

BiTXml

M2M Communications Protocol

Rel. 2.0.1

Last revision: 2007-11-06

Index

FOREWORDS	4
FOREWORDS TO THE SECOND EDITION	4
CHANGES.....	5
INTRODUCTION.....	6
1. REFERENCE MODEL.....	7
2. BITXML V2 PROTOCOL DEFINITION: SYNTAX	10
2.1 ERROR VALUE	11
2.1.1 Description.....	11
2.1.2 Syntax.....	11
2.1.3 Semantic.....	13
2.1.4 Examples.....	15
2.2 SYSTEM STATE VALUE	16
2.2.1 Description.....	16
2.2.2 Syntax.....	16
2.2.3 Semantic.....	20
2.2.4 Examples.....	20
2.3 HISTORY VALUE.....	21
2.3.1 Description.....	21
2.3.2 Syntax.....	21
2.3.3 Semantic.....	21
2.3.4 Examples.....	21
2.4 INITIALIZATION VALUE.....	23
2.4.1 Description.....	23
2.4.2 Syntax.....	23
2.4.3 Semantic.....	46
2.4.4 Examples.....	46
2.5 GATEWAY REINITIALIZATION COMMAND.....	51
2.5.1 Description.....	51
2.5.2 Syntax.....	51
2.5.3 Return values.....	52
2.5.4 Semantic.....	52
2.5.5 Examples.....	52
2.6 STATE RETRIEVAL COMMAND.....	53
2.6.1 Description.....	53
2.6.2 Syntax.....	53
2.6.3 Return value	55
2.6.4 Semantic.....	55
2.6.5 Examples.....	55
2.7 HISTORY RETRIEVAL COMMAND.....	56
2.7.1 Description.....	56
2.7.2 Syntax.....	56
2.7.3 Return value	56
2.7.4 Semantic.....	56
2.7.5 Examples.....	56
2.8 STATE SETUP COMMAND.....	57
2.8.1 Description.....	57
2.8.2 Syntax.....	57
2.8.3 Return value	68
2.8.4 Semantic.....	68
2.8.5 Examples.....	69
2.9 EVENT VALUE.....	70
2.9.1 Description.....	70
2.9.2 Syntax.....	70

2.9.3	<i>Return values</i>	72
2.9.4	<i>Semantic</i>	72
2.9.5	<i>Examples</i>	72
3.	BITXML V2 PROTOCOL DEFINITION: DATA FLOWS	73
3.1	COMPLETE GATEWAY APPLICATION INITIALIZATION	74
3.2	INITIALIZATION ERROR DATA FLOW	75
3.3	CONDITION MONITOR DATA FLOW	76
3.4	HISTORY MONITOR DATA FLOW	77
3.5	COMMANDS SERVER DATA FLOW.....	78
3.6	RECONFIGURATION CHECKER DATA FLOW.....	80
	APPENDIX A – BITXML V2 SCRIPTING LANGUAGE	81
	GENERAL DESCRIPTION.....	81
	SYNTAX	81
	<if> <i>Conditional statement</i>	81
	<while> <i>Iteration statement</i>	82
	<delay> <i>Operator</i>	82
	<exit> <i>Operator</i>	82
	<set> <i>Operator</i>	83
	<get> <i>Operator</i>	83
	<ret> <i>Operator</i>	83
	<port> <i>Operator</i>	84
	EXAMPLES	84
	APPENDIX B – BITXML DNS PROTOCOL	86
	DESCRIPTION	87
	START-UP MESSAGE	88
	KEEP-ALIVE MESSAGE	89
	APPENDIX C - SAMPLE GATEWAY APPLICATION START-UP META CODE	90

Forewords

While other efforts have been carried out to identify a 'killer protocol' in the M2M arena, it seems that presently no one has already reached the goal.

So, one way to approach the problem might be the definition of an extendible protocol, targeting at first the core functionalities commonly required by M2M applications, lying on a very general architectural model: this is in fact the BiTXml protocol design approach.

Our main guidelines have been readability and easy extendibility of the protocol specs, including the support for different I/O ports, network protocols, and core functionalities.

The result, as an evolutionary project, wants to be of help to everyone is facing M2M projects and needs a reference framework to express commands and control processes in an easy yet (hopefully enough !) powerful way.

The authors, Milan (Italy), Jun 2006.

Forewords to the second edition

Based on our most recent experiences across different M2M projects, a whole new set of hints and requirements have been considered and studied to completely revisit the first release of the BITXml protocol: the result is a more accurate and shaped language, expressing a new set of concepts, in a more accurate form of expression versus the previous edition, at the price of a few backward incompatibilities. The evolution of the protocol is still underway, but we feel we have placed important additional milestone references with this new version of BITXml.

The authors, Milan (Italy), Dec 2006.

Changes

Rel. 2.0.1

- Introduced awakenable connected gateway managers
- ReInit command can force the server monitor to disconnect
- New error codes related to the awake configuration

Introduction

The BiTXml communication protocol has been designed to implement an application level of the (OSI-based) communication stack reference, with the main goal to standardize the way commands and control information are exchanged for the specific target of M2M communication demands (i.e. communication with generic devices with or without processing power on board – like sensors, actuators, as well as air conditioning systems, lifts, .. – or a combination of them).

The current protocol specification defines:

1. The abstraction of a BiTXml gateway application
2. The abstraction of a BiTXml controller
3. The syntax and the semantic of a set of values used to exchange data between two communicating parties
4. The syntax and semantic of a set of commands used to drive the devices connected to the BiTXml gateway application
5. The syntax and semantic for events, generated by the BiTXml gateway application following the recognition of specific conditions on the controlled devices

The protocol syntax will be meta-coded in Xml Schema language.

The protocol semantic will be defined in natural language (no formalization will be provided).

1. Reference model

The architectural reference model underlying the BiTXml protocol is shown in Fig. 1.

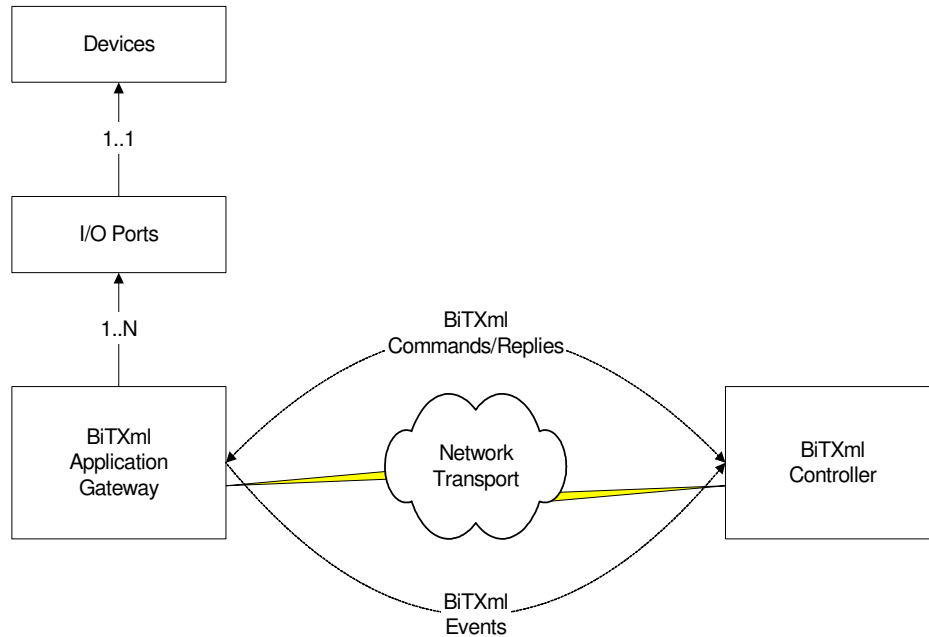


Fig. 1 – Architectural reference model

The main parts of the reference model are:

1. BiTXml gateway application: any kind of software application, speaking the BiTXml language, working as an 'intelligent' remote execution unit
2. BiTXml controller: any kind of software application speaking the BiTXml language, working as the master (controlling) unit for one or more application gateways
3. Network transport: any kind of network transport layer
4. I/O ports: any kind of connection port enabling the control of whatever physical or logical device connected. Actually supported connection ports range through analog and digital GPIOs, positioning devices, serial ports and user-definable (logical) ports.
5. Devices: any kind of logical or physical device connectable to the available I/O ports.

While no detail may be given for the BiTXml controller component, being it the side where applications or middleware solutions as well may take the role of master unit, an architectural model for a standard gateway application may be shown (see Fig. 2):

