An interface may have no <ExternalTypes> section if it borrows no types from other interfaces. An interface may have not <Types> section if it does not define new types and only uses built-in types and types borrowed from other interfaces.

Example.

<Interface xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xd="http://schemas.microsoft.com/office/infopath/2003" formatRev="1.1" revision="1.0" uri="Communications\WiFi" xsi:noNamespaceSchemaLocation="file:///C:/Program%20Files/S&amp;C%20Electric/WinKit%206/Schemas/Interface\_1.1.xsd">

<ExternalTypes>

*External types definitions*

</ExternalTypes>

<Types>

*Types definitions*

</Types>

<Objects>

*Objects definitions*

</Objects>

</Interface>

## External Types

The <ExternalTypes> section borrows type definitions given in other interfaces to be used in this interface – in object definitions and type definitions.

The <ExternalTypes> elements may contain a list of <FromInterface> elements, each of which borrows type definitions from a particular interface. The foreign interface is identified by attributes of <FromInterface> element, and borrowed types – by child <ExternalType> elements.

<FromInterface> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| uri | Yes |  | Uri of the foreign interface. |
| revision | Yes |  | Revision of the foreign interface. |

The <ExternalType> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | Yes |  | Name of the type in the foreign interface. |
| alias | No | Same as name. | Alias name of the type to be used in this interface. |

When the content of an xintdef file is processed by the parser, the parser maintains the list of “known” types which is used in processing of type definitions and object definitions. This list on “known” types for the interface definitions includes:

* Names of predefined types (such as byte, int, etc.)
* Names of types defined in this interface
* Aliases for names of types borrowed from other interfaces.

All names in the list should be different.

Example.

<FromInterface uri="CommonTypes" revision="1.0">

<ExternalType name="MACAddress"/>

<ExternalType name="SimpleCommand"/>

<ExternalType name="Version4" alias="V4"/>

</FromInterface>

# Type Definitions

Type definitions are given in the <Types> section of xintdef file.

There are simple types (usually mapped to a single object of the underlying communication protocol, or mapped to single contiguous range of virtual memory) and compound types such as groups and arrays. Also, objects of variable types – which may change at run-time (unions) - are supported.

An interface object (defined in the <Objects> section of the interface) may have a *named* type. In this case the object (group, array) definitions contains the attribute type=”...”. Alternatively, group and array objects may have anonymous types. These are the types that are built by the parser automatically based on the object definition. The object definition in this case should include all necessary type information. These may be inline type definition (for group objects and simple type objects), or special attributes (size, elementType) for arrays.

Named type definitions have an attribute name=”...” giving the name to the type.

Explicit type definitions are defined in the elements of the form <xxxType>, where xxx stands for the type category.

The order in which type definitions appear in the <Types> section does not matter – for example, a group type definition may precede the type definition for a group member. However, there should be no circular references. An example of a circular reference: a group type G1 defines a group member with type G2, and the group type G2 defines a group member with type G1. Group types G1 and G2 may be in the same or in different interfaces.

## Simple Types

### Numeric Types

Numeric type is defined by the element <NumericType>. The <NumericType> element may have child elements

* <Range>
* <SpecialValue>

which define possible values for the type.

<NumericType> attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| units | No | None | Optional string representing units of measure for the numeric values of the type. |
| defaultValue | No | 0 or min | Default value for objects of this type when they are created. |

The <Range> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| min | Yes |  | Minimal value for objects of this type |
| max | Yes |  | Maximal value for objects of this type |
| step | Yes |  | Difference between two adjacent values of this type. |

min should be less than or equal than max. If min is less than max, step should not be equal to zero. Step should be non-negative number.

The <SpecialValue> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | Yes |  | String name for this SpecialValue. |
| value | Yes |  | Numeric value for this SpecialValue. |

All names in <SpecialValue>s should be different.

All values in <SpecialValue>s should be different and should not lie in the Range.

Example.

<NumericType name="SyncInterval" units="min">

<Range min="1" max="10080" step="1"/>

<SpecialValue name="Disabled" value="0"/>

</NumericType>

### String Types

String type is defined by the element <StringType> which has a single attribute:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| Length | Yes |  | Maximum length of the string |

Strings of variable length are not currently supported.

Example.

<StringType name="ApplicationID" length="16"/>

### Enumeration Types

Enumeration types are defined by the element <EnumerationType> which contain a sequence of the <Item> elements, each defining a particular possible value for the enumeration.

The <Item> element has a single attribute:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| value | Yes |  | One of the possible values for the enumeration. |

All values defined in different <Item> elements should be different.

Example.

<EnumerationType name="ActiveTimeSource">

<Item value="ProcessorClock"/>

<Item value="RealTimeClock"/>

<Item value="GPS"/>

</EnumerationType>

### Version Types

Version types are defined by the element <VersionType> which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| numberCount | Yes |  | Quantity of numbers in version string |
| numberSize | No | 1 (byte) | A range for each of the numbers constituting a version string. Is given as a number of bytes used to represent individual number. |

Example.

<VersionType name="Version2" numberCount="2" numberSize="byte">

### Timestamp Types

Timestamp types are defined by the element <TimestampType> which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| precision | Yes |  | Seconds or milliseconds |
| from | No | January 1, 1970 | Earliest timestamp value |
| to | No | 0x7FFFFFFF seconds from Jan 1, 1970 | Latest timestamp value |

The value of *from* attribute should precede the value of *to* attribute on time scale.

### IPAddress, Boolean, GPSLatitude, GPSLongitude, and TimerByte Types

There is only one built-in instance of a type of each of these categories. No xml elements exist for these types.

### OctetString Type

OctetString type is defined by the <OctetStringType> element which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| Size | Yes |  | Length of octet string |

OctetString type is essentially a SimpleArray of bytes.

Example.

<OctetStringType size="6" name="MACAddress"/>

### SimpleArray Type

SimpleArray type is defined by the <SimpleArrayType> element which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| size | Yes |  | Length of the array |
| elementType | Yes |  | Type of an element in the array. |

elementType should refer to one of the built-in simple numeric types.

Example.

<SimpleArrayType size="6" elementType="byte" name="MACAddress"/>

## Compound Types

### Group Types

Explicit group types are defined by <GroupType> element. The <GroupType> element has the following child elements, defining the members of the group:

* <Inherits> means that all members of another group type are included also into this group type in place of the <Inherits> element.
* <Group> - a group member
* <Array> - an array member
* <Object> - a simple type member
* <Union> - a union type member of the group.

The Group, Array, Object, and Union members should have the name=”...” attribute which specify the member name.

The type of a member may be specified by either the type=”...” attribute, or by inline type definition.

The type=”...” attribute should refer to the named type definition having corresponding category – GroupType for <Group>, ArrayType for <Array>, UnionType for <Union>, and a simple type for <Object>.

Inline type definitions have a different form for those elements.

* <Union> does not have inline type definition possibility.
* For <Object> element the inline simple type definition is given by the child <xxxType> element, where xxx stands for the category of one of the simple types.
* For <Array> the inline type definition is provided by two attributes: size=”...” and elementType=”...”. These are the same attributes with the same meaning as in the <ArrayType> element.
* For <Group>, inline type definition is provided as a sequence of child elements – the same as in <GroupType>, except for the <Inherits> elements. Thus, the child elements of the <GroupType> element may form a tree with several levels of child <Group> elements.

If the type=”...” attribute is present, there should be no inline type definition, and vise versa.

The <Union> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| type | Yes |  | Union type of this group member. |
| selector | Yes |  | Name of the selector object. |

The type should refer to a name of some <UnionType>. The selector should refer to the name of another member of the same group. This member should have EnumerationType which should be the same as specified as selector type for the union type.

### Array Types

The named array types are defined by the <ArrayType> element which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| size | Yes |  | Length of the array |
| elementType | Yes |  | Type of an element in the array. |

The difference with <SimpleArrayType> is that <Array> allows any type as its element type. Also, the objects of these types are represented differently at run-time.

Example.

<ArrayType size="8" name="TeamMembers" elementType="TeamMember"/>

### Union Types

Union types implement the “variable type with selector” concept. For each union type there is an associated “selector” type, which has to be an EnumerationType. Similarly, for each object of union type there is an associated “selector” object of corresponding “selector” type. A particular value of the “selector” object at run-time determines which particular type (listed in union type definition) has the union object.

The union type is defined by the <UnionType> element which has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| Selector | Yes |  | Name of some EnumerationType. |
| defaultType | No | None | Type of the union object when selector value is not listed in <Case> elements. |

The selector attribute should refer to the existing EnumerationType name.

defaultType attribute should refer to the existing type name.

The <UnionType> contains a sequence of <Case> elements which defines possible actual types for different values of the selector object. The <Case> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| Selector | Yes |  | Value of the selector object |
| Type | Yes |  | Type of the union object when the selector object has the specified value. |
| readonly | No | false | When true and selector object has the specified value, the union object is not writable. |

All values of the selector attribute should be different and should be the values of the corresponding EnumerationType. The converse may be not true: there may be values of the EnumerationType not present as selector attribute in any of the <Cases>. In this case, the union object has the defaultType type.

Types referred to by type and defaultType attributes cannot be union types, but may be group types containing unions.

Example.

<EnumerationType name="AnalogInputCode">

<Item value="Temperature"/>

<Item value="Pole1OpenCount"/>

</EnumerationType>

<UnionType selector="AnalogInputCode" name="FixedDeadBand">

<Case selector="Temperature" type="TemperatureDeadBand"/>

<Case selector="Pole1OpenCount" type="ushort"/>

</UnionType>

## Built-in (Predefined) Types

|  |  |  |
| --- | --- | --- |
| Type name | Description | Equivalent definition |
| Sbyte |  | <NumericType>  <Range min=”-128” max=”127” step=”1”/>  </NumericType> |
| Byte |  | <NumericType>  <Range min=”0” max=”255” step=”1”/>  </NumericType> |
| Short |  | <NumericType>  <Range min=”” max=”” step=”1”/>  </NumericType> |
| Ushort |  | <NumericType>  <Range min=”” max=”” step=”1”/>  </NumericType> |
| int |  | <NumericType>  <Range min=”” max=”” step=”1”/>  </NumericType> |
| uint |  | <NumericType>  <Range min=”” max=”” step=”1”/>  </NumericType> |
| float |  |  |
| double |  |  |
| boolean |  | None |
| gpslatitude |  | None |
| gpslongitude |  | None |
| ipaddress |  | None |
| timerbyte |  | None |
| timestamp | Unix timestamp |  |
| timestampms | DNP timestamp |  |

## Interface Objects

The <Objects> section contains definitions of application objects that this interface defines. The objects are organized as a tree. Each object (a node in a tree) has a name which identifies it in the parent object. The top-most object is the interface itself (the <Objects> element). Each object has a type. The type may be given explicitly as the type=”...” attribute, or be given inline. In this respect, the content of the <Objects> element is analogous to the content of <GroupType> element – the <Objects> element defines the group type of the root object of this interface. The only difference being that the <Objects> element does not contain the <Inherits> and <Union> elements.

Thus, we have the following child elements of the <Objects> element:

* <Group>
* <Array>
* <Object>

The <Group> element, in turn, may have the following child elements:

* <Group>
* <Array>
* <Object>
* <Union>

## Writable Objects

By default, an interface object is not writable. When an object is not writable, an attempt to write a value to it via some IAppDataProvider will result in InvalidOperationException. An interface object may be made writable by specifying the writable=”true” attribute in group type definition or in <Objects> section on the object definition.

# Variable Length Arrays

When an array type definition (named or inline) specifies size=”0”, the array has variable size. That is, the size of the corresponding array may be different at run-time.

Please note that this feature is not implemented in interface profile implementations based on virtual memory, thus it may be used only with data snapshots (xdss, xspt files) or with implementation base on different underlying protocols such as Iocp.

## Use of Identifiers

In interface profile definitions, several object names should be identifiers (that is, contain only letters, digits, and underscore, and start with a letter). There are:

* Interface names
* Object names (simple objects, groups, arrays)
* Type names
* SRGroup names
* Values of EnumerationType
* SpecialValues of NumericType

Profile Implementation Xml File Format

Interface profile implementation files satisfy the schema ProfileVMImplementation\_1.1.xsd. However, the schema does not validate all necessary conditions that should be satisfied for the file to be valid. These conditions (constrains) are explained in this document.

Profile implementation files have extension .xprfimp. Profile implementation may be written in one file, or it may be written in several files, one of which is considered the main file, and the others are “included”.

The root element of profile implementation file is <InterfaceProfileImplementation>. It may contain the following elements:

* Any number of <Include>
* Any number of <InterfaceTypeMappings>
* Any number of <ObjectMappings>
* An optional <EventRecordMapping>

<Include> elements specify additional files to be processed.

<InterfaceTypeMappings> elements define type mappings for types in particular interface.