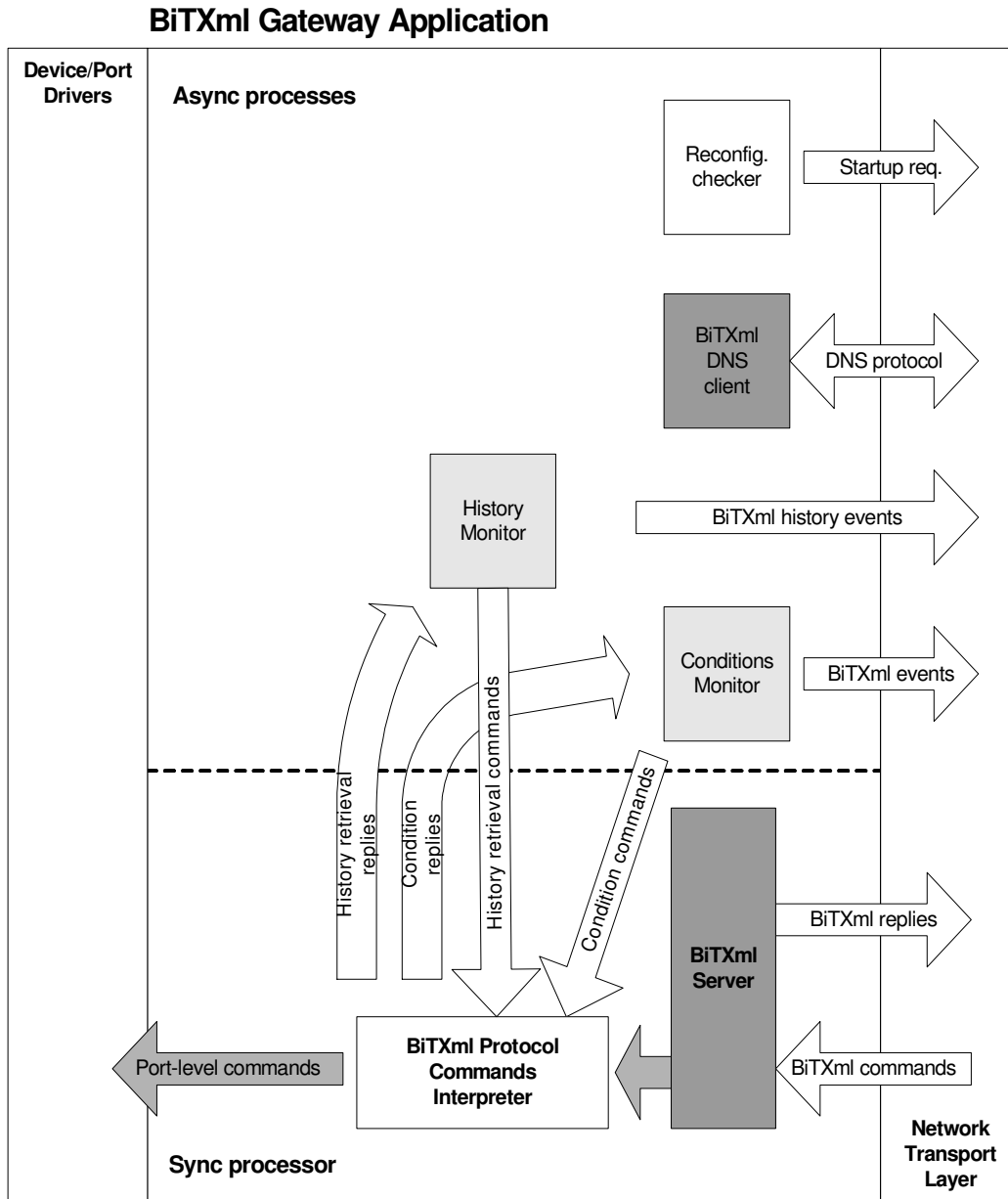


Fig. 2 - Standard gateway application reference model

The application gateway is surrounded by the network layer (right side of the drawing) and the device/ports drivers at the other side.

The gateway may be split into two main logical parts:

- asynchronous processing
- synchronous processing

Asynchronous processing involves several executing units, each one accomplishing a well-defined function within the system.

Currently defined asynchronous units are:

- Conditions monitor: a set of conditions may be installed on the system and polled for matches; if any match is found, an event for each match is generated and sent to the master unit.

- History monitor: a list of port state requests may be installed on the system, executed by the monitor, so that any state reply is stored locally in a non-volatile fixed-size FIFO area. The history is sent to the master unit in a programmable fashion.
- Reconfiguration monitor: a schedule table may be installed on the system, allowing the monitor to periodically check for reconfiguration needs.

Synchronous processing allows the basic execution of BiTXml commands, returning the replies for each executed command.

Currently defined synchronous units are:

- BiTXml Server: network client, connected with the controlling unit, able to receive BiTXml commands, execute them using the commands interpreter, and return the corresponding results.
- BiTXml DNS: a custom protocol client, specifically designed to allow the handling of gateways having their network address dynamically assigned (such as GPRS- or WI-FI- networked devices), used when the gateway is connected (see later). The DNS is responsible to connect to the Controller in order to register the gateway, and to maintain the connection open where network transport may be broken by operator-asserted timeouts.

Gateway applications can be configured in three fundamental ways:

- connected
- detached
- custom

Connected gateway applications must run the protocol server and the DNS client, and may run conditions and history monitors.

Detached gateway applications must not run the protocol server and the DNS client, and must run at least one from the condition and history monitors.

Custom applications should use the configuration at their wish, although respecting the message flow defined by the protocol.

The reconfiguration monitor may be run by whatever type of gateway or application.

2. BITXML V2 Protocol Definition: Syntax

2.1 Error Value

2.1.1 Description

The **<error>** value is used to code any error that happens in the system, both following the execution of a command, and following an exceptional (self-recognized) condition; it may be returned as an event to the controlling unit (see the Event value description) when an asynchronous monitors must communicate an error.

2.1.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Error"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.bitxml.org/v2/Error.xsd"
  xmlns:self="http://www.bitxml.org/v2/Error.xsd"
>
  <xs:element name="error" type="self:ErrorNodeType" />

  <xs:complexType name="ErrorNodeType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="id-device" type="xs:string" use="required" />
        <xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
        <xs:attribute name="code" type="self:CodeType" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="CodeType">
    <xs:restriction base="xs:nonNegativeInteger">

      <!-- ERR_NO -->
      <xs:enumeration value="0" />

      <!-- ERR_GENERIC -->
      <xs:enumeration value="1" />
      <!-- ERR_SYNTAX -->
      <xs:enumeration value="2" />
      <!-- ERR_MEMORY -->
      <xs:enumeration value="3" />

      <!-- ERR_PORTINIT -->
      <xs:enumeration value="100" />
      <!-- ERR_DISABLEDGW -->
      <xs:enumeration value="110" />
      <!-- ERR_NOGWCFG -->
      <xs:enumeration value="120" />
      <!-- ERR_NEWGWCFG -->
      <xs:enumeration value="130" />
      <!-- ERR_NOBRIDGEAVAIL -->
      <xs:enumeration value="140" />

      <!-- ERR_PORTINIT -->
      <xs:enumeration value="500" />
      <!-- ERR_STARTCONDMONITOR -->
```

```
<xs:enumeration value="501" />
<!-- ERR_STARTHISTMONITOR -->
<xs:enumeration value="502" />
<!-- ERR_STARTRCFGMONITOR -->
<xs:enumeration value="503" />
<!-- ERR_STARTSERVER -->
<xs:enumeration value="504" />
<!-- ERR_STARTDNSCLIENT -->
<xs:enumeration value="505" />

<!-- ERR_SYNGETSTATE -->
<xs:enumeration value="510" />
<!-- ERR_SYNSETSTATE -->
<xs:enumeration value="511" />
<!-- ERR_SYNGETHISTORY -->
<xs:enumeration value="512" />
<!-- ERR_SYNREINIT -->
<xs:enumeration value="513" />

<!-- ERR_INVPORTID -->
<xs:enumeration value="1000" />
<!-- ERR_INVPORTSYSID -->
<xs:enumeration value="1001" />
<!-- ERR_INVPORTTYPE -->
<xs:enumeration value="1002" />
<!-- ERR_INVPORTVAL -->
<xs:enumeration value="1003" />
<!-- ERR_DUPPORTID -->
<xs:enumeration value="1004" />
<!-- ERR_INVPORTDIR -->
<xs:enumeration value="1005" />
<!-- ERR_INVSCRIPT -->
<xs:enumeration value="1006" />
<!-- ERR_DUPPROPID -->
<xs:enumeration value="1007" />
<!-- ERR_INVTIMEVAL -->
<xs:enumeration value="1008" />
<!-- ERR_INVCMD -->
<xs:enumeration value="1009" />
<!-- ERR_INVBINVAL -->
<xs:enumeration value="1010" />
<!-- ERR_INVVAL -->
<xs:enumeration value="1011" />
<!-- ERR_INVDATETIME -->
<xs:enumeration value="1012" />
<!-- ERR_INVPROPID -->
<xs:enumeration value="1013" />

<!-- ERR_DUPCONDID -->
<xs:enumeration value="2000" />
<!-- ERR_INVRANGEMINVAL -->
<xs:enumeration value="2001" />
<!-- ERR_INVRANGEMAXVAL -->
<xs:enumeration value="2002" />
<!-- ERR_INVCMPVAL -->
<xs:enumeration value="2003" />
<!-- ERR_EXECCONDITION -->
<xs:enumeration value="2004" />
<!-- ERR_EXECSCRIPT -->
<xs:enumeration value="2005" />
<!-- ERR_EXECOP -->
<xs:enumeration value="2006" />
```

```

<!-- ERR_NOAWAKER -->
<xs:enumeration value="2010" />
<!-- ERR_INVAWAKER -->
<xs:enumeration value="2011" />

<!-- ERR_COMMUNICATION -->
<xs:enumeration value="3000" />

<!-- ERR_UNEXPECTED -->
<xs:enumeration value="5000" />

<!-- ERR_UNSUPPORTED -->
<xs:enumeration value="6000" />

</xs:restriction>
</xs:simpleType>

</xs:schema>

```

The `error@id-device` attribute contains the (originating) gateway application logical name. The `error@datetime-utc` attribute contains the UTC date and time when the error has occurred.

The `error@code` attribute contains the numeric error code.

The inner text of the `<error>` element may contain error details.

2.1.3 Semantic

The `error@code` attribute must assume one of the following values:

Value	Detail message	Description
0	-	No error - unused
1	Custom message describing the error	Generic system error
2	Custom message describing the error	Syntax error
3	-	Unavailable memory
100	Gateway identifier	Invalid <code>id-device</code> value - device not registered. Error issued by the controller to the gateway application when the logical name sent to the controller (with a BOOT system event) is not registered
110	-	Gateway is registered but not operable. Error issued by the controller to the gateway application when the logical name sent to the controller (with a BOOT system event) is registered but the gateway has been taken out of operation (disabled)
120	-	No configuration available. Error issued by the controller to the gateway application when no new configuration is available (requested with a

		REINIT system event)
130	-	Configuration available. Error issued by the controller to the gateway application when a new configuration is available (requested with a REINIT system event)
140	-	No Bridge available for connected support
500	Port identifier; custom message describing the error	Port initialization error.
501	Custom message describing the error	Error starting condition monitor. Error issued by the gateway application when the condition monitor fails to start correctly
502	Custom message describing the error	Error starting history monitor. Error issued by the gateway application when the history monitor fails to start correctly
503	Custom message describing the error	Error starting reconfiguration checker. Error issued by the gateway application when the reconfiguration checker fails to start correctly
504	Custom message describing the error	Error starting on board BiTXml server Error issued by the gateway application when the command server fails to start correctly
505	Custom message describing the error	Error starting BiTXml DNS client. Error issued by the gateway application when the BiTXml DNS client fails to start correctly
510	Custom message describing the error	Error executing synchronous <getstate> command
511	Custom message describing the error	Error executing synchronous <setstate> command
512	Custom message describing the error	Error executing synchronous <gethistory> command
513	Custom message describing the error	Error executing synchronous <reinit> command
1000	Port identifier	Invalid I/O port identifier
1001	System port identifier	Invalid system port identifier
1002	Port name	Invalid I/O port type
1003	Port name; invalid value	Invalid I/O port value
1004	Duplicated port identifier	Duplicated I/O port identifier
1005	Port name	Invalid I/O port direction
1006	Port name	Invalid I/O port script
1007	Property identifier	Duplicated property identifier
1008	Time value	Invalid time value
1009	-	Invalid command. Error issued by the commands server when a malformed command has been submitted for execution
1010	Invalid value	Invalid binary value. Binary values must be hex binary encoded.

1011	Invalid value	Invalid value
1012	Invalid value	Invalid date and time value
1013	Property identifier	Invalid property identifier
2000	Condition identifier	Duplicated condition identifier
2001	Condition name; port identifier ; wrong value	Wrong minimum value for a range element
2002	Condition name; port identifier ; wrong value	Wrong maximum value for a range element
2003	Wrong value	Invalid compare value
2004	Condition name	Condition evaluation error
2005	Port name	Script execution error
2006	Port name	Operation execution error
2010	-	No awaker defined
2011	Awaker method name	Invalid awaker
3000	-	Undefined gateway position
5000	Custom message describing the error	Unexpected error

2.1.4 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<error xmlns="http://www.bitxml.org/v2/Error.xsd"
  id-device="010101011102034" code="1000"
>DIG-28</error>
```

2.2 System State Value

2.2.1 Description

The **<state>** value codes a partial or complete gateway state, which is composed from the state of some or all of the configured ports.

The state of a port depends on its type:

- for analog readable (either INPUT or INPUTOUTPUT type) ports, the state is the actual digitized value of the port, normalized within the given port range configured by the initialization
- for digital readable (either INPUT or INPUTOUTPUT type) ports, the state is the actual pin value (either true (1) or false (0))
- for positioning ports, the state is the actual geographical position detected by the positioning device
- for serial and user readable (either INPUT or INPUTOUTPUT type) ports, the state may be either the last binary or textual value read from the port, or the current port value. If the port is not an input-only port, the state must be reset to an empty value after each retrieval.

2.2.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="State"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/State.xsd"
  xmlns:self="http://www.bitxml.org/v2/State.xsd"
  >

  <xs:element name="state" type="self:StateNodeType"/>

  <xs:complexType name="StateNodeType">
    <xs:sequence>
      <xs:sequence maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="self:a-io" />
          <xs:element ref="self:d-io" />
          <xs:element ref="self:s-io" />
          <xs:element ref="self:ps-o" />
          <xs:element ref="self:user-io" />
        </xs:choice>
      </xs:sequence>
    </xs:sequence>
    <xs:attribute name="id-device" type="xs:string" use="required" />
    <xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
  </xs:complexType>

  <xs:element name="ps-o" type="self:GeoPosType" />

  <xs:complexType name="GeoPosType">
    <xs:attribute name="id" type="xs:string" use="required" />

    <xs:attribute name="longitude" type="xs:float" use="optional" />
```

```
<xs:attribute name="longitude-side" type="self:LongitudeSide" use="optional"
  default="W"/>

<xs:attribute name="latitude" type="xs:float" use="optional" />
<xs:attribute name="latitude-side" type="self:LatitudeSide" use="optional"
  default="N"/>

<xs:attribute name="altitude" type="xs:float" use="optional" />
<xs:attribute name="acquisition-time-utc" type="xs:dateTime" use="optional" />
<xs:attribute name="speed" type="xs:float" use="optional" />
<xs:attribute name="visible-satellites" type="xs:short" use="optional" />
<xs:attribute name="error" type="xs:string"
  use="optional"/>
</xs:complexType>

<xs:simpleType name="LongitudeSide">
  <xs:restriction base="xs:string">
    <xs:enumeration value="W" />
    <xs:enumeration value="E" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LatitudeSide">
  <xs:restriction base="xs:string">
    <xs:enumeration value="N" />
    <xs:enumeration value="S" />
  </xs:restriction>
</xs:simpleType>

<xs:element name="a-io" type="self:aioType" />

<xs:complexType name="aioType">
  <xs:simpleContent>
    <xs:extension base="self:aioValueType">
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="error" type="xs:string"
        use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="aioValueType">
  <xs:union memberTypes="xs:double">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="ERROR" />
        <xs:enumeration value="" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<xs:element name="d-io" type="self:dioType" />

<xs:complexType name="dioType">
  <xs:simpleContent>
    <xs:extension base="self:dioValueType">
      <xs:attribute name="id" type="xs:string" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```

    <xs:attribute name="error" type="xs:string"
      use="optional"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>

<xs:simpleType name="dioValueType">
  <xs:union memberTypes="xs:boolean">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="ERROR" />
        <xs:enumeration value="" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<xs:element name="s-io" type="self:sioType" />

<xs:complexType name="sioType">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element name="error" type="xs:string" />
    <xs:element name="binary" type="xs:hexBinary" />
    <xs:element name="text" type="xs:string" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:element name="user-io" type="self:userIoType" />

<xs:complexType name="userIoType">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element name="error" type="xs:string" />
    <xs:element name="binary" type="xs:hexBinary" />
    <xs:element name="text" type="xs:string" />
    <xs:element name="xml">
      <xs:complexType>
        <xs:sequence>
          <xs:any processContents="skip" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>
</xs:schema>

```

The `state@id-device` attribute contains the originating gateway logical name.
 The `state@datetime-utc` attribute contains the UTC date and time when the state has been retrieved.

`state/a-io` elements contain each one the value of an analog I/O port
 The `state/a-io@id` attribute contains the unique identifier of the analog port
 The `state/a-io@error` attribute, if defined, contains the error message generated by the value retrieval process.

The inner text of the `state/a-io` element contains the read value for the analog port (or ERROR if the value is not available)

`state/d-io` elements contain each one the value of a digital I/O port

The `state/d-io@id` attribute contains the unique identifier of the digital port

The `state/a-io@error` attribute, if defined, contains the error message generated by the value retrieval process.

The inner text of the `state/d-io` element contains the read value for the digital port (or ERROR if the value is not available)

The `state/ps-o` element contains the geographical position of the gateway, retrieved by a positioning on-board device (such as GPS).

The `state/ps-o@longitude` attribute contains the actual longitude of the gateway, expressed in degrees with format **dddmm.ssss**, where ddd are degrees, mm are degree minutes, and ssss are minutes fraction. This attribute may also contain ERROR when no position can be retrieved.

The `state/ps-o@longitude-side` attribute contains the actual longitude side of the gateway, which may evaluate to W (west) or E (east).

The `state/ps-o@latitude` attribute contains the actual latitude of the gateway, expressed in absolute degrees with format **ddmm.ssss**, where dd are degrees, mm are degree minutes, and ssss are minutes fraction. This attribute may also contain ERROR when no position can be retrieved.

The `state/ps-o@latitude-side` attribute contains the actual latitude side of the gateway, which may evaluate to N (north) or S (south).

The `state/ps-o@altitude` attribute contains the actual altitude of the gateway (expressed in meters above the sea level).

The `state/ps-o@acquisition-time-utc` attribute contains the UTC date and time of the position acquisition.

The `state/ps-o@speed` attribute contains the gateway speed (expressed in knots)

The `state/ps-o@visible-satellites` attribute contains the number of satellites involved in the measurement of the gateway position.

The `state/ps-o@error` attribute, if defined, contains the error message generated by the value retrieval process.

`state/s-io` elements contain each one the value of a serial I/O port (that is, the last set of bytes read from the port)

The `state/s-io@id` attribute contains the unique identifier for the serial port

The `state/s-io/error` element, if defined, contains the error message generated by the value retrieval process.

The `state/s-io/binary` element, if defined, contains the hex binary encoded value retrieved from the port.

The `state/s-io/text` element, if defined, contains the string value retrieved from the port.

`state/user-io` elements contain each one the value of a user I/O port (that is, the last set of bytes read from the port)

The `state/user-io@id` attribute contains the unique identifier for the user port

The `state/user-io/error` element, if defined, contains the error message generated by the value retrieval process.

The `state/user-io/binary` element, if defined, contains the hex binary encoded value retrieved from the port.

The `state/user-io/text` element, if defined, contains the string value retrieved from the port.

The `state/user-io/xml` element, if defined, contains the xml value (as a single root element) retrieved from the port.

2.2.3 Semantic

1. Every port value description element (<ps-o>, <a-io>, <d-io>, <s-io> and <user-io>) must have a unique logical identifier, different from every other name used to identify any other port value (of any type).

2.2.4 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<state
  xmlns="http://www.bitxml.org/v2/State.xsd"
  id-device="010101011102034"
  datetime-utc="2006-09-27T13:42:04Z"
>
  <ps-o acquisition-time-utc="2006-09-27T13:43:22Z"
    longitude="9541.345" longitude-side="E"
    latitude="4743.778" latitude-side="S"
    altitude="123.33" speed="3.4" visible-satellites="5" />
  <a-io id="dss-46">23.4</bitxml:a-io>
  <d-io id="scx-45">ERROR</bitxml:d-io>
  <d-io id="ere-89">>true</bitxml:d-io>
  <s-io id="COM1"><binary>41540d</binary></bitxml:s-io>
</state>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<state
  xmlns="http://www.bitxml.org/v2/State.xsd"
  id-device="010101011102034"
  datetime-utc="2006-05-04T08:33:44Z"
>
  <d-io id="my-switch">>true</bitxml:d-io>
</state>
```

2.3 History Value

2.3.1 Description

The **<history>** value is the container element for a sequence of **<state>** values, collected by the history monitor, and returned by **<gethistory>** commands or encapsulated inside an **<event>** value by the history monitor (or asynchronous forwarding to the master unit).

2.3.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="History"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.bitxml.org/v2/History.xsd"
  xmlns:self="http://www.bitxml.org/v2/History.xsd"
  xmlns:state="http://www.bitxml.org/v2/State.xsd"
>

  <xs:import namespace="http://www.bitxml.org/v2/State.xsd" schemaLocation="State.xsd" />

  <xs:element name="history" type="self:HistoryValueType" />

  <xs:complexType name="HistoryValueType">
    <xs:sequence>
      <xs:element ref="state:state" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id-device" type="xs:string" use="required" />
    <xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
  </xs:complexType>

</xs:schema>
```

The **history@id-device** attribute contains the gateway logical name.

The **history@datetime-utc** attribute contains the UTC date and time when the history list has been generated.

The **history/state** elements contain the retrieved state values.

2.3.3 Semantic

None.

2.3.4 Examples

History element returned as a reply for a **<gethistory>** command:

```
<?xml version="1.0" encoding="utf-8" ?>
<history
  xmlns="http://www.bitxml.org/v2/History.xsd"
  id-device="010101011102034"
  datetime-utc="2006-03-04T00:01:02Z"
>
  <state xmlns="http://www.bitxml.org/v2/State.xsd"
```

```
id-device="010101011102034"
datetime-utc="2006-03-03T14:20:00Z"
>
  <a-io id="dss-46" >22</a-io>
  <a-io id="sad-89" >3.3</a-io>
  <d-io id="scx-45" >ERROR</d-io>
  <d-io id="ere-89" >true</d-io>
  <s-io id="COM1"><binary>41540d</binary></s-io>
</state>