The <ObjectMappings> element defines object mappings for some or all objects contained in the profile.

<EventRecordMapping> element defines the type mapping for the EventRecord type specified in the profile.

## InterfaceProfileImplementation attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| Name | no | No default | Name of the implementation (corresponds to controller memory map ident) |
| Revision | no | No default | Revision of the implementation (corresponds to the controller memory map revision) |
| implementsInterfaceProfileName | no | No default | Interface profile name this implementation file implements. |
| implementsInterfaceProfileRevision | no | No default | Interface profile revision this implementation file implements. |
| formatRev | yes |  | Version of the file format – should be 1.1. |

# Include Files

The <Include> element has one attribute – filename – which specifies the path to a file to be included. This may be a relative or absolute path. The relative path is evaluated relative to the directory in which the file containing this element is located. There should be no circular references of include files.

The profile implementation parser takes the main xprfimp file as a parameter and then finds all the files referenced by the <Include> elements within it. If the referenced files, in turn, contain <Include> elements, those files are found too. After all the referenced files are found, the parser processes them as a whole. No particular order between files is assumed.

In all collected files that are processed by the parser, there should be only one <EventRecordMapping> element.

All <InterfaceTypeMappings> elements in all files are collected and processed as if they were contained in a single file.

The content of all <ObjectMappings> elements in all files is combined and processed as if it was contained in a single <ObjectMappings> element in one file.

Only main xprfimp file may contain < InterfaceProfileImplementation> element attributes name, revision, implementsInterfaceProfileName, implementsInterfaceProfileRevision. It should specify all these attributes.

All files should have formatRev=”1.1” which is the current supported version of this xml file format.

# Type mappings and object mappings

Type mapping define how a value of particular type is serialized to the virtual memory. The starting address of this serialization is not defined by type mapping. Normally, all data contained in the value is serialized as a contiguous sequence of bytes, but there are exceptions – see <Space> element and allocateSeparately attributes below.

Object mapping defines the entire serialization details for the object, that is, it defines both type mapping for the type of the object and starting address. However, there are exceptions – again, see <Space> and allocateSeparately.

For simple type objects the allocation is most “clean” – they have a starting address and a type mapping that serialize the object value in contiguous block of addresses starting from that starting address.

A similar situation will be for allocation of compound objects (groups, arrays) that are allocated in their entirety in a single <Allocate .../> statement using a type mapping for their type which does not contain allocateSeparately attributes. Such allocations are similar to allocations of simple objects in that respect that we may think of them as having a starting address and a type mapping that perform serialization of the value starting from given address. The serialized object will occupy a contiguous range of virtual memory.

For the objects high in the application profile tree, usually this is not the case. These objects may be allocated as

* In multiple <Allocate .../> statements each specifying the same object but different type mappings and starting address for the object. The type mappings contain allocateSeparately attribute, and thus each allocation serializes only part of object value data.
* A compound object may not be allocated as whole at all. Instead, multiple <Allocate .../> statements may be issued for its components – group and array members.
* A mixture of the above two variants.

In such cases various parts of the compound object are serialized to different places of virtual memory and do not form a contiguous range.

# Type Mappings

The <InterfaceTypeMappings> element has the following attributes, which specifies the interface:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes | No default | Name of an interface, as assigned in the profile. |
| revision | yes | No default | Revision of an interface. Should match the revision of the interface in the profile. |

There should be at most one <InterfaceTypeMappings> element for any particular interface name in the profile. The content of this element defines mappings for the types defined in the corresponding interface.

Type mappings for particular types are defined in the elements whose names have the form <xxxMapping>, where xxx stands for the type category. Thus, there are <GroupMapping>, <EnumerationMapping>, etc.

# Common <xxxMapping> attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | no | No default | Name of this type mapping |
| appType | no | No default | A type for which a mapping is defined |
| object | no | No default | A type mapping is defined for the type of this object |
| isDefault | no | false | If true, this mapping is the default mapping for the type. |

appType, if present, should be a name of a type defined in the corresponding interface (determined by the containing InterfaceTypeMappings element).

object, if present, should be a name of an object in the corresponding interface (determined by the containing InterfaceTypeMappings element).

name and isDefault attributes cannot be present if the object attribute is present.

If appType attribute is present, but name attribute is not, it is assumed to be equal to the value of appType attribute.

The value of the name attribute should not coincide with any of the names of built-in type mappings (see below).

Only one of object and appType attributes may be present.

Since there exist only one instance of Boolean type and IPAddress type, elements <BooleanMapping> and <IPAddressMapping> should not specify appType attribute but may specify the object attribute. All other <xxxMapping> elements should specify either appType or object attribute.

There should be no more than one <xxxMapping> element for each particular value of the object attribute.

There should be no more than one <xxxMapping> element having isDefault=”true” for each particular value of the appType attribute.

When <xxxMapping> element specifies appType attribute, it may specify the name attribute. This name may be used to refer to this mapping in other places of this implementation definition and in run-time.

Note that the names of type mappings are local to the interfaces for which they are defined. E.g., there may be several type mappings with the same name which belong to different interfaces. A name belongs to the interface defined by the surrounding <InterfaceTypeMappings> element. When a type mapping name is used (such as in typeMapping=”…” attribute), the relevant interface is determined by the context – by the interface to which the type in question belongs.

Note also that built-in types belong simultaneously to all interfaces. However, in each interface the built-in types are considered as its own types. Thus, each <InterfaceTypeMappings> may define its own mappings for a built-in type. In mapping selection rules (see below) only mappings defined in an interface in question are considered.

# Simple Type Mappings

## String Mappings

<StringMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| encoding | yes |  | Encoding to be used when serializing the string to byte array. |

Possible values for encoding are:

|  |  |
| --- | --- |
| Encoding name | Description |
| Western | Same as Windows-1252 |
| Windows-1252 | Standard 8-bit Western European Windows encoding. |
| UTF-16 | Standard 16-bit Unicode encoding |
| UTF-16BE | Standard 16-bit Unicode encoding, big-endian. |

Example.

<StringMapping name="Location" appType="Location" encoding="Western"/>

## Numeric Mappings

<NumericMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| byteOrder | No | Normal | Byte order to be used when serializing the value as a sequence of bytes. Ignored, if bitMapping=true. |
| size | Yes |  | Size of the object when serialized into virtual memory. |
| bitMapping | No | false | When “true”, size is in bits, otherwise in bytes. |
| isSigned | Yes |  | Bytes in virtual memory are interpreted as signed number (not to be confused with signed/unsigned status of corresponding application type). |
| float | No | false | When true, the number is stored in memory in IEEE floating point format. |

The <NumericMapping> element may have the following child elements:

* At most one <Scaling> element;
* At most one <FixedPoint> element;
* One or more <SpecialValue> elements.

The <Scaling> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| multiplier | yes |  | A number to multiply by. |
| adder | yes |  | A number to be added to the result of multiplication. |

The <Scaling> element defines a linear transformation which is performed on a value read from virtual memory to obtain an application object value. When an application value is written to memory, a reverse transformation is performed and the result is being put to virtual memory. Thus, the multiplier cannot be zero.

If <Scaling> element is not present, identical transformation is assumed: multiplier=”1.0” and adder=”0.0”.

The <FixedPoint> element has the following attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| lengthInBits | yes |  | A number of binary digits after dot in fixed point representation of the number. |

The <SpecialValue> element attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes |  | Special value, as defined in NumericType definition. |
| mapsTo | yes |  | Numeric representation of this special value in virtual memory. |

If NumericType defines <SpecialValue>s, in <NumericMapping> there should be <SpecialValue>s with exactly same names. There should be no <SpecialValue> elements if <NumericType> does not define them.

All mapsTo values should be different.

All mapsTo values should not belong to the interval of values corresponding to the <Range> interval of <NumericType> under scaling.

If float=”true”, there should be no <FixedPoint> element, bitMapping should be false, isSigned should be true, and size should be equal to 4 or 8.

If the <FixedPoint> element is present, the float attribute should be “false” or absent.

Example.

<NumericMapping name="PowerFactorAngleDeadBand" appType="PowerFactorAngleDeadBand"

isSigned="false" size="2" isDefault="true">

<Scaling adder="0" multiplier="0.125"/>

<SpecialValue name="NA" mapsTo="65535"/>

</NumericMapping>

## GPSLongitude and GPSLatitude Mappings

There are built-in gpslongitude and gpslatitude types and type mappings, but creation of user-defined mappings for these types is not supported.

## Boolean Mappings

<BooleanMapping> element does not specify the appType attribute, because there is only one Boolean type.

<BooleanMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
|  |  |  |  |

<BooleanMapping> element should have one of the two child elements:

* <cstyle>
* <fixed>

<cstyle> element attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| numericMapping | Yes |  | Name of the <NumericMapping> used to store the numeric representation of the Boolean value. |

The <Range> of the numeric type of numericMapping should contain the value 0 and at least one non-zero value.

When <cstyle> element is present, numeric zero value corresponds to “false”, and all other numeric values are interpreted as “true”. When a “false” value is written to virtual memory, then the value of 0 is written. When a “true” value is written, the corresponding provider writes some non-zero value belonging the <Range> of corresponding numeric type. Which particular value will be written is not specified.

<fixed> element attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| numericMapping | yes |  | Name of the <NumericMapping> used to store the numeric representation of the Boolean value. |

The <fixed> element may have the following child elements, defining which numeric values should be treated as “true” and “false” respectively:

* <true>
* <false>

If <true> is not present, value of “1” is assumed. If <false> is not present, value of “0” is assumed. When a numeric value read from virtual memory does not coincide with values given by <true> and <false>, the value of the object is considered invalid.

Values defined by <true> and <false> elements should be valid values for the numeric type of numericMapping.

Example.

<BooleanMapping object="CMD.SetHotLineTag">

<fixed numericMapping="byte"><true>1</true><false>2</false></fixed>

</BooleanMapping>

## Enumeration Mappings

<EnumerationMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| targetTypeMapping | yes |  | Underlying numeric type mapping to be used. |

The targetTypeMapping should be a name of existing <NumericMapping>.

<EnumerationMapping> conatins <Item> child elements which specify the numeric values to be used for each possible value of corresponding enumeration type.

<Item> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| value | yes |  | One of possible values of corresponding EnumerationType. |
| mapsTo | yes |  | Numeric value corresponding for this value. |

For each <Item> element in <EnumerationType> definition there should be an <Item> element with the same value attribute, and vise versa. All mapsTo attribute values should be different. All mapsTo values should be valid values of the numeric type of the targetTypeMapping.

Example.

<EnumerationMapping targetTypeMapping="byte" appType="ActiveTimeSource">

<Item value="ProcessorClock" mapsTo="1"/>

<Item value="RealTimeClock" mapsTo="2"/>

<Item value="GPS" mapsTo="3"/>

</EnumerationMapping>

## SimpleArray Mappings

<SimpleArrayMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| elementTypeMapping | yes |  | Type mapping to be used to serialize array elements. |

elementTypeMapping should be the name of one of the built-in type mappings for the elementType of this SimpleArrayType.

## IPAddress Mappings

<IPAddressMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| byteOrder | yes |  | Byte order to be used when serializing this IP address to virtual memory. |

Possible values for byteOrder attribute:

|  |  |
| --- | --- |
| Attribute value | Description |
| normal | MSB first format (Motorola) |
| reversed | LSB first format (Intel) |
| wordreversed | Same as normal but high and low words are swapped. |

## TimeStamp Mappings

<Mapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| byteOrder | Yes |  | The order in which numeric timestamp representation is serialized to memory. |
| packaging | Yes |  | Timestamp representation as a number |

Possible values for the packaging attribute are:

|  |  |
| --- | --- |
| Packaging name | Description |
| unix | 32-bit number of seconds since Jan 1, 1970 |
| dnp | 48-bit numer of milliseconds since Jan 1, 1970 |
| elsipacked | 32-bit proprietary format of EnergyLine Systems Inc. |

Possible value for byteOrder attribute:

|  |  |
| --- | --- |
| Attribute value | Description |
| normal | MSB first format (Motorola) |
| reversed | LSB first format (Intel) |
| wordreversed | Same as normal but high and low words are swapped. |
| dnp | Same as normal |
| dnpreversed | Same as reversed |

byteOrders dnp and dnpreversed are applicable only to packaging=”dnp”.

byteOrders normal, reversed, and wordreversed are applicable only to packagings unix and elsipacked.

## Version Mappings

<Mapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| byteOrder | Yes |  | The order in which version byte representation is serialized to memory. |

Possible byteOrder values are the same as for IPAddressMapping.

# Group Mappings



A <GroupMapping> element may contain the following elements:

* <Group>
* <Array>
* <Object>
* <Space>

These should be (with the exeption for <Space>) exactly the same elements that are in the corresponding GroupType, but may be in different order. For each <Union> element in group type definition there should be an <Object> element.