<state xmlns="http://www.bitxml.org/v2/State.xsd"
id-device="010101011102034"
datetime-utc="2006-03-03T14:25:00Z"
>
  <a-io id="dss-46" >22.5</a-io>
  <a-io id="sad-89" >ERROR</a-io>
  <d-io id="scx-45" >ERROR</d-io>
  <d-io id="ere-89" >true</d-io>
  <s-io id="COM1"><binary>41540d</binary></s-io>
</state>

<state xmlns="http://www.bitxml.org/v2/State.xsd"
id-device="010101011102034"
datetime-utc="2006-03-03T14:30:00Z"
>
  <a-io id="dss-46" >22</a-io>
  <a-io id="sad-89" >NONE</a-io>
  <d-io id="scx-45" >true</d-io>
  <d-io id="ere-89" >false</d-io>
  <s-io id="COM1"><binary>41520d</binary></s-io>
</state>
</history>
```

2.4 Initialization Value

2.4.1 Description

The **<init>** element contains a complete gateway application initialization.

The initialization of a gateway application is composed by the following logical parts:

- ports set-up: definition of all the controlled ports (the ports that may be used by asynchronous and synchronous units). Not all the ports physically or logically available have to be necessarily configured.
- asynchronous monitors set-up: definition of the asynchronous monitors that must be enabled, with their specific configurations (conditions for the conditions monitor, setstate/getstate requests for history monitor, ...)
- synchronous units (currently the BiTXML commands server unit)

2.4.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Init"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/Init.xsd"
  xmlns:self="http://www.bitxml.org/v2/Init.xsd"
  xmlns:getstate="http://www.bitxml.org/v2/GetState.xsd"
  xmlns:setstate="http://www.bitxml.org/v2/SetState.xsd"
>

  <xs:import namespace="http://www.bitxml.org/v2/GetState.xsd"
  schemaLocation="GetState.xsd"/>
  <xs:import namespace="http://www.bitxml.org/v2/SetState.xsd"
  schemaLocation="SetState.xsd"/>

  <!--
    Init Node Definition
  -->
  <xs:element name="init" type="self:InitNodeType"/>

  <xs:complexType name="InitNodeType">
    <xs:sequence>

      <xs:element ref="self:io-list" minOccurs="1" maxOccurs="1" />

      <xs:choice minOccurs="1">
        <xs:element name="connected">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="self:server" minOccurs="1" maxOccurs="1"/>
              <xs:element name="events" type="self:ConnectedEventsType"
                minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="detached">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="events" type="self:DetachedEventsType" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```

    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="custom">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="self:server" minOccurs="0" maxOccurs="1"/>
      <xs:element name="events" type="self:ConnectedEventsType"
        minOccurs="0" maxOccurs="1" />
      <xs:element ref="self:properties" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>

<xs:element ref="self:reconfiguration-check" minOccurs="0" maxOccurs="1" />

</xs:sequence>

<xs:attribute name="id-device" type="xs:string" use="required" />
<xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
<xs:attribute name="boot-policy" type="self:bootPolicyType"
  use="optional" default="REQUEST" />
</xs:complexType>

<!--
  cfgPolicyType Type Definition
-->
<xs:simpleType name="bootPolicyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="REQUEST" />
    <xs:enumeration value="STANDALONE" />
  </xs:restriction>
</xs:simpleType>

<!--
  io-list Node Definition
-->
<xs:element name="io-list" type="self:ioListType" />

<xs:complexType name="ioListType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice minOccurs="1">
      <xs:element ref="self:a-io" />
      <xs:element ref="self:d-io" />
      <xs:element ref="self:s-io" />
      <xs:element ref="self:ps-o" />
      <xs:element ref="self:user-io" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<!--
  a-io Node Definition
-->
<xs:element name="a-io" type="self:aioType" />

<xs:complexType name="aioType">
  <xs:choice minOccurs="0" >
    <xs:element name="val" type="xs:double" />
    <xs:element ref="setstate:a-script" />
  </xs:choice>

```

```

</xs:choice>
<xs:attribute name="id" type="xs:string" use="required" />
<xs:attribute name="sys-id" type="xs:string" use="required" />
<xs:attribute name="type" type="self:dirType" use="optional"
  default="IO" />
<xs:attribute name="analog-min" type="xs:double" use="required" />
<xs:attribute name="analog-max" type="xs:double" use="required" />
</xs:complexType>

```

```

<!--
  d-io Node Definition
-->
<xs:element name="d-io" type="self:dioType" />

```

```

<xs:complexType name="dioType">
  <xs:choice minOccurs="0" >
    <xs:element name="val" type="xs:boolean" />
    <xs:element ref="setstate:d-script" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="sys-id" type="xs:string" use="required" />
  <xs:attribute name="type" type="self:dirType" use="optional"
    default="IO"/>
</xs:complexType>

```

```

<!--
  ps-o Node Definition
-->
<xs:element name="ps-o" type="self:psNodeType" />

```

```

<xs:complexType name="psNodeType">
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="sys-id" type="xs:string" use="required" />
</xs:complexType>

```

```

<!--
  s-io Node Definition
-->
<xs:element name="s-io" type="self:sioType" />

```

```

<xs:complexType name="sioType">
  <xs:choice minOccurs="0" >
    <xs:element name="val" type="self:condSerialIoCmdType" />
    <xs:element ref="setstate:s-script" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="sys-id" type="xs:string" use="required" />
  <xs:attribute name="type" type="self:dirType"
    use="optional" default="IO" />
  <xs:attribute name="writechar-delay" type="xs:nonNegativeInteger"
    use="optional" default="0" />
  <xs:attribute name="readchar-timeout" type="xs:nonNegativeInteger"
    use="optional" default="500" />
  <xs:attribute name="execution-delay" type="xs:nonNegativeInteger"
    use="optional" default="0" />
  <xs:attribute name="bps" type="self:BaudRateType"
    use="optional" default="9600" />

```

```

<xs:attribute name="parity-bit" type="self:ParityBitType"
  use="optional" default="NONE" />
<xs:attribute name="stop-bit" type="self:StopBitType"
  use="optional" default="1" />
<xs:attribute name="bits-char" type="self:Bits4CharType"
  use="optional" default="8" />
<xs:attribute name="dsr-dtr" type="xs:boolean"
  use="optional" default="false" />
<xs:attribute name="xon-xoff" type="xs:boolean"
  use="optional" default="false" />
<xs:attribute name="cts-rts" type="xs:boolean"
  use="optional" default="false" />
<xs:attribute name="enable-dtr" type="xs:boolean"
  use="optional" default="false" />
<xs:attribute name="enable-rts" type="xs:boolean"
  use="optional" default="false" />
</xs:complexType>

```

```

<xs:simpleType name="BaudRateType">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:enumeration value="1200" />
    <xs:enumeration value="2400" />
    <xs:enumeration value="4800" />
    <xs:enumeration value="9600" />
    <xs:enumeration value="19200" />
    <xs:enumeration value="38400" />
    <xs:enumeration value="57600" />
    <xs:enumeration value="115200" />
  </xs:restriction>
</xs:simpleType>

```

```

<xs:simpleType name="ParityBitType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="EVEN" />
    <xs:enumeration value="MARK" />
    <xs:enumeration value="NONE" />
    <xs:enumeration value="ODD" />
    <xs:enumeration value="SPACE" />
  </xs:restriction>
</xs:simpleType>

```

```

<xs:simpleType name="StopBitType">
  <xs:restriction base="xs:double">
    <xs:enumeration value="1" />
    <xs:enumeration value="1.5" />
    <xs:enumeration value="2" />
  </xs:restriction>
</xs:simpleType>

```

```

<xs:simpleType name="Bits4CharType">
  <xs:restriction base="xs:short">
    <xs:minInclusive value="5" />
    <xs:maxInclusive value="8" />
  </xs:restriction>
</xs:simpleType>

```

```

<!--
  user-io Node Definition
-->
<xs:element name="user-io" type="self:userioType" />

```

```

<xs:complexType name="userioType">
  <xs:choice minOccurs="0">
    <xs:element name="command" type="self:condUserIoCmdType" />
    <xs:element ref="setstate:u-script" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="type" type="self:dirType" use="optional"
    default="IO" />
</xs:complexType>

<!--
direction Type Definition
-->
<xs:simpleType name="dirType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="I" />
    <xs:enumeration value="O" />
    <xs:enumeration value="IO" />
  </xs:restriction>
</xs:simpleType>

<!--
server Node Definition
-->
<xs:element name="server" type="self:ServerType" />

<xs:complexType name="ServerType">
  <xs:sequence>
    <xs:element name="awake" minOccurs="0" maxOccurs="1" >
      <xs:complexType>
        <!-- awakening method name -->
        <xs:attribute name="method" type="xs:string" use="required" />
        <!-- maximum awakening duration in seconds - 0 -> forever -->
        <xs:attribute name="timeout" type="xs:nonNegativeInteger" use="optional"
          default="300" />
        <!-- inactivity timeout in seconds - 0 -> no timeout -->
        <xs:attribute name="inactivity-timeout" type="xs:nonNegativeInteger" use="optional"
          default="300" />
      </xs:complexType>
    </xs:element>
  </xs:sequence>

  <xs:attribute name="bridge-uri" type="self:serverUriType" use="required" />
  <xs:attribute name="bridge-timeout" type="self:bridgeTimeoutType" use="optional"
    default="300" />
  <xs:attribute name="bridge-retries" type="xs:nonNegativeInteger" use="optional"
    default="0" />
</xs:complexType>

<xs:simpleType name="serverUriType">
  <xs:restriction base="xs:anyURI">
  </xs:restriction>

```

```

</xs:simpleType>

<xs:simpleType name="bridgeTimeoutType">
  <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="60" />
  </xs:restriction>
</xs:simpleType>

<!--
  events Types Definition
-->
<xs:complexType name="ConnectedEventsType">
  <xs:sequence>
    <xs:element name="history-def" type="self:HistoryDefType"
      minOccurs="0" maxOccurs="1"/>
    <xs:element name="conditions-def" type="self:ConditionsDefType"
      minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DetachedEventsType">
  <xs:sequence>
    <xs:element name="history-def" type="self:HistoryDefType" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="conditions-def" type="self:ConditionsDefType" minOccurs="1"
      maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="HistoryDefType">
  <xs:sequence>

    <xs:sequence maxOccurs="unbounded">
      <xs:choice minOccurs="1">
        <xs:element ref="getstate:getstate" />
        <xs:element ref="setstate:setstate" />
      </xs:choice>
    </xs:sequence>

    <xs:element name="event-triggers" minOccurs="1" maxOccurs="1" >
      <xs:complexType>
        <xs:sequence>
          <xs:choice minOccurs="1">

            <xs:element name="on-delay" type="xs:nonNegativeInteger"
              minOccurs="0" maxOccurs="1" />

            <xs:element name="daily" minOccurs="0" maxOccurs="1" >
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="at" type="xs:time"
                    minOccurs="1" maxOccurs="unbounded" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>

            <xs:element name="weekly" minOccurs="0" maxOccurs="1" >
              <xs:complexType>
                <xs:sequence>

                  <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType>

```

```

        <xs:attribute name="weekday" type="self:WeekDayType" />
        <xs:attribute name="at" type="xs:time" />
    </xs:complexType>
</xs:element>

</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="monthly" minOccurs="0" maxOccurs="1" >
    <xs:complexType>
        <xs:sequence>

            <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="monthday" type="self:MonthDayType" />
                    <xs:attribute name="at" type="xs:time" />
                </xs:complexType>
            </xs:element>

        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="yearly" minOccurs="0" maxOccurs="1" >
    <xs:complexType>
        <xs:sequence>

            <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="yearday" type="self:YearDayType" />
                    <xs:attribute name="at" type="xs:time" />
                </xs:complexType>
            </xs:element>

        </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="delay" type="xs:nonNegativeInteger" use="required" />
<xs:attribute name="size" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<!--
Conditions definition Node Definition
-->
<xs:complexType name="ConditionsDefType">
    <xs:sequence>
        <xs:element ref="self:condition" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="condition-delay" type="xs:nonNegativeInteger"
        use="optional" default="60" />
</xs:complexType>

```

```

<xs:element name="condition" type="self:superConditionType" />
<xs:complexType name="superConditionType">
  <xs:sequence minOccurs="0">
    <xs:choice minOccurs="0" >
      <xs:element name="and" type="self:ComplexConditionType"/>
      <xs:element name="or" type="self:ComplexConditionType"/>
      <xs:group ref="self:SimpleConditionSet" />
    </xs:choice>
    <xs:element ref="self:return-ports" minOccurs="0" maxOccurs="1" />
    <xs:element ref="self:actions" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="level" type="xs:nonNegativeInteger"
    use="optional" default="0" />
  <xs:attribute name="condition-delay" type="xs:positiveInteger"
    use="optional" />
  <xs:attribute name="send-history" type="xs:boolean"
    use="optional" default="false"/>
  <xs:attribute name="send-once" type="xs:boolean"
    use="optional" default="false"/>
</xs:complexType>

<xs:complexType name="ComplexConditionType">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="and" type="self:ComplexConditionType"/>
      <xs:element name="or" type="self:ComplexConditionType"/>
      <xs:group ref="self:SimpleConditionSet" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:group name="SimpleConditionSet">
  <xs:choice>
    <xs:element name="a-io" type="self:condAioType" />
    <xs:element name="d-io" type="self:condDioType" />
    <xs:element name="s-io" type="self:condSioType" />
    <xs:element name="user-io" type="self:condUserIoType" />
    <xs:element name="time" type="self:condTimeType" />
  </xs:choice>
</xs:group>

<!--
  Time condition definition
-->
<xs:complexType name="condTimeType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="inside" type="self:trangeType"/>
      <xs:element name="outside" type="self:trangeType"/>
      <xs:element name="eq" type="xs:time"/>
      <xs:element name="ne" type="xs:time"/>
      <xs:element name="ge" type="xs:time"/>
      <xs:element name="gt" type="xs:time"/>
      <xs:element name="le" type="xs:time"/>
      <xs:element name="lt" type="xs:time"/>
    </xs:choice>
  </xs:sequence>

```

```

</xs:complexType>

<xs:complexType name="trangeType">
  <xs:attribute name="lower" type="xs:time" use="required" />
  <xs:attribute name="upper" type="xs:time" use="required" />
</xs:complexType>

<!--
  Analog port simple condition definition
-->
<xs:complexType name="condAioType">
  <xs:choice minOccurs="1">
    <xs:element name="is-error" />
    <xs:element name="is-not-error" />
    <xs:element name="inside" type="self:arangeType" />
    <xs:element name="outside" type="self:arangeType"/>
    <xs:element name="eq" type="xs:double" />
    <xs:element name="ne" type="xs:double" />
    <xs:element name="ge" type="xs:double" />
    <xs:element name="gt" type="xs:double" />
    <xs:element name="le" type="xs:double" />
    <xs:element name="lt" type="xs:double" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="arangeType">
  <xs:attribute name="lower" type="xs:double" use="required" />
  <xs:attribute name="upper" type="xs:double" use="required" />
</xs:complexType>

<xs:simpleType name="aequalValueType">
  <xs:union memberTypes="xs:double">
    <xs:simpleType>
      <xs:restriction base="xs:string" >
        <xs:enumeration value="ERROR" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<!--
  Digital port condition definition
-->

<xs:complexType name="condDioType">
  <xs:choice minOccurs="1">
    <xs:element name="is-error" />
    <xs:element name="is-not-error" />
    <xs:element name="eq" type="xs:boolean" />
    <xs:element name="ne" type="xs:boolean" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<!--
  Serial port condition definition

```

```

-->
<xs:complexType name="condSioType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="command" type="self:condSerialIoCmdType" />
      <xs:element ref="setstate:s-script"/>
    </xs:choice>
    <xs:element ref="self:response" minOccurs="1" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
</xs:complexType>

<xs:complexType name="condSerialIoCmdType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="binary-request" type="xs:boolean"
        use="optional" default="false" />
      <xs:attribute name="writechar-delay" type="xs:nonNegativeInteger" use="optional"/>
      <xs:attribute name="execution-delay" type="xs:nonNegativeInteger" use="optional" />
      <xs:attribute name="at-most" type="xs:nonNegativeInteger" use="optional" />
      <xs:attribute name="readchar-timeout" type="xs:nonNegativeInteger" use="optional" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!--
  User port condition definition
-->

<xs:complexType name="condUserIoType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="command" type="self:condUserIoCmdType" />
      <xs:element ref="setstate:u-script"/>
    </xs:choice>
    <xs:element ref="self:response" minOccurs="1" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
</xs:complexType>

<xs:complexType name="condUserIoCmdType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="binary-request" type="xs:boolean"
        use="optional" default="false" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!--
  response Node Definition
-->
<xs:element name="response" type="self:ResponseNodeType" />

<xs:complexType name="ResponseNodeType">

```

```

<xs:sequence minOccurs="1" maxOccurs="unbounded">
  <xs:choice minOccurs="1">
    <xs:element name="is-error" />
    <xs:element name="is-not-error" />
    <xs:element name="eq" type="xs:string" />
    <xs:element name="ne" type="xs:string" />
    <xs:element name="starts-with" type="xs:string" />
    <xs:element name="not-starts-with" type="xs:string" />
    <xs:element name="ends-with" type="xs:string" />
    <xs:element name="not-ends-with" type="xs:string" />
    <xs:element name="contains" type="xs:string" />
    <xs:element name="not-contains" type="xs:string" />
  </xs:choice>
</xs:sequence>
</xs:complexType>

<!--
  return Node Definition
-->
<xs:element name="return-ports" type="self:ReturnNodeType" />

<xs:complexType name="ReturnNodeType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="self:ReturnSet" />
  </xs:sequence>
</xs:complexType>

<xs:group name="ReturnSet">
  <xs:choice>
    <xs:element name="a-io" type="self:ReturnType" />
    <xs:element name="d-io" type="self:ReturnType"/>
    <xs:element name="ps-o" type="self:ReturnType"/>
    <xs:element name="s-io" type="self:SioReturnType"/>
    <xs:element name="user-io" type="self:UserIOReturnType"/>
  </xs:choice>
</xs:group>

<xs:complexType name="ReturnType">
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="SioReturnType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="val" type="self:condSerialIoCmdType" />
      <xs:element ref="setstate:s-script"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean"
    use="optional" default="false" />
</xs:complexType>

<xs:complexType name="UserIOReturnType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="val" type="self:condUserIoCmdType" />
      <xs:element ref="setstate:u-script"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

```

```

    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean"
    use="optional" default="false" />
</xs:complexType>

<!--
  actions Node Definition
-->

<xs:element name="actions" type="self:ActionsNodeType" />

<xs:complexType name="ActionsNodeType">
  <xs:sequence>
    <xs:element ref="setstate:setstate" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<!--
  Programmable reconfiguration check
-->
<xs:element name="reconfiguration-check">
  <xs:complexType>
    <xs:choice>
      <xs:element name="on-delay" type="xs:nonNegativeInteger"
        minOccurs="0" maxOccurs="1" />

      <xs:element name="daily" minOccurs="0" maxOccurs="1" >
        <xs:complexType>
          <xs:sequence>
            <xs:element name="at" type="xs:time"
              minOccurs="1" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="weekly" minOccurs="0" maxOccurs="1" >
        <xs:complexType>
          <xs:sequence>

            <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="weekday" type="self:WeekDayType" />
                <xs:attribute name="at" type="xs:time" />
              </xs:complexType>
            </xs:element>

          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="monthly" minOccurs="0" maxOccurs="1" >
        <xs:complexType>
          <xs:sequence>

            <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="monthday" type="self:MonthDayType" />
                <xs:attribute name="at" type="xs:time" />
              </xs:complexType>
            </xs:element>

          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