The order in which these elements occur in the <GroupMapping> element define the order in which corresponding group members are serialized to virtual memory.

The <Group> element may either specify the attribute typeMapping=”…”, or provide inline type mapping definition by child elements (again, <Group>, <Array>, <Object>, and <Space>).

The <Object> element may specify the attribute typeMapping=”…”, or contain one of the <xxxMapping> elements to make the “inline” type mapping definition, or may specify neither of these. In this case, the type mapping selection rules will be applied to find the type mapping for type of this object.

Common attributes for <Group>, <Array>, and <Object> elements are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes | No default | Name of the corresponding Group, Array, or Object member of the group. |
| typeMapping | no | No default | Use this type mapping for this group member. |
| allocateSeparately | no | false | This group member will not be serialized to virtual memory by this mapping. |



If a <Group> specifies typeMapping, it should not contain child elements. If a <Group> does not specify typeMapping, its child elements should form an inline type mapping definition for this group – that is, all type members should be listed exactly once.

<Array> element attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| elementTypeMapping | no | No default | Specifies the type mapping for the elementType. |

If <Array> specifies typeMapping, it should not specify elementTypeMapping.

If <Array> specify neither typeMapping nor elementTypeMapping, type mapping selection rules will be attempted to find a type mapping for the elementType and thus construct the type mapping for this <Array>.

<Space> element attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| size | yes | No default | Defines the size of the gap between previous and next allocated objects. |
| bitSize | no | false | If bitSize=”yes”, the size units are bits (and not bytes). |

Example.

<GroupMapping name="SerialPort" appType="SerialPort">

<Object name="BaudRate" typeMapping="BaudRateMapping2"/>

<Space size="2"/>

<Object name="RTSActiveBeforeXmitTime"/>

<Object name="RTSActiveAfterXmitTime"/>

<Object name="InterpacketDelay"/>

<Space size="2"/>

<Object name="Protocol"/>

</GroupMapping>

# Array Mappings

<ArrayMapping> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| elementTypeMapping | no | No default | Specifies the type mapping for the elementType. |

Example.

<ArrayMapping elementTypeMapping="MemberPort" appType="TeamMembers" name="MemberPorts"/>

# Union Mappings

The <UnionMapping> element contains a list of <Case> elements which define a type mapping for each possible actual type of this union type.

<Case> element attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| selector | yes |  | One of the possible selector values, as defined in corresponding UnionType. |
| typeMapping | no | No default | Type mapping for the actual type corresponding of this value of selector. |

If typeMapping attribute is not given, the type mapping selection rules will be applied to find the type mapping for the type corresponding to this selector value.

TBD: rules for EnumerationType’s Items and <Case>s of <UnionType>.

Example.

<UnionMapping appType="FixedDeadBand">

<Case selector="Temperature" typeMapping="TemperatureDeadBand"/>

<Case selector="Pole1OpenCount"/>

</ UnionMapping >

# Built-in Type Mappings

There are predefined type mappings for predefined types used by the interface profile parser.

|  |  |  |
| --- | --- | --- |
| Type name | Type mapping names | Default mapping |
| byte | byte | byte |
| sbyte | sbyte | sbyte |
| short | short | short |
| shortReversed |
| ushort | ushort | ushort |
| ushortReversed |
| int | int | int |
| intReversed |
| intWordReversed |
| uint | uint | uint |
| uintReversed |
| uintWordReversed |
| float | float | float |
| floatReversed |
| floatWordReversed |
| double | double | double |
| doubleReversed |
| doubleWordReversed |
| boolean | bitNormal | byteNormal |
|  | bitReversed |
|  | byteNormal |
|  | byteReversed |
|  | wordReversed |
|  | wordNormal |
| gpslatitute | gpslatitute | gpslatitute |
| gpslongitude | gpslongitude | gpslongitude |
| ipaddress | ipaddress | ipaddress |
| timerbyte | timerbyte | timerbyte |
| timestamp | timestamp | timestamp |
| timestampms | timestampms | timestampms |

User type mappings definitions equivalent to predefined mappings

|  |  |
| --- | --- |
| Predefined type mapping name | Equivalent definition |
| sbyte | <NumericMapping name="sbyte" appType ="sbyte" byteOrder="normal" size="1" isSigned="true"/> |
| byte | <NumericMapping name="byte" appType ="byte" byteOrder=" normal" size="1" isSigned="false"/> |
| short | <NumericMapping name="short" appType ="short" byteOrder="normal" size="2" isSigned="true"/> |
| shortReversed | <NumericMapping name="shortReversed" appType ="short" byteOrder="reversed" size="2" isSigned="true"/> |
| ushort | <NumericMapping name="ushort" appType ="ushort" byteOrder="normal" size="2" isSigned="false"/> |
| ushortReversed | <NumericMapping name="ushortReversed" appType ="ushort" byteOrder="reversed" size="2" isSigned="false"/> |
| int | <NumericMapping name="int" appType ="int" byteOrder="normal" size="4" isSigned="true"/> |
| intReversed | <NumericMapping name="intReversed" appType ="int" byteOrder="reversed" size="4" isSigned="true"/> |
| intWordReversed | <NumericMapping name="intWordReversed" appType ="int" byteOrder="wordreversed" size="4" isSigned="true"/> |
| uint | <NumericMapping name="uint" appType ="uint" byteOrder="normal" size="4" isSigned="false"/> |
| uintReversed | <NumericMapping name="uintReversed" appType ="uint" byteOrder="reversed" size="4" isSigned="false"/> |
| uintWordReversed | <NumericMapping name="uintWordReversed" appType ="uint" byteOrder="wordreversed" size="4" isSigned="false"/> |
| float | <NumericMapping name=”float” appType=”float” isFloat=”true” byteOrder=”normal” size=”4” isSigned=”true”/> |
| floatReversed | <NumericMapping name=”float” appType=”float” isFloat=”true” byteOrder=”reversed” size=”4” isSigned=”true”/> |
| floatWordReversed | <NumericMapping name=”float” appType=”float” isFloat=”true” byteOrder=”wordreversed” size=”4” isSigned=”true”/> |
| double | <NumericMapping name=”double” appType=”double” isFloat=”true” byteOrder=”normal” size=”8” isSigned=”true”/> |
| doubleReversed | <NumericMapping name=”double” appType=”double” isFloat=”true” byteOrder=”reversed” size=”8” isSigned=”true”/> |
| doubleWordReversed | <NumericMapping name=”double” appType=”double” isFloat=”true” byteOrder=”wordreversed” size=”8” isSigned=”true”/> |
| bitNormal | <NumericType name=”onebit”>  <Range min=”0” max=”1” step=”1”>  </NumericType>  <NumericMapping name=”onebit” appType=”onebit” size=”1” bitSize=”true” isSigned=”false”/>  <BooleanMapping name=”bitNormal”>  <fixed numericMapping=”onebit”>  <true>1</true><false>0</false>  </fixed>  </BooleanMapping> |
| bitReversed | <NumericType name=”onebit”>  <Range min=”0” max=”1” step=”1”>  </NumericType>  <NumericMapping name=”onebit” appType=”onebit” size=”1” bitSize=”true” isSigned=”false”/>  <BooleanMapping name=”bitNormal”>  <fixed numericMapping=”onebit”>  <true>0</true><false>1</false>  </fixed>  </BooleanMapping> |
| byteNormal | <BooleanMapping name=”byteNormal”>  <fixed numericMapping=”byte”>  <true>1</true><false>0</false>  </fixed>  </BooleanMapping> |
| byteReversed | <BooleanMapping name=”byteReversed”>  <fixed numericMapping=”byte”>  <true>0</true><false>1</false>  </fixed>  </BooleanMapping> |
| wordNormal | <BooleanMapping name=”wordNormal”>  <fixed numericMapping=”word”>  <true>1</true><false>0</false>  </fixed>  </BooleanMapping> |
| wordReversed | <BooleanMapping name=”wordReversed”>  <fixed numericMapping=”word”>  <true>0</true><false>1</false>  </fixed>  </BooleanMapping> |
| ipaddress | <IPAddressMapping name=”ipaddress” appType=”ipaddress” byteOrder=”normal” /> |
| gpslatitute | <GPSLatitudeMapping name= “gpslatitude” appType=”gpslatitude” byteOrder=”normal” /> |
| gpslongitude | <GPSLongitudeMapping name= “gpslongitude” appType=”gpslongitude” byteOrder=”normal” /> |
| timestamp | <TimeStampMapping name= “timestamp” appType=”timestamp” byteOrder=”normal” packaging=”Unix”> |
| timestampms | <TimeStampmsMapping name= “timestampms” appType=”timestampms” byteOrder=”normal” packaging=”DNP”> |

# Automatic type mappings

There are several places where a type mapping for a particular application type should be chosen. These are:

* Type mapping for <Object>, <Group>, or <Array> elements inside a group mapping definition <GroupMapping>;
* Type mapping for the type of object which is allocated in the <Allocate> element;
* Type mapping for one of the possible actual types of a union type in <Case> element of <UnionMapping>.

This needed type mapping may be

* given explicitly by the typeMapping attribute;
* given explicitly for the <Object> element as a child <xxxMapping> element;
* a default type mapping for the type chosen;
* the parser may construct an appropriate type mapping.

The exact ***type mapping selection rules*** follow:

First, try to use explicit user supplied mapping (note that only one of them may exist simultaneously):

* If the typeMapping attribute is given, then the type mapping designated by this attribute is used. Otherwise,
* If it is an <Object> element and it contains an <xxxMapping> element, that mapping is used. Otherwise,
* If it is an <Allocate name=”yyy” … /> element, and there exist <xxxMapping> for the object=”yyy”, this type mapping is used. Otherwise,

Second, try to find the default mapping, if exists:

* If there is one or more type mapping for the type and one of them is marked by isDefault=”true”, that mapping is used. Otherwise,
* If the type is a built-in type, the default built-in type mapping for this type is used. Otherwise,
* If there is exactly one user-defined type mapping for this type, it is used. Otherwise,
* If there two or more user-defined type mappings for this type, selection fails and the parser reports an error – type mapping cannot be selected due to ambiguity.

Finally, when the type is not a built-in type and there are no user-defined type mappings for this type, the parser attempts to automatically generate the type mapping. If that fails, the parser reports an error.

***Automatic generation of a type mapping for a particular (not built-in) type*** proceeds as follows:

|  |  |
| --- | --- |
| Type Category | Type mapping constructed |
| GroupType | GroupMapping is constructed listing all group members sequentially as they appear in group type definition. For each member, the type mapping selection rules are applied (with exception of looking for typeMapping attribute, which is obviously not present) to find a mapping of the type of the member. If that fails for any member, report an error. |
| ArrayType | ArrayMapping is constructed by applying the type mapping selection rules to the elementType of the array. If type mapping for elementType could not be determined, report an error. |
| UnionType | UnionMapping is constructed by listing all <Case> members. For each member, the type mapping selection rules are applied (with exception of looking for typeMapping attribute, which is obviously not present) to find a mapping of the type of the corresponding member. If that fails for any member, report an error. |
| NumericType, EnumerationType, TimeStampType | Report an error – these types does not have automatic mappings. |
| StringType | StringMapping is constructed by setting encoding=”en-US”. |
| SimpleArrayType | SimpleArrayMapping is constructed by setting elementTypeMapping to the default built-in type mapping of its elementType. |
| VersionType, IPAddressType | Mapping is constructed by setting byteOrder attribute to Normal. |

In a sense, type mapping selection algorithm and automatic mapping generation algorithm perform mutual recursion: type mapping selection algorithm ultimately calls automatic mapping generation algorithm when attempts to find already existing applicable mapping fail, and automatic mapping generation algorithm calls type mapping selection algorithm for member types to build compound types – arrays, groups, and unions – which ultimately may result in executing the automatic mapping generation for these member types.

# Parser implementation details

Automatic mapping generation is implemented as a three-step process in the parser.

* Step 1. Build a table of all defined named type mapping for application types and type mappings for particular objects, without actually building those type mappings. This table serves as a look-up table for the type mapping selection algorithm.
* Step 2. Verify the correctness – that it is possible to generate all automatic type mappings where needed. For this verification, the table constructed in Step 1 is sufficient, without the actually built type mappings.
* Step 3. Actually build all the needed automatically generated type mappings.

Steps 2 and 3 may be combined.

# Object Mappings

The <ObjectMappings> element contains a sequence of <Segment> elements.

<Segment> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes |  | Name of the segment. Currently not used. |
| address | yes |  | Non-negative integer. Starting address for object allocations in this segment. |

The <Segment> element may contain the following elements:

* <Allocate>
* <AllocateRange>
* <Context>
* <Space>

The <Space> element defines the gap in memory between allocated objects.

The <Allocate> element allocates a particular application object.

The <AllocateRange> element allocates a range of elements of an array application object.