```

```

    </xs:complexType>
  </xs:element>

  </xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="yearly" minOccurs="0" maxOccurs="1" >
  <xs:complexType>
    <xs:sequence>

      <xs:element name="when" minOccurs="1" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="yearday" type="self:YearDayType" />
          <xs:attribute name="at" type="xs:time" />
        </xs:complexType>
      </xs:element>

    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:choice>
</xs:complexType>
</xs:element>

<xs:simpleType name="WeekDayType">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="6" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="MonthDayType">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="1" />
    <xs:maxInclusive value="31" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="YearDayType">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="1" />
    <xs:maxInclusive value="365" />
  </xs:restriction>
</xs:simpleType>

<!--
  Custom properties
-->
<xs:element name="properties">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="property" minOccurs="0" maxOccurs="unbounded" >
        <xs:complexType>
          <xs:complexContent mixed="true">
            <xs:extension base="xs:anyType">
              <xs:sequence>
                <xs:any namespace="##any" processContents="skip"
                  minOccurs="0" maxOccurs="1" />
              </xs:sequence>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required" />
    </xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

The `init@id-device` attribute contains the (target) gateway application logical name, used to uniquely identify the gateway at master unit level: it must be used in all subsequent message exchanges with the master unit.

The `init@datetime-utc` attribute contains the current UTC date and time, which may be used to synchronize the gateway real-time clock to the master unit clock.

The `init@boot-policy` attribute defines the gateway boot policy:

- if its value is REQUEST, every boot or reboot must be accomplished by a mandatory request to the master unit for the currently defined initialization value.
- if its value is STANDALONE, every boot or reboot must be accomplished by a mandatory request to the master unit for the currently defined initialization value, but if no reply is received, the gateway application can start-up with a locally stored version of the initialization configuration.

The `init/io-list` element defines the I/O ports available on the gateway

The `init/io-list/a-io` elements define the analog I/O ports available on the gateway

The `init/io-list/a-io@id` attribute defines the unique logical identifier of the analog port

The `init/io-list/a-io@sys-id` attribute defines the local system name used to identify the port.

The `init/io-list/a-io@type` attribute defines the analog port direction type (input only, output only or both, default input/output)

The `init/io-list/a-io@analog-min` attribute defines the minimum analog value (in unspecified units) available for the port

The `init/io-list/a-io@analog-max` attribute defines the maximum analog value (in unspecified units) available for the port

The `init/io-list/a-io/val` element contains the initialization value for the analog port

The `init/io-list/a-io/a-script` element contains the initialization script for the analog port

The `init/io-list/d-io` elements define the digital I/O ports available on the gateway

The `init/io-list/d-io@id` attribute defines the unique logical identifier of the digital port

The `init/io-list/d-io@sys-id` attribute defines the local system name used to identify the port.

The `init/io-list/d-io@type` attribute defines the digital port direction type (input only, output only or both, default input/output)

The `init/io-list/d-io/val` element contains the initialization value for the digital port

The `init/io-list/d-io/d-script` element contains the initialization script for the digital port

The `init/io-list/ps-o` elements define the positioning read-only ports available on the gateway

The `init/io-list/ps-o@id` attribute defines the unique logical identifier of the positioning port

The `init/io-list/ps-o@sys-id` attribute defines the local system name used to identify the port.

The `init/io-list/s-io` elements define the serial I/O ports available on the gateway

The `init/io-list/s-io@id` attribute defines the unique logical identifier for the serial port

The `init/io-list/s-io@sys-id` attribute defines the local system name used to identify the port.

The `init/io-list/s-io@writechar-delay` attribute defines (in milliseconds units) the delay applied to each write operation of every single byte transferred to the port. Default value is 0.

The `init/io-list/s-io@execution-delay` attribute defines the delay (in milliseconds units) applied before starting to read bytes from the port. Default value is 0.

The `init/io-list/s-io@readchar-timeout` attribute defines (in milliseconds units) the maximum delay before a byte read operation on the port is declared terminated. Default value is 500.

The `init/io-list/s-io@bps` attribute defines the baud rate for the port. Default is 9600.

The `init/io-list/s-io@bits-char` attribute defines the number of bits per char for the port. Default is 8.

The `init/io-list/s-io@parity-bit` attribute defines the parity for the port. Default N (no parity).

The `init/io-list/s-io@stop-bit` attribute defines the stop bit count for the port. Default 1 bit.

The `init/io-list/s-io@dsr-dtr` attribute enables dsr/dtr flow control (default false)

The `init/io-list/s-io@xon-xoff` attribute enables xon/xoff flow control (default false)

The `init/io-list/s-io@cts-rts` attribute enables cts and rts signals for hardware flow control (default false)

The `init/io-list/s-io@enable-dtr` attribute enables dtr signal for hardware flow control (default false)

The `init/io-list/s-io@enable-rts` attribute enables rts signal for hardware flow control (default false)

The `init/io-list/s-io/val` element contains the constant initialization value for the serial port

The `init/io-list/s-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus must hex binary encoded, otherwise it is assumed to be textual (the default value is false - textual value).

The `init/io-list/s-io/val@writechar-delay` attribute defines the delay in milliseconds to be applied between any two transmitted bytes of the constant initialization value (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/io-list/s-io/val@execution-delay` attribute defines the delay in milliseconds to be applied before starting to retrieve data from the port (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/io-list/s-io/val@at-most` attribute defines the maximum number of bytes to retrieve from the port.

The `init/io-list/s-io/val@readchar-timeout` attribute defines the timeout in milliseconds to be applied before declaring terminated the retrieval of data from the port (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/io-list/s-io/s-script` element contains the initialization script for the serial port (BiTXml scripts are described in Appendix A)

The `init/io-list/user-io` elements define the user I/O ports available on the gateway
The `init/io-list/user-io@id` attribute defines the unique logical identifier for the user port

The `init/io-list/user-io@sys-id` attribute defines the local system name used to identify the port.

The `init/io-list/user-io/val` element contains the initialization value for the user port

The `init/io-list/user-io/val` element contains the constant initialization value for the user port

The `init/io-list/user-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus must hex binary encoded, otherwise it is assumed to be textual (the default value is false - textual value).

The `init/io-list/user-io/u-script` element contains the initialization script for the user port

The `init/connected` element contains the gateway configuration when configured to work in connected mode.

The `init/connected/server` element contains the configuration for the on board commands server, used to process direct commands.

The `init/connected/server/awake` element contains the configuration for the awake configuration for partially connected mode.

The `init/connected/server/awake@method` attribute contains the name of the awakener to be used to detect awakenings and asleeps request.

The `init/connected/server/awake@timeout` attribute contains the awake overall timeout value, in seconds (when expired the server disconnects). A value of 0 disables the timeout.

The `init/connected/server/awake@inactivity-timeout` attribute contains the awake relative timeout value, in seconds, applied after each transmission. A value of 0 disables the timeout.

The `init/connected/server@bridge-uri` attribute defines the URI to be used as a reference to the Bridge server (*socket* URI schema must be supported)

The `init/connected/server@bridge-timeout` attribute defines the maximum time (in seconds) the command server must wait, without receiving any command or BiTXml DNS message, before declaring closed the bridge connection.

The `init/connected/events` element contains the configuration for the asynchronous events generators available for connected gateways.

The `init/connected/events/history-def` element configures the history monitor

The `init/connected/events/history-def/getstate` elements are commands which must be used by the history monitor as the iterated requests used to collect history state values

The `init/connected/events/history-def/setstate` elements are commands which must be used by the history monitor as the iterated requests used to collect history state values

The `init/connected/events/history-def@delay` attribute defines the minimum delay time (expressed in seconds) that must be applied by the history monitor (the difference between two subsequent set of state requests).

The `init/connected/events/history-def@size` attribute defines the maximum number of **<state>** elements (corresponding to the **<getstate>** or **<setstate>** commands replies) that the history monitor should store (as a FIFO list).

The `init/connected/events/history-def/event-triggers` element contains the time configuration applied by the history monitor to identify when the history content must be sent as an event to the master unit

The `init/connected/events/history-def/event-triggers/on-delay` element defines a relative (minimum) delay between two subsequent history events generations. The content of the element is the number of seconds to apply as delay.

The `init/connected/events/history-def/event-triggers/daily` element defines the history events generation relative to every day.

The `init/connected/events/history-def/event-triggers/daily/at` elements define absolute times, relative to every day, when history events generations must happen.

The `init/connected/events/history-def/event-triggers/weekly` element defines the history events generation relative to every week.

The `init/connected/events/history-def/event-triggers/weekly/when` elements define week days and absolute times, relative to every week, when history events generations must happen.

The `init/connected/events/history-def/event-triggers/weekly/when@weekday` attribute defines the week day when history events generation must happen (0=Sunday, 6=Saturday).

The `init/connected/events/history-def/event-triggers/weekly/when@at` attribute defines the time for the selected week day when history events generation must happen.

The `init/connected/events/history-def/event-triggers/monthly` element defines the history events generation relative to every month.

The `init/connected/events/history-def/event-triggers/monthly/when` elements define month days and absolute times, relative to every month, when history events generations must happen.

The `init/connected/events/history-def/event-triggers/monthly/when@monthday` attribute defines the month (calendar) day when history events generation must happen (values range from 1 to 31).

The `init/connected/events/history-def/event-triggers/monthly/when@at` attribute defines the time for the selected month day when history events generation must happen.

The `init/connected/events/history-def/event-triggers/yearly` element defines the history events generation relative to every year.

The `init/connected/events/history-def/event-triggers/yearly/when` elements define year days and absolute times, relative to every year, when history events generations must happen.

The `init/connected/events/history-def/event-triggers/yearly/when@yearday` attribute defines the year day when history events generation must happen (values range from 1 to 365).

The `init/connected/events/history-def/event-triggers/yearly/when@at` attribute defines the time for the selected year day when history events generation must happen.

The `init/connected/events/conditions-def` element enables and configures the conditions monitor.

The `init/connected/events/conditions-def@condition-delay` attribute defines (in seconds) the overall (minimum) delay that must be applied between two following applications of the conditions checks. The value may be overridden by custom condition configuration.

The `init/connected/events/conditions-def/condition` elements define the conditions to be installed into the gateway and evaluated by the history monitor. If no condition statement is established within this element then the evaluation value must be automatically set to false.

The `init/connected/events/conditions-def/condition@id` attribute defines the unique identifier for the condition.

The `init/connected/events/conditions-def/condition@level` attribute defines the condition severity level. Default value is 0.

The `init/connected/events/conditions-def/condition@condition-delay` attribute defines a custom (minimum) delay in seconds that must be applied between two following checks of the condition. If unspecified, the overall delay is applied.

The `init/connected/events/conditions-def/condition@send-history` attribute enables the forwarding of the actual history (if enabled) after the generated condition event. Default value is false.

The `init/connected/events/conditions-def/condition@send-once` attribute enables or disables the forwarding of the event when the condition holds true for two subsequent checks. The default behaviour (false value) is to send events continuously (for every condition match)

The `init/connected/events/conditions-def/condition/and` element defines the application of the overall logical connector AND to the conditions defined within its body (the condition evaluates to true if each of the sub conditions evaluates true).

The `init/connected/events/conditions-def/condition/or` element defines the application of the overall logical connector OR to the conditions defined within its body (the condition evaluates to true if at least one of the conditions evaluates to true).

The `init/connected/events/conditions-def/condition//a-io` element defines a simple condition over an analog port.

The `init/connected/events/conditions-def/condition//a-io@id` attribute defines the logical name of the analog port to be used to evaluate the condition.

The `init/connected/events/conditions-def/condition//a-io/is-error` element defines a condition which evaluates to true when the port value cannot be read.

The `init/connected/events/conditions-def/condition//a-io/is-not-error` element defines a condition which evaluates to true when the port value can be read.

The `init/connected/events/conditions-def/condition//a-io/inside` element defines a condition which evaluates to true when the port value rests inside a right-open interval of values.

The `init/connected/events/conditions-def/condition//a-io/inside@lower` attribute defines the lower interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//a-io/inside@upper` attribute defines the upper interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//a-io/outside` element defines a condition which evaluates to true when the port value rests outside a left-open interval of values.

The `init/connected/events/conditions-def/condition//a-io/outside@lower` attribute defines the lower interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//a-io/outside@upper` attribute defines the upper interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//a-io/eq` element defines a condition which evaluates to true when the port value equals the value contained into the element.

The `init/connected/events/conditions-def/condition//a-io/ne` element defines a condition which evaluates to true when the port value does not equals the value contained into the element.

The `init/connected/events/conditions-def/condition//a-io/ge` element defines a condition which evaluates to true when the port value is greater than or equals the value contained into the element.

The `init/connected/events/conditions-def/condition//a-io/gt` element defines a condition which evaluates to true when the port value is greater than the value contained into the element.

The `init/connected/events/conditions-def/condition//a-io/le` element defines a condition which evaluates to true when the port value is less than or equals the value contained into the element.

The `init/connected/events/conditions-def/condition//a-io/lt` element defines a condition which evaluates to true when the port value is less than the value contained into the element.

The `init/connected/events/conditions-def/condition//d-io` element defines a simple condition over a digital port.

The `init/connected/events/conditions-def/condition//d-io@id` attribute defines the logical name of the digital port to be used to evaluate the condition.

The `init/connected/events/conditions-def/condition//d-io/is-error` element defines a condition which evaluates to true when the port value cannot be read.

The `init/connected/events/conditions-def/condition//d-io/is-not-error` element defines a condition which evaluates to true when the port value can be read.

The `init/connected/events/conditions-def/condition//d-io/eq` element defines a condition which evaluates to true when the port value equals the value contained into the element.

The `init/connected/events/conditions-def/condition//d-io/ne` element defines a condition which evaluates to true when the port value does not equals the value contained into the element.

The `init/connected/events/conditions-def/condition//s-io` element defines a simple condition over a serial port.

The `init/connected/events/conditions-def/condition//s-io@id` attribute defines the logical name of the serial port to be used to evaluate the condition.

The `init/connected/events/conditions-def/condition//s-io@binary-result` attribute defines the type of the value read from the port: if its value is false (the default), the value is a text string, otherwise it is a binary value, and thus must be hex binary encoded