The <Context> element may contain the following elements:

* <Allocate>
* <AllocateRange>
* <Space>

It defines a common name prefix, which is prepended to all names of objects that are <Allocate>ed within it.

<Context> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| path | yes |  | Part of the name which is prepended to all object names of <Allocate> elements within it. |

Thus,

<Context path=”A”>

<Allocate name=”B” … />

</Context>

is effectively equivalent to

<Allocate name=”A.B” …/>.

The <Allocate> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes |  | Name of an object to be allocated. |
| typeMapping | no | No default | Type mapping to be used for this object allocation. |

Name (combined with path prefix from enclosing <Contex> element, if any) should specify a full name (starting from interface name) of an existing object in the interface profile.

typeMapping attribute, if present, should specify a name of a type mapping for the type of the object being allocated.

If typeMapping attribute is present, there should be no <xxxMapping> element specifying the same application object in its object=”…” attribute.

The <AllocateRange> element attributes are:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Required | Default | Description |
| name | yes |  | Name of an array object whose elements are to be allocated. |
| elementTypeMapping | no | No default | Type mapping to be used for array elements allocation. |
| index | yes |  | Starting index of the array elements range. |
| count | yes |  | Length of the array elements range. |

Name (combined with path prefix from enclosing <Contex> element, if any) should specify a full name (starting from interface name) of an existing array object in the interface profile.

elementTypeMapping attribute, if present, should specify a name of a type mapping for the type of the array elements being allocated.

If elementTypeMapping attribute is present, there should be no <xxxMapping> element specifying the same array member object in its object=”…” attribute.

index and count should be non-negative numbers such that index + count <= size of the corresponding array.

Example.

<Segment name="S1" address="0x00250000">

<Context path="SiteRelated.SPT">

<Allocate name="ForwardCurrentDirection"/>

<Allocate name="SystemFrequency"/>

<Allocate name="PhaseRotation"/>

<Allocate name="SystemType"/>

<Allocate name="SystemVoltage"/>

<Allocate name="VoltageReporting"/>

</Context>

</Segment>

# Byte and Bit Addresses

When an application object of simple type is mapped to virtual memory, it occupies some range of virtual memory, starting from some address and having some length. The length of range of virtual memory occupied by the object is determined by its type mapping. The type mapping may specify the required length of virtual memory range in bytes or in bits. Specifically, if bitMapping attribute of <NumericMapping> has the value “true”, the range is measured in bits and the object may occupy a fractional number of bytes. The <BooleanMapping> and <EnumerationMapping> mappings rely on <NumericMapping> to store their numeric representation in memory, and hence may also occupy fractional number of bytes of memory.

Thus, a Boolean, Enumeration, or Numeric mapping is a bit-mapping if the underlying Numeric mapping specifies bitMapping=”true”. All other mappings are byte-mappings.

# Object Memory Allocation

The exact rules are as follows:

Objects in successive <Allocate>, <AllocateRange>, and <Space> elements within a <Segment> are allocated successively in memory.

Group members in <GroupMapping> are allocated successively.

Array members of an array are allocated successively.

Union members are all allocated starting from the same address – the starting address of the union object.

*Successive allocation* means the following algorithm:

The parser maintains the “current allocation address”, expressed in bits.

Initially, the current allocation address is set the value of the address attribute of <Segment> element (by multiplying by eight).

When a next successive object needs to be allocated, do the following:

* If the mapping of the object is byte-mapping, advance the current allocation address to the nearest byte address (that is, to the bit-address evenly dividable by 8);
* The resulting current allocation address is the address of the object;
* Byte-mapped objects occupy successive bytes starting from the current allocation address. Bit-mapped objects occupy successive bits starting from the current allocation address. If the byte in which the bits are allocated overflows, the rest of the bits are allocated in the next byte, etc.
* Advance the current allocation address by the object size as specified by the object type mapping. This is the new current allocation address.
* Continue with next successive object.

When a <Space> element is encountered (as a member of <GroupMapping> or <Segment>/<Contex> elements), it is treated as object allocation with bit-Mapping, if bitSize=”true” is specified, and byte-Mapping, if bitSize=”false”.

# Object sizes in virtual memory

The following table describes the size of memory a serialized object value will occupy, for different type mappings.

|  |  |
| --- | --- |
| Type Mapping | Size |
| NumericMapping | Equals to the size attribute, in bytes or in bits depending on the bitMapping attribute. |
| EnumerationMapping | Equals to the size of underying numeric mapping targetTypeMapping. |
| BooleanMapping | Equals to the size of underlying numericMapping. |
| StringMapping | For Western and Windows-1252 encodings, equals to the string type length. For UTF-16 and UTF-16BE encodings equals to twice the string type length. |
| IPAddressMapping | 4 |
| GPSLongitudeMapping | 6 |
| GPSLatitudeMapping | 6 |
| VersionMapping | Product of numberCount and numberSize attributes of version type. |
| OctetStringMapping | Equals the length of the octet string type. |
| SimpleArrayMapping | Product of array length and the size of elementTypeMapping. |
| PasswordStringMapping | 16 |
| TimestampMapping | 4 for packagings unix and elsipacked, 8 for dnp. |
| UnionMapping | Maximum of the sizes of union members as determined by their respective type mappings. |
| GroupMapping | Sum of the sizes of group members as determined by their respective type mappings. |
| ArrayMapping | Size of the elementTypeMapping multiplied by the length of the array. |

# Allocation correctness checks

After all <Allocate>s in all <Segment>s are processed, all interface profile objects should be allocated – that is, for all simple type objects contained in the interface (as leaf nodes of the object tree) a starting address should be assigned. If there exists a simple type object without assigned starting address, an error is reported.

There is no any specific order in which <Segment> elements are processed, especially because they may be contained in different xprfimp files which are recursively included from the main xprfimp file – no specific order of processing the <Include> elements is defined.

It may happen that one and the same simple type object is allocated more than once – either explicitly or as part of a containing compound object. This is allowed, but a warning message is issued. The starting address is set in the first allocation and is not changed on subsequent allocations of the same object.

Since there is no a specific order in which the <Segment>s are process, there is no guarantee that an object will get any specific address – it will depend on the order in which the <Segment>s are processed.

This should be avoided. allocateSeparately technique should be used to avoid multiple allocation for the same object.

The parser does not check that different objects are allocated to the same memory location. In particular, there may be <Segments> with the same starting address attribute.

# Variable length arrays

Implementation of an interface profile to the virtual memory described in this document does not support objects having Array and SimpleArray types with variable length (size attribute equals zero) defined in the <Objects> section. However, it does support defining mappings for variable length Arrays and SimpleArrays types in <InterfaceTypeMappings> sections.

# Parser Errors

Error code in the text is now written in **bold**.

*Italic* in the text is the place for which there is a bug*.*

## Profile parser errors with examples

<InterfaceProfile name="<interface profile name>" …

<Interfaces>

<Interface uri="<uri>" revision="<expected revision>"/>

</Interfaces>

</InterfaceProfile>

…

<Interface uri="<uri>" revision="*<found revision>*"…

…

**Error** **EAP0101:** InterfaceProfile name=`<interface profile name>` requires different revision of `<Interface name>` Interface - expected `<expected revision>`, found *`<found revision>`.* Location: 'path.xintdef'

Or

<InterfaceProfile name="<interface profile name>" …

<Interfaces>

<Interface uri="<expected uri>" …/>

</Interfaces>

</InterfaceProfile>

…

<Interface uri="<*found uri*>" …

…

**Error EAP0101**: InterfaceProfile name=`<interface profile name>` requires different Interface uri - expected `<expected uri>`, found `<*found uri*>`. Location: 'path.xintdef'

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …

<EventRecordType>IREvents.HistEventRecord\_CFE</EventRecordType>

…

**Error EAP0102**: File=`xx.xprfdef` InterfaceProfile name=`<interface profile name>` - appType `IREvents.HistEventRecord\_CFE` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Object name="<object name>" type="<*array type name*>"/>

…

**Error EAP0702**: Interface uri=`<Interface name>` GroupType=`<GroupType name>` Object=`<object name>` type=`<array type name>` appType category = `*Array*` - type attribute should specify a simple type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Group name="<group name>" type=*"int"*/>

…

**Error EAP0703**: Interface uri=`<Interface name>` GroupType=`<group type name>` Group name=`<group name>` type=*`int`* appType category=`*Numeric*` - type attribute should specify a group type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Array name="array name" type=*"int"* writable="true">

…

**Error EAP0704**: Interface uri=`<Interface name>` GroupType=`<group type name>` Array=`<array name>` type=*`int`* - type attribute should specify an array type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<ExternalTypes>

<FromInterface uri="<uri>" …>

<ExternalType name="<aname>" />

<ExternalType name="<name>" alias="*<aname>*"/>

…

**Error EAP1001**: Interface name=`<interface>` - ExternalType alias=`*<aname>*` already exists

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri1>" …>

<ExternalTypes>

<FromInterface uri=*"<uri2>"* revision="<revision>">

…

**Error EAP1004**: Interface uri=`<uri1>` name=`<interface>` ExternalTypes.FromInterface uri=*`<uri2>`* revision=`<revision>` - interface *`<uri2>`* not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri1>" …>

<ExternalTypes>

<FromInterface uri="<uri2>" …>

<ExternalType name=*"<name>"* alias="<alias name>"/>

…

Lets *<name>* is not defined

**Error EAP1102**: Interface `<uri1>` ExternalType name (alias)=*`* <alias name>*`* - appType `<uri2.aliasname>` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<groupType1>">

<Object name="object1" type=*"z1"*/>

</GroupType>

<GroupType name="<groupType2>">

<Object name="object2" type=*"z2"*/>

…

Lets *z1 and z2* are not defined

**Error EAP1401**: There is [are] unresolved type[s] - interface=`<interface>`: type=*`z1`*, GroupType= `<groupType1>`, type= *`z2`*, GroupType= `<groupType2>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>" units=*" kw"*/>

…

**Error EAP1502**: Interface name=`<interface>` NumericType name=`<name>` units=*` kw`* - leading or trailing white space is not allowed in units attribute

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>"><Range *min="0" max="1" step="0"*/></NumericType>

…

**Error EAP1503**: Interface name=`<interface>` NumericType name=`<name>` Range: *min=`0` max=`1` step=`0`* - min not equal to max and step equal to 0 are not allowed simultaneously

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>">

<Range min=*"100000.1"* max=*"1440.1"* step="0.1"/>

…

**Error EAP1506**: Interface name=`<interface>` NumericType name=`<name>` - incorrect Range: min= *`100000.1`* should be less or equal to max= *`1440.1`*

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>">

<Range min="10.1" max="1440.1" step="0.1"/>

<SpecialValue name="one" value*="1000"*/>

<SpecialValue name="two" value="1450"/>

…

**Error EAP1508**: Interface name=`<interface>` NumericType name=`<name>` - SpecialValue=`*1000*` should not lie in the Range: min=`10.1` max=`1440.1`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>" defaultValue="65535">

<Range min="10.1" max="1440.1" step="0.1"/>

<SpecialValue name="one" value*=*"1000"/>

…

**Error EAP1510**: Interface name=`<interface>` NumericType name=`<name>` - defaultValue=`65535` should belong to allowed NumericType values (be in Range or one of SpecialValues)

\_\_\_\_\_\_\_\_\_\_\_\_\_

<GroupType name="<group type>">

<Object name="<Obj1>">

<StringType length="1" />

</Object>

</GroupType>

<GroupType name="<inheritGT>">

<Inherits type="<group type>"/>

<Object name="*<Obj1>*">

<StringType length="2" />

</Object>

</GroupType>

…

**Error EAP1606**: Interface name=`<interface>` `GroupType` name=`<inheritGT>` - `Object` name= `*<Obj1>*` already exists in base GroupType= `<group type>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<GroupType name="<group type>">

<Object name=*"<Obj1>"*>

<StringType length="1" />

</Object>

</GroupType>

<GroupType name="<inheritGT>">

<Object name="<Obj1>">

<StringType length="2" />

</Object>

<Inherits type="<group type>"/>

</GroupType>

…

**Error EAP1607**: Interface name=`<interface>` `GroupType` name=`<inheritGT>` Inherits type=`<group type>` - `Object` name=`*<Obj1>*` already exists

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<name>" type=*"<undefined type>"*/>

…

**Error EAP2001**: Interface name=`<interface>` Object name=`<name>` type/elementType=*`<undefined type>`* - AppType *`<undefined type>`* not found

Or

<Objects>

<Array name="<name>" elementType=*"<undefined type>"* size="4" />

…

**Error EAP2001**: Interface name=`<interface>` Array name=`<name>` type/elementType=*`<undefined type>`* - AppType *`<undefined type>`* not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<name>" type=*"<array / group type>"*/>

…

**Error EAP2203**: Interface name=`<interface>` Object name=`<name>` type=*`<array / group type>`* - type attribute should specify a simple type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object name>" type="<numeric type>" units=*"kw "*/>

…

**Error EAP2204**: Interface name=`<interface>` Object name=`<object name>` units=*`kw `* - leading or trailing white space is not allowed in units attribute

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object name>" type=*"boolean"* units="kw"/>

…

**Error EAP2205**: Interface name=`<interface>` Object name=`<object name>` units=`kw` type=*`Boolean`*- units attribute may be applied only to numeric type objects

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<name>" units="kw">

<Range min="10.1" max="1440.1" step="0.1"/>

<Objects>

<Object name="<object name>" type="<numeric type>" *units="ohms"*/>

…

**Error EAP2206**: Interface name=`<interface>` Object name=`<object name>` *units= `ohms`* type units= `kw`- unit already defined in type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Group name="<name>" type=*"<undefined type>"*/>

…

**Error EAP2301**: Interface name=`<interface>` Group name=`<group name>` - AppType *`<undefined type>`* not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Group name="<name>" type=*"<numeric type>"*/>

…

**Error EAP2303**: Interface name=`<interface>` Group name=`<name>` type= `<numeric type>` category= *`Numeric`* - type attribute should specify a group type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Group name="<group name>" *type*="<group type>">

*<Object name "<object>" …/>*

…

**Error EAP2305**: Interface name=`<interface>` `Group` name=`<group name>` type=`<group type>` `Object` name=`<object>` - the simultaneous presence of the type attribute and element[s] is not allowed in the Group

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Array name="<array name>" type=*"boolean"* />

…

**Error EAP2402**: Interface name=`<interface>` Array name=`<array name>` type= *`boolean`* category= *`Boolean`* - type attribute should specify an array type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Array name="<array name>" *type*="<array type>" *elementType="int" size="4"*/>

…

**Error EAP2403**: Interface name=`<interface>` Array name=`<array name>` *type*= `<array type>` *elementType= `int` size=`4`* - array cannot specify type attribute simultaneously with *elementType* or *size* attribute

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Array name="<array name>" size="4"/>

…

**Error EAP2405**: Interface name=`<interface>` Array name=`<array name>` - elementType is missing

\_\_\_\_\_\_\_\_\_\_\_\_\_

<GroupType name="<group type name>">

<Object name="Object2"*>*

*<*/Object>

</GroupType>

…

**Error EAP2605**: Interface name=`<Interface name>` GroupType/Group name=`<group type name>` object name=`<Object2>`- object type not specified. Object type may be specified either by type=`...` attribute or by child element such as <StringType>, <NumericType>, etc

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" type="<group type1>">

*<Object name="<object>" …/>*

…

**Error EAP2801**: Interface name=`<interface>` GroupType name=`<group type>` Group name=`<group>` type=`<group type1>` `Object` name=`<object>` - the simultaneous presence of the type attribute and element[s] is not allowed in the Group

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" *><*/Group>

…

**Error EAP2803**: Interface name=`<interface>` GroupType name= `<group type>`- element <Group name= `<group>`/> defines empty application group which is not allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

<Array name="<array>" *type*="<array type>" *size*="1">

…

**Error EAP2902**: Interface name=`<interface>` Array name=`<array>` *type*=`<array type>` *size*=`1` - if type of array is specified by type attribute, size and elementType attributes cannot be present

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

<Array name="<array>" *elementType*="int" *type*="int">

…

**Error EAP2903**: Interface name=`<interface>` Array name=`<array>` *type*=`<array type>` *elementType*=`int`- if type of array is specified by type attribute, size and elementType attributes cannot be present

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<TimeStampType name="<timestamp type>" precision="milliseconds" from = *"01/31/2010T13:00:00….001"* to = "01/31/2010T13:00:00.003"/>

…

**Error EAP3101**: Interface name = `<interface>` TimeStampType = `<timestamp type>` precision = `milliseconds` format = `MM/dd/yyyyTHH:mm:ss.fff` from = `*01/31/2010T13:00:00….001*` - invalid `from` string: String was not recognized as a valid DateTime.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<TimeStampType name="<timestamp type>" precision="milliseconds" from = "12/29/1949T13:00:00.*002*" to = "12/29/1949T13:00:00.*001*"/>

…

**Error EAP3102**: Interface name = `<interface>` TimeStampType = `<timestamp type>` from = `12/29/1949T13:00:00.*002*` to = `12/29/1949T13:00:00.*001*` - `from` time should be earlier or equal to `to` time

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri >" …>

<ExternalTypes>

<FromInterface uri="<ext uri >" revision=*""*>

…

**Error EAP3401**: Interface name=`<interface>` FromInterface uri=`<ext uri>` revision=*``* - invalid `revision` string: Version string portion was too short or too long.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="SpUn1En">

<Item value="Case1"/>

<Item value="Case2"/>

</EnumerationType>

<UnionType name="SpUn1Un" selector="SpUn1En">

<Case selector=*"Case"* type="uint"/>

<Case selector="Case2" type="int"/>

…

**Error EAP3801**: Interface name=`<interface>` UnionType name=`SpUn1Un` - attribute selector value *`Case`* should belong to selector type `SpUn1En`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="SpUn1En">

<Item value="Case1"/>

<Item value="Case2"/>

</EnumerationType>

<UnionType name="SpUn1Un" *selector="int"*>

<Case selector="Case1" type="uint"/>

<Case selector="Case2" type="int"/>

…

**Error EAP3802**: Interface name=`<interface>` UnionType name=`SpUn1Un` selector*=`int`* - selector *type* should be EnumerationType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="SpUn1En">

*<Item value="Case1"/>*

<Item value="Case2"/>

</EnumerationType>

<UnionType name="SpUn1Un" selector="SpUn1En*"*>

<Case selector="Case2" type="int"/>

…

**Error EAP3805**: Interface name=`<interface>` UnionType name=`SpUn1Un` - type not defined for selector= *`Case1`*. Either in <Case> or defaultType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<AppData profileName="<profile name>" profileRevision="<profile revision>" …>

<Interface uri="<interface uri>" …>

<Object *nam*="<object name>" …>

…

**Error EDP0101**: Schema validation error in Line 3 Position 187: The '<nam>' attribute is not declared.

\_\_\_\_\_\_\_\_\_\_\_\_\_

appProfileParser.Main(<*profile name*>, <resolver>);

…

**Error EII0101**: Could not find file '<*profile name*>'. Location: `<full path *profile name*>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface *asdfg* uri="<uri>" …>

…

**Error EII0104**: Interface uri=`<uri>` is incorrect. Invalid xml file: 'uri' is an unexpected token. The expected token is '='. Line 2, position 18. Location: `<full path name>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<Interfaces>

<Interface uri="<uri>" revision="<rev>"/>

…

**Error EII0108**: profile name=`<interface profile name>` - Interface name `xxx` generated from uri `<uri>` already exists

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type name>" *length="-6"*/>

…

**Error EII0201**: Interface=`<interface>` - scheme validation error in Line 107 Position 31: The 'length' attribute is invalid - The value '-6' is invalid according to its datatype 'StringLengthType' - The MinInclusive constraint failed. Location: `<interface uri>`

Or

<Types>

<ArrayType name="<array type name>" size="3"/>

…

**Error EII0201**: Interface=`<interface>` - schema validation error in Line 83 Position 7: The required attribute 'elementType' is missing. Location: `<full path name>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri >" revision=*"01.0.0.09999999999"* …>

…

**Error EII0301**: Interface uri=`<uri>` revision=*`01.0.0.09999999999`* - invalid revision value: Value was either too large or too small for an Int32. Location: `<full path name>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

appProfileParser.Main(*<profile name>*, <resolver>);

…

**Error EIP0100**: Could not find file '*<profile name>*'.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile *qwert* name="<interface profile name>"

…

**Error EIP0103**: InterfaceProfile is incorrect. Invalid xml file: 'name' is an unexpected token. The expected token is '='. Line 2, position 24. Location: `<full path name>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<Interfaces>

<Interface uri="<uri>" name="*path*\name" …/>

…

**Error EIP0104**: file=`<profile definition file name.xprfdef>` Interface uri=`<uri>` name=`*path*\name`. Backslash in Interface name not allowed if uri is present

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<Interfaces>

<Interface revision="<revision>"/>

…

**Error EIP0105**: file=`xxx.xprfdef` . Uri or name attribute should be specify

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<SaveRestoreGroups file=*"<srgroups file name>"*>

…

**Error EIP0108**: file=`<profile definition file name.xprfdef>`. SaveRestoreGroups file=*`<srgroups file name>`* - setpoint definition file not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<Interfaces>

<Interface uri="<uri>" name="<name1>" revision="01.0.0.0"/>

<Interface uri=*"<uri>"* name="<name2>" revision="01.0.0.0"/>

…

**Error EIP0109**: file=`<profile definition file name.xprfdef>` Interface uri=*`<uri>`* name=`<name2>` - Interface with the same uri already exists (Interface uri=`<uri>` name=`<name1>`)

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" …>

<Interfaces>

<Interface uri="*path*\88name" …/>

…

**Error EIP0112**: file=`<profile definition file name.xprfdef>` Interface uri=`*path*\88name` - the `name` attribute is missing. Could not deduce name from uri

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile asdf name="<interface profile name>"

<Description>interface profile<Description>

…

**Error EIP0201**: file=`<profile definition file name.xprfdef>`. Syntax error in Line 12 Position 33: The element 'Description' cannot contain child element 'Description' because the parent element's content model is text only.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interface profile name>" revision="*1. 99999999999*"…>

…

**Error EIP0301**: InterfaceProfile name=`<interface profile name>` revision=`*1. 99999999999*` - Value was either too large or too small for an Int32.

\_\_\_\_\_\_\_\_\_\_\_\_\_

## Profile implementation parser errors

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="enumUT">

<Item value="Case1"/>

</EnumerationType>

<EnumerationType name="enum">

<Item value="Direction1"/>

</EnumerationType>

<UnionType name="UT" selector="enumUT">

<Case selector="Case1" type="enum"/>

</UnionType>

…

<InterfaceTypeMappings name="<interface>" …>

<UnionMapping name="UM" appType="UT">

<Case selector="Case1" *typeMapping="uint"*/>

…

**Error EIP0401**: InterfaceTypeMappings=`interface` UnionMapping name=`UM` `appType`=`UT` Case selector=`Case1` - typeMapping=*`interface.uint`* is not a mapping of the type `interface.enum`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="ET">

<Item value="Case1"/>

<Item value="Case2"/>

</EnumerationType>

<UnionType name="UT" selector="ET">

<Case selector="Case1" type="uint"/>

<Case selector="Case2" type="int"/>

</UnionType>

…

<InterfaceTypeMappings name="<interface>" …>

<UnionMapping name="UM" appType="UT">

<Case selector="Case1" typeMapping="zzz"/>

<Case selector="Case2" typeMapping="int"/>

…

**Error EIP0404**: InterfaceTypeMappings=`<interface>` UnionMapping name=`UM` `appType`=`UT` Case selector=`Case1`- typeMapping `zzz` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="ET">

<Item value="Case1"/>

</EnumerationType>

<UnionType name="UT" selector="ET">

<Case selector="Case1" type="uint"/>

</UnionType>

…

<InterfaceTypeMappings name="<interface>" …>

<UnionMapping name="UM" appType="UT">

<Case selector="Case1" typeMapping="int"/>

…

**Error EIP0405**: InterfaceTypeMappings=`<interface>` UnionMapping name=`UM` selector=`Case1` typeMapping=`int` corresponding UnionType name=`SpUn1Un` selector=`Case1` - typeMapping `Interface.int` is not a mapping of the type `Interface9SPT.uint`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="ET">

<Item value="Case1"/>

<Item value="Case2"/>

</EnumerationType>

<UnionType name="UT" selector="ET">

<Case selector="Case1" type="uint"/>

<Case selector="Case2" type="int"/>

</UnionType>

…

<InterfaceTypeMappings name="<interface>" …>

<UnionMapping name="UM" appType="UT">

<Case *selector="Case"* typeMapping="uint"/>

<Case selector="Case2" typeMapping="int"/>

…

**Error EIP0406**: InterfaceTypeMappings=`interface` UnionMapping name=`UM` `appType`=`UT` - case selector attribute `Case` is not a member of selector type `ET`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="ET">

<Item value="Case1"/>

<Item value="Case2"/>

</EnumerationType>

<UnionType name="UT" selector="ET">

<Case selector="Case1" type="uint"/>

<Case selector="Case2" type="int"/>

</UnionType>

<InterfaceTypeMappings name="<interface>" …>

<UnionMapping name="UM" appType="UT">

<Case selector*=*"Case1" typeMapping="uint"/>

*<Case selector="Case2" />*

…

**Error EIP0407**: InterfaceTypeMappings=`interface` UnionMapping name=`UM` `appType`=`UT` - default mapping for Case selector `Case2` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

IInterfaceResolver resolver = new StandardInterfaceResolver("*path*");

…

**Error EIR0102**: The directory `*path*` does not exist. Location: `<interface file name>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

AppInterfaceProfileImplementation aipi = profileImplementationParser.Load("<profile implementation file name.*xprfim*>");

…

**Error EPI0200**: Could not find file '<profile implementation file name.*xprfim*>'.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfileImplementation *asdf* name="<interface profile implementation name>" …>

…

**Error EPI0203**: file=`<profile implementation file name.xprfimp>` - InterfaceProfileImplementation is incorrect. Not a valid xml file: 'name' is an unexpected token. The expected token is '='. Line 2, position 38.

\_\_\_\_\_\_\_\_\_\_\_\_\_

File : xxx.xprfdef -

<InterfaceProfile name="<interface profile name>" revision=*"3.1.1.0"*…>

File : xxx.xprfimp -

<InterfaceProfileImplementation name="<interface profile implementation name>" implementsInterfaceProfileName="<interface profile name>" implementsInterfaceProfileRevision=*"3.1.1.099"*…>

…

**Error EPI0204**: implementsInterfaceProfileName=`<profile name>` - implementsInterfaceProfileRevision differs from InterfaceProfile revision: expected *`3.1.1.0`,* found *`3.1.1.99`*. File=`xxx.xprfimp` Line 6 Position 33"

\_\_\_\_\_\_\_\_\_\_\_\_\_

File : xxx.xprfdef -

<InterfaceProfile name=*"<expected interface profile name>"* …>

File : xxx.xprfimp -

<InterfaceProfileImplementation name="<interface profile implementation name>" implementsInterfaceProfileName=*"<found interface profile name>"* …>

…

**Error EPI0205**: implementsInterfaceProfileName differs from InterfaceProfile name: expected *`<expected interface profile name>`,* found *`<found interface profile name>`*

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfileImplementation name="<name>" …>

<Include fileName="<file name.xprfimp>"/>

<Include fileName=*"<file name.xprfimp>"*/>

…

**Error EPI0207**: InterfaceProfileImplementation=`<name>` - duplicate <Include> element with the same fileName= *`<file name.xprfimp>`*. File= `<name>` Line 15 Position 12

\_\_\_\_\_\_\_\_\_\_\_\_\_

File xxx.xprfimp:

<InterfaceProfileImplementation name="<name>" …>

<Include fileName="<name.xprfimp>"/>

…

File fileName.xprfimp:

<InterfaceProfileImplementation name="<name>" …>

<InterfaceTypeMappings name=<interface1> …>

…

<InterfaceTypeMappings name=*<interface1>* …>

…

**Error EPI0208**: InterfaceProfileImplementation name=`` - duplicate InterfaceTypeMappings for the name=*<interface1>*. File= `fileName.xprfimp` Line 1100 Position 4

\_\_\_\_\_\_\_\_\_\_\_\_\_

File fileName.xprfimp:

<InterfaceProfileImplementation *name=""* …>

…

Error **EPI0209**: InterfaceProfileImplementation *name=``* - attribute name must not be null or empty in the head file. File=`fileName.xprfimp `

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Object name="<object1>" type="boolean"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group mapping name>" appType=… >

<Object name="<object1>" typeMapping="bitNormal"/>

<Object name=*"<object1>"* typeMapping="bitNormal"/>

…

**Error EPI0701**: Interface name=`<interface>` - more than one mapping for `Object` name=*`<object1>`* in the `GroupMapping` name=`<group mapping name>` found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Object name="<object1>" type="boolean"/>

</GroupType>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping appType="<group type name>">

<Object name="<object1>" typeMapping="bitNormal"/>

</GroupMapping>

<GroupMapping appType=*"<group type name>"*>

<Object name="<object1>" typeMapping="bitNormal"/>

…

**Error EPI0901**: InterfaceTypeMappings=`<interface>` mapping=`GroupMapping` name=`` appType= `<group type name>` - duplicated appType for mapping without name not allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type name>">

<Object name="<object1>" type="boolean"/>

</GroupType>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping appType=*"<group type na>"*>

<Object name="<object1>" typeMapping="bitNormal"/>

…

**Error EPI0902**: InterfaceTypeMappings=`<interface>` `GroupMapping` name=`` - appType *`<group type na>`* not found. File=`<name>` Line 18 Position 6

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type name>" length="1"/>

…

<InterfaceTypeMappings name="<interface type mappings name>" …>

<StringMapping name="<string mapping name>" appType="<string type name>" isDefault="true" …/>

<StringMapping name="<string mapping name1>" appType="<string type name>" isDefault=*"true"* …/>

…

**Error EPI0903**: InterfaceTypeMappings=<interface type mappings name> StringMapping name=`<string mapping name1>` appType=<string type name> isDefault=true - default typeMapping for this appType already exists (previous default mapping name=`<string mapping name>`)

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type name>" length="1"/>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="<string mapping name>" appType="<string type name>" …/>

<StringMapping name=*"<string mapping name>"* appType="<string type name>" …/>

…

**Error EPI0904**: InterfaceTypeMappings=`<interface type mappings name>` appType=`<string type name>` StringMapping name=*`<string mapping name>`* - mapping with the same name already exists

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri>" revision=*"1.0.0.0"* …>

…

<InterfaceTypeMappings name="<interface>" revision=*"1.0.0.0999"*>

…

**Error EPI0905**: InterfaceTypeMappings name=`<interface>` - Interface revision in <InterfaceTypeMappings> differs from revision of interface it refers to: expected *`1.0.0.0`,* found *`1.0.0.999`*. File=`<file name>` Line 15 Position 47

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Interface uri="<uri>" …>

…

<InterfaceProfile …>

<Interfaces>

<Interface uri="<uri>" name="<interface>" …/>

…

<InterfaceProfileImplementation …>

<InterfaceTypeMappings name=*"<interface1>"* …>

…

**Error EPI0906**: Interface=*`<interface1>`* for InterfaceTypeMappings not found in profile

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<XxxType name="<type name>" …/>

…

<InterfaceTypeMappings name="<interface>" …>

<XxxMapping name="<mapping name>" />

…

**Error EPI0908**: InterfaceTypeMappings=`<interface>` `XxxMapping` name=`<mapping name>` - one of two appType and object attributes should be present

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Array name="<array1>" elementType="int" size="2"/>

…

<InterfaceTypeMappings name="<interface>" …>

<ArrayMapping object="*<array2>*" />

…

**Error EPI0910**: InterfaceTypeMappings=`<interface>` `ArrayMapping` name=`` object=*`array2`* - object not found in profile

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<XxxMapping object="<object>" …/>

<XxxMapping *object="<object>"* …/>

…

**Error EPI0911**: InterfaceTypeMappings=`<interface>` `XxxMapping` name=`` object=`<*object*>` - duplicated typeMapping definition for the same object not allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<ArrayType name="<dynamic array type>" elementType="int"/>

…

<Objects>

<Array name="<dynamic type array>" type="<dynamic array type>"/>

…

**Error EPI0914**: Interface name=`<interface>` Array name=`<dynamic type array>` size=0 - dynamic array objects not supported

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<ArrayType name="<dynamic array type>" elementType="int"/>

…

<Objects>

<Array name="<array of dynamic arrays>" elementType="<dynamic array type>" size="2"/>

…

**Error EPI0915**: Interface name=`<interface>` Array name=`<array of dynamic arrays>` size=`2` elementType= `<dynamic array type>` category= `Array` element size=0 – array of dynamic array objects not supported

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<XxxMapping object="<object>" *name="<name>"*…/>

…

**Error EPI0916**: InterfaceTypeMappings=`<interface>` `XxxMapping` object=`<object>` *name=`<name>`* - name for object not allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Group name="group">

<Object name="name">

<StringType length="2"/>

…

<ObjectMappings>

<Segment name="<segm>" …>

<AllocateRange name=*"<*interface*.group>"* …/>

…

**Error EPI1019**: Segment name=`<segm>` Context path=`` `AllocateRange` name `*<*interface*.group>*` appType category=`*Group*` - category should be array

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<ArrayType name="<range array type>" elementType="int" size="6"/>

…

<Objects>

<Array name="<range array>" type="<range array type>"/>

…

<ObjectMappings>

<Segment name="<segm>" …>

<Context path="<interface>">

<AllocateRange name=*"*<range array>*"* elementTypeMapping="int" index="*100*" count="2"/>

…

**Error EPI1020**: Segment name=`<segm>` Context path=`<interface>` `AllocateRange` name= `<range array>` index=`*100*` count=`2` - `*index*` value out of array

Or

<Types>

<ArrayType name="<range array type>" elementType="int" size="6"/>

…

<Objects>

<Array name="<range array>" type="<range array type>"/>

…

<ObjectMappings>

<Segment name="<segm>" …>

<AllocateRange name=*"*<interface>.<range array>*"* elementTypeMapping="int" index="5" count="*100*"/>

…

**Error EPI1020**: Segment name=`<segm>` Context path=`` `AllocateRange` name= `<interface>.<range array>` index=`5` count=`*100*` - `*count*` value out of array

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Group name="group1">

<Group name="group2">

<Group name="group3">

<Object name="name">

<StringType length="12"/>

…

<ObjectMappings>

<Segment name="<segment name>" …>

<Allocate name="interface.group1.*groupZ*.group3.name" …/>

…

**Error EPI1021**: Segment name=`<segment name>` - `Allocate` `name` `interface.group1.*groupZ*.group3.name` not found in profile (Bad part name=`*groupZ*`)

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object>" type="<type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="<mapping>" appType="<type>" …/>

…

<ObjectMappings>

<Segment name="<name>" …>

<Context path="<interface>">

<Allocate name=*"<absentObject>"* typeMapping="<mapping>"/>

…

**Error EPI1022**: Segment name=`<name>` Context path=`<interface>` - `Allocate` name `*<absentObject>*` not found in interface (bad part name=`<*absentObject>*`)

\_\_\_\_\_\_\_\_\_\_\_\_\_

<ObjectMappings>

<Segment name="<name>" *address=""* …>

…

**Error EPI1104**: Segment name=`<name>` - invalid *address=``:* Input string was not in a correct format.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<ObjectMappings>

<Segment name="<name>" address=*"0xx250000"*…>

…

**Error EPI1105**: Segment name=`name` - invalid address*=0x`x250000`*: Input string was not in a correct format.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<GT>">

<Object name="<object>" type="int"/>

</GroupType>

…

<Objects>

<Group name="<group>" type="<GT>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<GM>" appType="<GT>" >

<Object name="<object>" typeMapping="*<absentName>*"/>

…

**Error EPI1201**: Mapping `interface.GM` was not built - mapping `interface.*absentName*` for `Object` name= `<object>` type= `int` not found; Error EPI2601: `Object` name= `<object>` - typeMapping= `<*absentName>*` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name=*"<internal group>"* >

…

</Group>

</GroupType>

…

<Objects>

<Group name="<group object>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping object="<group object>">

<Group name=*"<internal gr>"* >

…

**Error EPI1202**: Mapping `<interface.group object>` was not built - group `*internal group*` from GroupType not found in GroupMapping; Error EPI2607: `Group` name `*internal group*` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface name>" …>

<GroupMapping name="<group mapping>" appType="*int*"…/>

…

**Error EPI1503**: Interface name=`<interface name>` GroupMapping name= `<group mapping>` appType= `*int*` category= `Numeric` - appType should be GroupType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<ObjectMappings>

<Segment name="<segment>" …>

<Allocate name="<interface>.<object>" typeMapping="*<absentMapping>*"/>

…

**Error EPI1801**: mappingName=`<interface name>.*<absentMapping>*` not found in the Segment name= `<segment>` Allocate name=`<interface>.<object>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type>" …/>

…

<InterfaceTypeMappings name="<interface name>" …>

<NumericMapping name="<mapping>" appType="*<string type>*" …>

…

**Error EPI2201**: InterfaceTypeMappings=`<interface>` NumericMapping name= `<mapping>` - appType `<*string type*>` should be NumericType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range min="0" max="1" step="1"/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" *size="7"* />

…

**Error EPI2203**: InterfaceTypeMappings=`<interface>` NumericMapping name= `<numeric mapping>` appType= `<numeric type>` bitMapping=`False*` size=`7`* - invalid attribute. Only 1, 2, 3 values are allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range min="0" max="1" step="1"/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" size="2" isSigned="false" *byteOrder*="normal" bitMapping="true"/>

…

**Error EPI2205**: InterfaceTypeMappings=`<interface>` NumericMapping name= `<numeric mapping>` appType= `<numeric type>` bitMapping= `true` - *byteOrder* attribute (=`normal`) is not allowed for bitMapping

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="string mapping object">

<StringType length="1"/>

</Object>

<Object name="numeric mapping object">

<NumericType>

<Range min="0.1" max="1440.1" step="0.1"/>

</NumericType>

</Object>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping object="*<string mapping object>*" size="2" isSigned="false">

…

**Error EPI2208**: InterfaceTypeMappings=`<interface>` NumericMapping object= `*<string mapping object>*` appType category= `String` - type of object should be NumericType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<special value type>">

<Range …/>

<SpecialValue name="<one>" …/>

<SpecialValue name="<two>" …/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<special value type>" …>

<Scaling …/>

<SpecialValue name="<one>" *mapsTo="9"* />

<SpecialValue name="<two>" *mapsTo="9"* />

…

**Error EPI2211**: InterfaceTypeMappings name=`<interface>` `appType`= `<special value type>` NumericMapping name= `*<*numeric mapping>` SpecialValue name= `<two>` *mapsTo= `9`* - duplicated *mapsTo* value

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<special value type>">

<Range min="10.1" max="1440.1" step="0.1"/>

<SpecialValue name="<one>" value="2.1"/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<special value type>" byteOrder="normal" size="1" isSigned="false" isDefault="true">

<Scaling multiplier="1.1" adder="2.1" />

<SpecialValue name="<one>" mapsTo="9" />

<SpecialValue *name="<two>"* mapsTo="7" />

…

**Error EPI2212**: InterfaceTypeMappings name =`<interface>` `appType`= `<special value type>` NumericMapping name= `*<*numeric mapping>` - SpecialValue *name= `<two>`* does not belong to special values of `NumericType`

Or

<Types>

<NumericType name="<special value type">

<Range min="10.1" max="1440.1" step="0.1"/>

<SpecialValue name="<one>" value="2.1"/>

<SpecialValue name="<two>" value="1500"/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<special value type>" …>

<Scaling multiplier="1.1" adder="2.1" />

<SpecialValue name="<one>" mapsTo="9" />

…

**Error EPI2212**: InterfaceTypeMappings name =`<interface>` `appType`= `<special value type>` NumericMapping name= `*<*numeric mapping>` - SpecialValue name= `<two>` does not belong to special values of `NumericMapping`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<special value type>">

<Range …/>

<SpecialValue name="<one>" …/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<special value type>" *size="1"* …>

<Scaling …/>

<SpecialValue name="<one>" *mapsTo="1000"* />

…

**Error EPI2213**: InterfaceTypeMappings name=`<interface>` NumericMapping name= `*<*numeric mapping>` `appType`= `<special value type>` SpecialValue name= `<one>` *mapsTo= `1000`* - *mapsTo value* does not fit into NumericMapping size

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range min="0" max="2" step="1"/>

<SpecialValue name="<one>" value="3"/>

<SpecialValue name="<two>" value="4"/>

</NumericType>

…

<Objects>

<Object name="object" type="<numeric type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" byteOrder="normal" size="2" isSigned="false">

<Scaling multiplier="1.0" adder="0.0"/>

<SpecialValue name="<one>" *mapsTo="1"*/>

<SpecialValue name="<two>" mapsTo="4"/>

</NumericMapping>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<object>" typeMapping="<numeric mapping>"/>

…

**Error EPI2214**: InterfaceTypeMappings name=`<interface>` `appType`= `<numeric type>` NumericMapping name= `<numeric mapping>` SpecialValue name= `<one>` value=`3` - mapsTo=*`1`* inside of resulting (after back-scaling) range = `0` -:- `2` is not allowed

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range min="0" max="1" step="1"/>

</NumericType>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" size="2" isSigned="false" byteOrder="normal" bitMapping="true" *float=`true*`/>

…

**Error EPI2215:** InterfaceTypeMappings=`<interface>` NumericMapping name= `<numeric mapping>` appType= `<numeric type>` - *float=`true`* attribute cannot be used simultaneously with the following attributes: bitMapping=`true`; isSigned=`false`; size equal to `1`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>" …/>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<mapping>" appType="*<string type>*" …>

…

**Error EPI2301**: InterfaceTypeMappings name=`<interface>` NumericMapping name=`<mapping>` - appType `<*string type*>` should be NumericType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<numeric mapping object>">

<NumericType>

…

</NumericType>

</Object>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="<mapping>" object=*"<numeric mapping object>"* …/>

…

**Error EPI2302**: InterfaceTypeMappings name=`<interface>` `StringMapping` object= *`<numeric mapping object>`* appType category= `*Numeric*` - appType should be StringType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<EnumerationMapping name="<enumeration mapping>" appType="*int*" …>

…

**Error EPI2401**: InterfaceTypeMappings name=`<interface>` `EnumerationMapping` name= `<enumeration mapping>` `appType`= `*<numeric type>*` appType category= `Numeric` - appType should be EnumerationType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>" >

<Range min="0.1" max="1440.1" step="0.1"/>

</NumericType>

<EnumerationType name="<enumeration type>">

<Item value=*"Base\_Memory"*/>

<Item value="Control"/>

</EnumerationType>

…

<Objects>

<Object name="<enumeration object>" type="<enumeration type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" size="2" isSigned="false">

<Scaling multiplier="1.1" adder="2.1"/>

</NumericMapping>

<EnumerationMapping name="<enumeration mapping>" appType="<enumeration type>" targetTypeMapping="<numeric mapping>">

<Item value=*"Base\_Mem"* mapsTo="0.1"/>

<Item value="Control" mapsTo="1.2"/>

…

**Error EPI2403**: InterfaceTypeMappings name=`<interface>` Mapping name= `<enumeration mapping>` `appType`= `<enumeration type>` appType category= `Enumeration` - Item value= *'Base\_Memory'* is missing or is not unique

Or

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

<Object name="<enumeration object>">

<EnumerationType>

<Item value="Base\_Memory"/>

<Item value="Control"/>

</EnumerationType>

</Object>

</Group>

</GroupType>

…

<Objects>

<Group name="<group with enumeration>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group mapping>" appType="<group type>">

<Group name="<group>">

<Object name="<enumeration object>">

<EnumerationMapping targetTypeMapping="int">

<Item value="Base\_Memory" mapsTo="0.1"/>

*<!--<Item value="Control" mapsTo="1.2"/>-->*

</EnumerationMapping>

</Object>

</Group>

</GroupMapping>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="group with enumeration"/><!-- or with typeMapping="group mapping"/>-->

…

**Error EPI2403**: InterfaceTypeMappings name=`<interface>` Mapping name=`` `appType`=`` - appType category= `Enumeration` Item value= *'Control'* is missing or is not unique in mapping

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<EnumerationMapping name="<enumeration mapping>" appType="<enumeration type>" targetTypeMapping=*"<string mapping>"*>

…

**Error EPI2404**: InterfaceTypeMappings name=`<interface>` EnumerationMapping name= `<enumeration mapping>` `appType`= `<enumeration type>` - appType category= `String` of targetTypeMapping= `*<string mapping>*` should be `Numeric`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<EnumerationMapping name="<enumeration mapping>" appType="<enumeration type>" targetTypeMapping="<numeric mapping>">

<Item value="Base\_Memory" *mapsTo="0xsss"*/>

…

**Error EPI2405**: InterfaceTypeMappings name=`<interface>` EnumerationMapping name= `<enumeration mapping>` `appType`= `<enumeration type>` Item value= `Base\_Memory` *mapsTo= `0xsss`* - incorrect mapsTo attribute: Input string was not in a correct format.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<EnumerationType name="<enumeration type>">

<Item value="Control"/>

</EnumerationType>

…

<Objects>

<Object name="<enumeration object>" type="<enumeration type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<EnumerationMapping name="<enumeration mapping>" appType="<enumeration type>" …>

<Item value="Control" mapsTo="1"/>

<Item value="*AbsentItem*" mapsTo="2"/>

<EnumerationMapping>

…

**Error EPI2406**: InterfaceTypeMappings name=`<interface>` Mapping name= `<enumeration mapping>` `appType`= `<enumeration type>` appType category= `Enumeration` - Item value= *'*AbsentItem*'* is missing in the appType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<boolean mapping object>" type="*int*"/>

…

<InterfaceTypeMappings name="<interface>" …>

<BooleanMapping object="<boolean mapping object>">

…

**Error EPI2501**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name=`` object= `<boolean mapping object>` - appType category `*Numeric*` should be Boolean

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceTypeMappings name="<interface>" …>

<BooleanMapping object="<boolean mapping object>">

<fixed numericMapping=*"<absent numeric mapping>"*>

…

**Error EPI2503**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name=`` object= `<boolean mapping object>` - `Fixed` NumericMapping `*<absent numeric mapping>*` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<boolean mapping object>" type="boolean"/>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="<string mapping>" appType="<string type>" …/>

<BooleanMapping object="<boolean mapping object>">

<fixed numericMapping="*<string mapping>*">

…

**Error EPI2504**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name=`` object= `<boolean mapping object>` - `Fixed` value of numericMapping `<interface>.<*string mapping*>` should be numericMapping

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>" >

<Range min="0" max="254" step="1"/>

</NumericType>

…

<Objects>

<Object name="<boolean mapping object>" type="boolean"/>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" size="1" isSigned="false">

<Scaling multiplier="1" adder="0"/>

</NumericMapping>

<BooleanMapping object="<boolean mapping object>">

<fixed numericMapping="<numeric mapping>">

*<true>255</true>*

</fixed>

</BooleanMapping>

…

**Error EPI2505**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name=`` object= `<boolean mapping object>` BooleanMappingCategory= `Fixed` numericMapping= `<numeric mapping>` - `<true>`= `*255*` is not valid value of NumericType `<numeric type>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range *min="1" max="2"* step="1"/>

…

<InterfaceTypeMappings name="<interface>" …>

<BooleanMapping name="<boolean mapping>">

<cstyle numericMapping="<numeric mapping>"/>

…

**Error EPI2506**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name= `<boolean mapping>` object=`` BooleanMappingCategory= `CStyle` numericMapping= `<numeric mapping>` - false value=0 is not valid value of NumericType `<numeric type>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numeric type>">

<Range *min="0" max="0"* step="1"/>

…

<InterfaceTypeMappings name="<interface>" …>

<NumericMapping name="<numeric mapping>" appType="<numeric type>" …>

<BooleanMapping name="<boolean mapping>">

<cstyle numericMapping="<numeric mapping>"/>

…

**Error EPI2507**: InterfaceTypeMappings name=`<interface>` `BooleanMapping` name= `<boolean mapping>` object=`` BooleanMappingCategory= `CStyle` numericMapping= `<numeric mapping>` - NumericType `<numeric type>` should contain non-zero values

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<GT>">

<Object name="<object>" type="int"/>

</GroupType>

…

<Objects>

<Group name="<group>" type="<GT>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<GM>" appType="<GT>" >

<Object name="<object>" typeMapping="*<absentName>*"/>

…

Error EPI1201: Mapping `interface.GM` was not built - mapping ` interface.*absentName*` for `Object` name= `<object>` type= `int` not found; **Error EPI2601**: `Object` name= `<object>` - typeMapping= `<*absentName>*` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<ArrayType name="<array type>" elementType="int" size="3"/>

<GroupType name="<group type>">

<Group name="<group>">

<Array name="<array>" elementType="int" size="2"/>

<Array name="<array with type>" type="<array type>"/>

</Group>

</GroupType>

…

<Objects>

<Group name="<group object>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<ArrayMapping name="<array type mapping>" appType="<array type>" elementTypeMapping="int"/>

<GroupMapping name="<group type mapping>" appType="<group type>" >

<Group name="<group>">

<Array name="<array with type>" typeMapping="<array type mapping>" elementTypeMapping="int"/>

<Array name="<array>" elementTypeMapping="int" />

</Group>

</GroupMapping>

…

**Error EPI2602**: InterfaceTypeMappings name=`<interface>` `Array` name= `<array with type>` `Array` typeMapping= `<array type mapping>` elementTypeMapping= `int` - cannot specify typeMapping and elementTypeMapping simultaneously

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Object name="<object>" type="int"/>

…

<Objects>

<Group name="<group object>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group type mapping>" appType="<group type>" >

<Object name="<object>" typeMapping="*<string mapping>*"/>

…

**Error EPI2603**: InterfaceTypeMappings name=`<interface>` `Object` name= `<object>` type= `int` typeMapping= `*<string mapping>*` appType= `<string type>` - typeMapping should be a mapping of the corresponding object type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Array name="<array>" type="array type"/>

…

<Objects>

<Group name="<group object>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group type mapping>" appType="<group type>" >

<Array name="<array>" typeMapping="*<string mapping>*"/>

…

**Error EPI2605**: InterfaceTypeMappings name=`<interface>` `Array` name= `<array>` - typeMapping `*<string mapping>*` should be a mapping of the corresponding object type

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<ArrayType name="<array type>" elementType="<numeric type>" size="3"/>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="string mapping" appType="<string type>" encoding="Western"/>

<ArrayMapping name="<array mapping>" appType="<array type>" elementTypeMapping*="<string mapping>"* />

…

**Error EPI2705**: InterfaceTypeMappings name=`<interface>` `ArrayMapping` name=`<array mapping>` appType=`` Category= `Array` elementType= `<numeric type>` elementTypeMapping= *`<string mapping>`* - elementTypeMapping should be a mapping of the corresponding elementType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>">

<Array name="<array>" elementType="<int>" size="3"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group type mapping>" appType="<group type>" >

<Group name="<group>">

<Array name="<array>" elementTypeMapping="*<byte>*" />

…

**Error EPI2706**: InterfaceTypeMappings name=`<interface>` `Array` name=`<array>` elementType= `<int>` - elementTypeMapping= `*<byte>*` - elementTypeMapping should be a mapping of the corresponding elementType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

<Object name="<object>">

<StringType length="12"/>

…

<Objects>

<Group name="<nested group>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping object="<nested group>">

<Group name="<group>" *typeMapping="…"*>

<Object name="<object>" >

<StringMapping encoding="Western"/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<nested group>"/>

…

**Error EPI2804**: InterfaceTypeMappings name=`<interface>` `GroupMapping` name=`` appType=`` object= `<nestedGroup>` `Group` name= `<group>` - cannot specify both *typeMapping attribute* and inline mapping

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Object name="<object0>" type="boolean"/>

<Object name="<object2>" type="boolean"/>

…

<InterfaceTypeMappings name="<interface>"…>

<GroupMapping name="<group mapping>" appType="<group type>" >

<Object name="*<object1>*" typeMapping="bitNormal" … />

<Object name="<object2>" typeMapping="bitNormal" …/>

…

**Error EPI2807**: Interface='<interface>' mapping name=`<group mapping>` - `*<object1>*` element not found in type definition

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<inline group type>">

<Group name="<group>" >

*<Object name="<inline string1>">*

*<StringType length="1" />*

*</Object>*

<Object name="<inline string2>">

<StringType length="2" />

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<inline group mapping>" appType="<inline group type>">

<Group name="<group>" >

<Object name="<inline string2>" >

<StringMapping encoding="Western"/>

</Object>

</Group>

</GroupMapping>

…

**Error EPI2900**: InterfaceTypeMappings name=`<interface>` `Group` name= `<group>` `Object` name= `*<inline string1>*` - [inline] mapping for the `Object` not found

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

<Object name=*"<inline object>"*>

*<NumericType>*

*<Range min="0" max="1" step="1"/>*

…

<Objects>

<Group name="<group>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

…

<ObjectMappings>

<Segment name=`<segm name>`…>

<Context path="<interface>">

<Allocate name="<group>" …/>

…

**Error EPI2901**: Interface name=`<interface>` `GroupMapping/Group` name=`` appType name=`` - could not find mapping for Object name `*<inline object>*` in the Segment name= `<segm name>` Context path= `<interface>` Allocate name=`<group>` address=`<address>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type>" …/>

…

<Objects>

<Object name="<object>" type="<string type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<StringMapping name="<string mapping>" appType="<string type>" …/>

…

<ObjectMappings>

<Segment name="<segment>" …>

*<!--<Allocate name="<object>" …/>-->*

*…*

**Error EPI3101**: AppInterfaceObject full name=`<interface>.<*object*>` - address not assigned

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object>" type=*"timestampms"*/>

…

<InterfaceTypeMappings name="<interface>" …>

<TimeStampMapping name="<mapping>" appType="timestamp"packaging="unix" …/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<object>" typeMapping="<mapping>"/>

…

**Error EPI3403**: object=`<object>` appType=`timestampms` typeMapping `mapping` appType= `timestamp` - typeMapping should be a mapping of the type `timestampms`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<NumericType name="<numericType>">

<Range min="0" max="100" step="1"/>

…

<Objects>

<Object name="<object>" type="<NumericType>" />

…

<InterfaceTypeMappings name="<interface>" …>

*<!--<NumericMapping name="<numeric mapping>" appType="<NumericType>" …>-->*

…

<ObjectMappings>

<Segment name=`<segm>`…>

<Context path="<interface>">

<Allocate name="<object>" />

…

**Error EPI3404**: AppInterfaceImplementationObject name=`interface.object` appType= `interface.numericType` - could not determine typeMapping for `interface.object` in the Segment name=`segm` Allocate name=`interface.object`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="group" >

<Object name="<object>">

<StringType length="12"/>

…

<Objects>

<Group name="group" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<GroupMapping name="<group mapping>" appType="<group type>">

<Group name="<group>">

<Object name="<object>" >

<StringMapping …/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

*<!--*<Allocate name="<group>" typeMapping="<group mapping>"/>*-->*

<Allocate name="<group>" *typeMapping="int"/>*

*…*

**Error EPI3503**: Allocate name=`<interface.group>` typeMapping= `*int*` - appType Category `Numeric` should be a `Group` in the Segment name= `Sgm` Allocate name= `<interface.group>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Array name="<array object>" elementType="<int>" size="2"/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<!--<Allocate name="<array object>" typeMapping="<array mapping> "/>-->

<Allocate name="<array object>" typeMapping=*"int"*/>

…

**Error EPI3701**: Allocate name=`<array object>` typeMapping=*`int`* - appType Category `Numeric` should be `Array` in the Segment name=…

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<GroupType name="<group type>">

<Group name="<group>" >

…

<Objects>

<Group name="<group>" type="<group type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

*<GroupMapping name="<group mapping1>" appType="<group type>">*

<Group name="<group>" >

…

*<GroupMapping name="<group mapping2>" appType="<group type>">*

<Group name="<group>" >

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<group>" />

…

**Error EPI3801**: AppType=`<interface.group type>` number of mappings=`2`, last mapping name= `<interface.group mapping1>` - more than one user-defined mapping for the appType exists. No default mapping specified (isDefault attribute). Allocate name=`Interface9SPT.group` typeMapping=null

\_\_\_\_\_\_\_\_\_\_\_\_\_

<ObjectMappings>

<Segment name="<name>" address="<address1>" …>

<Segment name*="<name>"* address="<address2>" …>

…

**Error EPI4001**: file=`<fileName>`. Syntax error in Line 678 Position 6: There is a duplicate key sequence *'<name>*' for the 'SegmentNameKey' key or unique identity constraint

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<StringType name="<string type>" …/>

<TimeStampType name="<timestamp type>" precision="seconds" …/>

…

<Objects>

<Object name="<object>" type="<timestamp type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<TimeStampMapping name="<mapping>" appType=*"<string type>"* …/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<object>" typeMapping="<mapping>"/>

…

**Error EPI4101**: InterfaceTypeMappings=`<interface>` `TimeStampMapping` name= `<mapping>` `appType`= `<*string type>*` - mapping should be a mapping of the corresponding appType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object>" type="timestampms"/>

…

<InterfaceTypeMappings name="<interface>" …>

<TimeStampMapping name="<mapping>" appType=*"timestampms"* packaging="unix" …/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<object>" typeMapping="<mapping>"/>

…

**Error EPI4102**: InterfaceTypeMappings=`<interface>` `TimeStampMapping` name= `<mapping>` `appType` `*timestampms*` - `unix` packaging is not applicable to timestampms precision

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<TimeStampType name="<type>" precision="seconds" from="01/01/1969T13:00:01" …/>

…

<Objects>

<Object name="<object>" type="<type>"/>

…

<InterfaceTypeMappings name="<interface>" …>

<TimeStampMapping name="<mapping>" appType="<type>"packaging="unix" …/>

…

<ObjectMappings>

<Segment …>

<Context path="<interface>">

<Allocate name="<object>" typeMapping="<mapping>"/>

…

**Error EPI4103**: InterfaceTypeMappings=`<interface>` `TimeStampMapping` name=`<mapping>` `appType`= `<type>` - from `01/01/1969 13:00:01` could not be earlier than `01/01/1970 00:00:00` for packaging `unix`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Types>

<VersionMapping name="<version mapping>" appType=*"int"* …/>

…

**Error EPI5301**: InterfaceTypeMappings=`<interface>` `VersionMapping` name= `<version mapping>` `appType`= *`int`* - appType category= *`Numeric`* - mapping should be a mapping of the corresponding appType"

\_\_\_\_\_\_\_\_\_\_\_\_\_

<Objects>

<Object name="<object>" type=*"int"*/>

…

<InterfaceTypeMappings name="<interface>" …>

<IPAddressMapping object="<object>" …/>

…

**Error EPI5302**: InterfaceTypeMappings=`<interface>` `IPAddressMapping` name=`` `object`= `<object>` appType category= `*Numeric*` - mapping should be a mapping of the corresponding appType

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfileImplementation name="<name>" *revision="" …*>

…

**Error EPI5801**: FileName=`<file name>` InterfaceProfileImplementation name=`<name>` *revision=``* - invalid revision value: Version string portion was too short or too long.

\_\_\_\_\_\_\_\_\_\_\_\_\_

## Setpoint definition parser errors

<InterfaceProfile name="<interface profile name>" …>

<SaveRestoreGroups file="path\*SRGroupsDef9.xsptde*"/>

…

SetpointsDefinitionParser sptParser = new SetpointsDefinitionParser();

sptParser.Load(prof, profile.SptFile);

…

**Error ESP0001**: Could not open file for reading: Could not find a part of the path 'path\*SRGroupsDef9.xsptde*'

\_\_\_\_\_\_\_\_\_\_\_\_\_

<SaveRestoreGroups name="<SGR>" …>

<SRGroup name=*"name admin = "*true"*>*

…

**Error ESP0002**: file=`<file name>`. SaveRestoreGroups file is incorrect - 'true' is an unexpected token. Expecting white space. Line 2, position 36.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<SaveRestoreGroups name="<SGR>" …>

<SRGroup name="<name>">

*qwert*

<SRGroup name="<name1>">

…

**Error ESP0201**: file `<file name>` is incorrect. Line 7 Position 30: The element 'SRGroup' cannot contain text. List of possible elements expected: 'SRGroup, Item'.

Or

<SaveRestoreGroups *name=""* …>

<SRGroup name="<name>">

…

**Error ESP0201**: file `<file name>` is incorrect. Line 7 position 14: The 'name' attribute is invalid - The value '' is invalid according to its datatype 'Ident' - The Pattern constraint failed.

Or

<SaveRestoreGroups name="<saveRestore group>" *formatRev=*"*1.2.3.4.5*" …>

…

**Error ESP0201**: file `<file name>` is incorrect. Line 3 position 20: The 'formatRev' attribute is invalid - The value '1.2.3.4.5' is invalid according to its datatype 'FormatRev' - The Enumeration constraint failed.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interfaceProfile>" …>

<Interfaces>

<Interface name="<interface>" …/>

…

<SaveRestoreGroups file="xxx.xsptdef" />

</InterfaceProfile>

…

<SaveRestoreGroups name="<SRG>" …>

<SRGroup name="<name>">

<Item name="*absentInterface.*Group.Object"/>

…

**Error ESP0303**: SRGroup name=`<name>` Item name=`*absentInterface.*Group.Object` - item name is incorrect: Interface `*absentInterface*` not found in profile. SaveRestoreGroups name=`<SRG>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<InterfaceProfile name="<interfaceProfile>" …>

<Interfaces>

<Interface name="<interface>" …/>

…

<SaveRestoreGroups file="xxx.xsptdef" />

</InterfaceProfile>

…

<SaveRestoreGroups name="<SRG>" …>

<SRGroup name="<name>">

<Item name="interface*.absentGroup*.Object"/>

…

**Error ESP0304**: SRGroup name=`<name>` Interface=`<interface>` Item name=`interface*.absentGroup*.Object` - subObject `*absentGroup*` not found in Interface. SaveRestoreGroups name=`<SRG>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<SaveRestoreGroups name="<SRG>" …>

<SRGroup name="<name>" admin="false">

…

**Error ESP0308**: SRGroup name=`<name>` - admin attribute value `false` is incorrect: only `true` value is allowed. SaveRestoreGroups name=`<SRG>`

\_\_\_\_\_\_\_\_\_\_\_\_\_

<SaveRestoreGroups name="<SRG>" revision=*”00123456789012345.0”* …>

…

**Error ESP0501**: SaveRestoreGroups name=`<SRG>` `revision`=*`00123456789012345.0`* - invalid revision value: Value was either too large or too small for an Int32.

\_\_\_\_\_\_\_\_\_\_\_\_\_

<SaveRestoreGroups name="<SRG>" …>

<SRGroup name="<name>" …>

<Item name="interface.Group1">

<Item name="interface.Group1">

*<Item name="interface.Group1.Group2">*

…

**Warning WSP0301**: SRGroup name=`<name>` - Item name `Interface. Group1` already exists;

**Warning WSP0303:** SRGroup name=`<name>` - Item `Interface.Group1` already contains Item *`Interface.Group1.Group2`*

\_\_\_\_\_\_\_\_\_\_\_\_\_