The `init/connected/events/conditions-def/condition//s-io/command` element defines a simple command to be used to retrieve a value from the port - the command is contained into the element body.

The `init/connected/events/conditions-def/condition//s-io/command@binary-request` attribute defines the format type of the request: if its value is false (the default), the request is a text string, otherwise the request is binary (and so must be hex binary encoded).

The `init/connected/events/conditions-def/condition//s-io/command@writechar-delay` attribute defines the locally applied override value (in milliseconds) of the corresponding attribute defined into the port configuration.

The `init/connected/events/conditions-def/condition//s-io/command@execution-delay` attribute defines the locally applied override value (in milliseconds) of the corresponding attribute defined into the port configuration.

The `init/connected/events/conditions-def/condition//s-io/command@at-most` attribute defines the maximum number of bytes to be read from the port.

The `init/connected/events/conditions-def/condition//s-io/command@readchar-timeout` attribute defines the locally applied override value (in milliseconds) of the corresponding attribute defined into the port configuration.

The `init/connected/events/conditions-def/condition//s-io/s-script` element defines a BiTXml script to be used to retrieve a value from the port - see appendix A for more details on scripts.

The `init/connected/events/conditions-def/condition//s-io/response` element defines the matching patterns to apply to the retrieved port value.

The `init/connected/events/conditions-def/condition//s-io/response/is-error` element defines a pattern which evaluates to true when the port value cannot be read or defined.

The `init/connected/events/conditions-def/condition//s-io/response/is-not-error` element defines a condition which evaluates to true when the port value can be read.

The `init/connected/events/conditions-def/condition//s-io/response/eq` element defines a condition which evaluates to true when the port value equals the value contained into the element.

The `init/connected/events/conditions-def/condition//s-io/response/ne` element defines a condition which evaluates to true when the port value does not equals the value contained into the element.

The `init/connected/events/conditions-def/condition//s-io/response/starts-with` element defines a condition which evaluates to true when the port literal value starts with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//s-io/response/not-starts-with` element defines a condition which evaluates to true when the port literal value does not start with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//s-io/response/ends-with` element defines a condition which evaluates to true when the port literal value ends with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//s-io/response/not-ends-with` element defines a condition which evaluates to true when the port literal value does not end with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//s-io/response/contains` element defines a condition which evaluates to true when the port literal value contains at least once the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//s-io/response/not-contains` element defines a condition which evaluates to true when the port literal value does not contain the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//users-io` element defines a simple condition over a user port.

The `init/connected/events/conditions-def/condition//user-io@id` attribute defines the logical name of the user port to be used to evaluate the condition.

The `init/connected/events/conditions-def/condition//user-io@binary-result` attribute defines the type of the value read from the port: if its value is false (the default), the value is a text string, otherwise it is a binary value, and thus must be hex binary encoded.

The `init/connected/events/conditions-def/condition//user-io/command` element defines a simple command to be used to retrieve a value from the port - the command is contained into the element body.

The `init/connected/events/conditions-def/condition//user-io/command@binary-request` attribute defines the format type of the request: if its value is false (the default), the request is a text string, otherwise the request is binary (and so must be hex binary encoded).

The `init/connected/events/conditions-def/condition//user-io/u-script` element defines a BiTXML script to be used to retrieve a value from the port - see appendix A for more details on scripts.

The `init/connected/events/conditions-def/condition//user-io/response` element defines the matching patterns to apply to the retrieved port value.

The `init/connected/events/conditions-def/condition//user-io/response/is-error` element defines a pattern which evaluates to true when the port value cannot be read or defined.

The `init/connected/events/conditions-def/condition//user-io/response/is-not-error` element defines a condition which evaluates to true when the port value can be read.

The `init/connected/events/conditions-def/condition//user-io/response/eq` element defines a condition which evaluates to true when the port value equals the value contained into the element.

The `init/connected/events/conditions-def/condition//user-io/response/ne` element defines a condition which evaluates to true when the port value does not equals the value contained into the element.

The `init/connected/events/conditions-def/condition//user-io/response/starts-with` element defines a condition which evaluates to true when the port literal value starts with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//user-io/response/not-starts-with` element defines a condition which evaluates to true when the port literal value does not start with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//user-io/response/ends-with` element defines a condition which evaluates to true when the port literal value ends with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//user-io/response/not-ends-with` element defines a condition which evaluates to true when the port literal value does not end with the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//user-io/response/contains` element defines a condition which evaluates to true when the port literal value contains at least once the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//user-io/response/not-contains` element defines a condition which evaluates to true when the port literal value does not contain the whole value contained into the element (the match must be lexicographic).

The `init/connected/events/conditions-def/condition//time/range` element defines a time condition.

The `init/connected/events/conditions-def/condition//time/inside` element defines a condition which evaluates to true when the time value rests inside a right-open interval of values.

The `init/connected/events/conditions-def/condition//time/inside@lower` attribute defines the lower interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//time/inside@upper` attribute defines the upper interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//time/outside` element defines a condition which evaluates to true when the time value rests outside a left-open interval of values.

The `init/connected/events/conditions-def/condition//time/outside@lower` attribute defines the lower interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//time/outside@upper` attribute defines the upper interval value used to specify the range interval.

The `init/connected/events/conditions-def/condition//time/eq` element defines a condition which evaluates to true when the time value equals the value contained into the element.

The `init/connected/events/conditions-def/condition//time/ne` element defines a condition which evaluates to true when the time value does not equals the value contained into the element.

The `init/connected/events/conditions-def/condition//time/ge` element defines a condition which evaluates to true when the time value is greater than or equals the value contained into the element.

The `init/connected/events/conditions-def/condition//time/gt` element defines a condition which evaluates to true when the time value is greater than the value contained into the element.

The `init/connected/events/conditions-def/condition//time/le` element defines a condition which evaluates to true when the time value is less than or equals the value contained into the element.

The `init/connected/events/conditions-def/condition//time/lt` element defines a condition which evaluates to true when the time value is less than the value contained into the element.

The `init/connected/events/conditions-def/condition/return-ports` element defines the list of port whose value must be returned when the condition holds true; these ports may be different from those used to check the condition.

The `init/connected/events/conditions-def/condition/return-ports/a-io` elements define the analog ports whose values must be returned within the generated event.

The `init/connected/events/conditions-def/condition/return-ports/a-io@id` attribute defines the logical identifier of the returned analog port.

The `init/connected/events/conditions-def/condition/return-ports/d-io` elements define the digital ports whose values must be returned within the generated event.

The `init/connected/events/conditions-def/condition/return-ports/d-io@id` attribute defines the logical identifier of the returned digital port.

The `init/connected/events/conditions-def/condition/return-ports/ps-o` elements define the positioning ports whose values must be returned within the generated event.

The `init/connected/events/conditions-def/condition/return-ports/ps-o@id` attribute defines the logical identifier of the returned positioning port.

The `init/connected/events/conditions-def/condition/return-ports/s-io` elements define the serial ports whose values must be returned within the generated event.

The `init/connected/events/conditions-def/condition/return-ports/s-io@id` attribute defines the logical identifier of the returned serial port.

The `init/connected/events/conditions-def/condition/return-ports/s-io/val` element contains the constant initialization value for the serial port

The `init/connected/events/conditions-def/condition/return-ports/s-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus must hex binary encoded, otherwise it is assumed to be textual (the default value is false - textual value).

The `init/connected/events/conditions-def/condition/return-ports/s-io/val@writechar-delay` attribute defines the delay in milliseconds to be applied between any two transmitted bytes of the constant initialization value (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/connected/events/conditions-def/condition/return-ports/s-io/val@execution-delay` attribute defines the delay in milliseconds to be applied before starting to retrieve data from the port (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/connected/events/conditions-def/condition/return-ports/s-io/val@at-most` attribute defines the maximum number of bytes to retrieve from the port.

The `init/connected/events/conditions-def/condition/return-ports/s-io/val@readchar-timeout` attribute defines the timeout in milliseconds to be applied before declaring terminated the retrieval of data from the port (the default value is defined by the port configuration at initialization time - this value works as an override).

The `init/connected/events/conditions-def/condition/return-ports/s-io/s-script` element contains the initialization script for the serial port (BiTXML scripts are described in Appendix A)

The `init/connected/events/conditions-def/condition/return-ports/user-io` elements define the user ports whose values must be returned within the generated event.

The `init/connected/events/conditions-def/condition/return-ports/user-io@id` attribute defines the logical identifier of the returned user port.

The `init/connected/events/conditions-def/condition/return-ports/user-io/val` element contains the constant initialization value for the user port

The `init/connected/events/conditions-def/condition/return-ports/user-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus must hex binary encoded, otherwise it is assumed to be textual (the default value is false - textual value).

The `init/connected/events/conditions-def/condition/return-ports/user-io/s-script` element contains the initialization script for the user port (BiTXML scripts are described in Appendix A)

The `init/connected/events/conditions-def/condition/action` defines the list of commands to be executed when the condition evaluates to true.

The `init/connected/events/conditions-def/condition/action/setstate` elements define state set-up commands.

The `init/detached` element contains the gateway configuration when configured to work in detached mode.

The `init/detached/events` element contains the configuration for the asynchronous events generators available for detached gateways - see the description for `init/connected/events` element.

The `init/detached/events/conditions-def` element enables and configures the conditions monitor - it is a mandatory element in a detached gateway configuration.

The `init/custom` element contains the gateway configuration when configured to work in custom mode.

The `init/custom/server` element contains the configuration for the on board commands server, used to process direct commands - see the description for `init/connected/server` element.

The `init/custom/events` element contains the configuration for the asynchronous events generators available for custom gateways - see the description for `init/connected/events` element.

The `init/custom/properties` element contains the list of properties defined for the custom gateway application

The `init/custom/property` element contains a custom property definition. The content of the element may be a string or a whole Xml element, and is the property value

The `init/custom/property@name` attribute contains the name of the custom property

The `init/reconfiguration-check` element contains the time configuration applied by the reconfiguration checker to identify when the request must be sent as an event to the master unit

The `init/reconfiguration-check/on-delay` element defines a relative delay between two subsequent reconfiguration request events generations. The content of the element is the number of seconds to apply as the delay.

The `init/reconfiguration-check/daily` element defines the reconfiguration check events generation relative to every day.

The `init/reconfiguration-check/daily/at` elements define absolute times, relative to every day, when reconfiguration check events generations must happen.

The `init/reconfiguration-check/weekly` element defines the reconfiguration check events generation relative to every week.

The `init/reconfiguration-check/weekly/when` elements define week days and absolute times, relative to every week, when reconfiguration check events generations must happen.

The `init/reconfiguration-check/weekly/when@weekday` attribute defines the week day when reconfiguration check events generation must happen (0=Sunday, 6=Saturday).

The `init/reconfiguration-check/weekly/when@at` attribute defines the time for the selected week day when reconfiguration check events generation must happen.

The `init/reconfiguration-check/monthly` element defines the reconfiguration check events generation relative to every month.

The `init/reconfiguration-check/monthly/when` elements define month days and absolute times, relative to every month, when reconfiguration check events generations must happen.

The `init/reconfiguration-check/monthly/when@monthday` attribute defines the month (calendar) day when reconfiguration check events generation must happen (values range from 1 to 31).

The `init/reconfiguration-check/monthly/when@at` attribute defines the time for the selected month day when reconfiguration check events generation must happen.

The `init/reconfiguration-check/yearly` element defines the reconfiguration check events generation relative to every year.

The `init/reconfiguration-check/yearly/when` elements define year days and absolute times, relative to every year, when reconfiguration check events generations must happen.

The `init/reconfiguration-check/yearly/when@yearday` attribute defines the year day when reconfiguration check events generation must happen (values range from 1 to 365).

The `init/reconfiguration-check/yearly/when@at` attribute defines the time for the selected year day when reconfiguration check events generation must happen.

2.4.3 Semantic

1. While defining gateway ports, every port value description element (`<ps-o>`, `<a-io>`, `<d-io>`, `<s-io>` and `<user-io>`) must have a unique logical identifier, different from every other name used to identify any other port value (of any type)
2. Initialization value for serial and user ports imply the reading and discarding of the resulting values
3. While defining a condition, the port type for every port referenced by the condition must be of type I (input) or input/output (IO) - bare output port cannot be used within conditions.
4. While defining a condition, if the attribute `@condition-delay` is defined, then its value overrides the default overall delay.
5. While evaluating a condition, short circuit must be applied (that is, while evaluating logically connected expressions, if the AND connector is used then if one of the sub expressions evaluates to false then the other expressions are ignored and the AND connector evaluates to false, else if the OR connector is used then if one of the sub expressions evaluates to true then the other expressions are ignored and the OR connector evaluates to true)
6. If an `<action>` element is defined for a condition, the operations it declares must be executed **after** the generation of the condition event.
7. Every error occurring during an action execution must be sent as an event to the master unit
8. If a condition defines no check to be executed, the condition is assumed to evaluate to false
9. If the history monitor is enabled, and if for a condition the history forwarding has been enabled, every `<event>` generated from that condition will be followed by another event including the current (complete) history content.
10. If during a read operation on a serial port the timeout defined by the `@readchar-timeout` attribute expires, it is assumed that the operation has terminated, and no more data can be read (for the current operation).

2.4.4 Examples

One analog and one digital port, detached gateway:

```
<?xml version="1.0" encoding="utf-8" ?>
<init
  xmlns='http://www.bitxml.org/v2/Init.xsd'
```

```
xmlns:setstate="http://www.bitxml.org/v2/SetState.xsd"
id-device='TestDevice'
datetime-utc='2006-11-20T13:00:01.00Z'
boot-policy='REQUEST'
>

<io-list>
  <a-io id='LAnalog-1' sys-id='Analog-1' analog-min='0.4' analog-max='12.3' />
  <d-io id='LDigital-1' sys-id='Digital-1' />
</io-list>

<detached>
  <events>
    <conditions-def>
      <condition id="IsError" >
        <a-io id='LAnalog-1'><is-error/></a-io>
      </condition>

      <condition id="IsNotError" >
        <a-io id='LAnalog-1'><is-not-error/></a-io>
      </condition>

      <condition id="Eq" >
        <a-io id='LAnalog-1'><eq>5.6</eq></a-io>
      </condition>

      <condition id="Ne" >
        <a-io id='LAnalog-1'><ne>5.6</ne></a-io>
      </condition>

      <condition id="Ge" >
        <a-io id='LAnalog-1'><ge>5.6</ge></a-io>
      </condition>

      <condition id="Gt" >
        <a-io id='LAnalog-1'><gt>5.6</gt></a-io>
      </condition>

      <condition id="Le" >
        <a-io id='LAnalog-1'><le>5.6</le></a-io>
      </condition>

      <condition id="Lt" >
        <a-io id='LAnalog-1'><lt>5.6</lt></a-io>
      </condition>

      <condition id="Range-within" >
        <a-io id='LAnalog-1'><inside lower='1.4' upper='3.6' /></a-io>
      </condition>

      <condition id="Range-outside" >
        <a-io id='LAnalog-1'><outside lower='1.4' upper='3.6' /></a-io>
      </condition>

      <condition id="Or" >
        <or>
          <d-io id='LDigital-1'><is-error/></d-io>
          <d-io id='LDigital-1'><eq>>true</eq></d-io>
        </or>
      </condition>

      <condition id="And" >
```

```

    <and>
      <d-io id='LDigital-1'><is-not-error/></d-io>
      <d-io id='LDigital-1'><ne>>true</ne></d-io>
    </and>
  </condition>

</conditions-def>
</events>
</detached>

</init>

```

Three analog ports, one serial port, custom gateway with complex conditions:

```

<?xml version="1.0" encoding="utf-8" ?>
<init
  xmlns='http://www.bitxml.org/v2/Init.xsd'
  xmlns:setstate="http://www.bitxml.org/v2/SetState.xsd"
  id-device='TestDevice'
  datetime-utc='2006-11-20T13:00:01.00Z'
  boot-policy='REQUEST'
  >

  <io-list>
    <a-io id='LAnalog-1' sys-id='Analog-1' analog-min='0.4' analog-max='12.3' />
    <a-io id='LAnalog-2' sys-id='Analog-2' analog-min='0.4' analog-max='12.3' type='I' />
    <a-io id='LAnalog-3' sys-id='Analog-3' analog-min='0.4' analog-max='12.3' type='O' />
    <s-io id='Modem' sys-id='COM1' />
  </io-list>

  <custom>
    <events>
      <conditions-def>

        <condition id="RetPort1" >
          <a-io id='LAnalog-1'><is-not-error/></a-io>
          <return-ports>
            <s-io id="Modem" binary-result="true" >
              <val binary-request="true">415449360d</val>
            </s-io>
          </return-ports>
        </condition>

        <condition id="RetPort2" >
          <a-io id='LAnalog-1'><is-not-error/></a-io>
          <return-ports>
            <s-io id="Modem" binary-result="false" >
              <val binary-request="true">415449360d</val>
            </s-io>
          </return-ports>
        </condition>

        <condition id="RetPort3" >
          <a-io id='LAnalog-1'><is-not-error/></a-io>
          <return-ports>
            <s-io id="Modem" binary-result="false" >
              <s-script>
                <port bps="9600" />
                <set binary-request="true" writechar-delay="1">415449360d</set>
                <delay ms="10" />
                <get readchar-timeout="40" />
              </s-script>
            </s-io>
          </return-ports>
        </condition>
      </conditions-def>
    </events>
  </custom>
</init>

```

```

<if>
  <or>
    <is-error />
    <eq>equal value</eq>
    <ne>not equal value</ne>
    <starts-with>start with value</starts-with>
    <not-starts-with>not start with value</not-starts-with>>
    <ends-with>end with value</ends-with>
    <not-ends-with>not end with value</not-ends-with>
    <contains>contains value</contains>
    <not-contains>not contains value</not-contains>
  </or>
  <then>
    <set>ATE1</set>
    <set binary-request="true">0d</set>
    <delay ms="60" />
    <get readchar-timeout="40" />
  </then>
  <else>
    <while max="10">
      <and>
        <is-not-error />
        <eq>equal value</eq>
      </and>

      <set binary-request="true" writechar-delay="1">415449370d</set>
      <delay ms="80" />
      <get readchar-timeout="50" />

      <on-max>
        <ret><set>Exit value</set></ret>
      </on-max>
    </while>
  </else>
</if>
</s-script>
</s-io>
</return-ports>
</condition>

<condition id="RetPort4" >
  <a-io id='LAnalog-1'><is-not-error/></a-io>
  <return-ports>
    <s-io id="Modem" binary-result="false" >
      <s-script>
        <port bps="9600" />
        <set binary-request="true" writechar-delay="1">415449360d</set>
        <delay ms="10" />
        <get readchar-timeout="40" />
        <ret><append>&lt;my-el&gt;</append></ret>
        <ret><append/></ret>
        <ret><append>&lt;/my-el&gt;</append></ret>
      </s-script>
    </s-io>
  </return-ports>
</condition>

</conditions-def>
</events>
</custom>

```

</init>

2.5 Gateway Reinitialization Command

2.5.1 Description

The **<reinit>** element defines the command request for a gateway reinitialization. The reinitialization may be both hot (no hardware reboot is required) and cold (hardware reboot is required).

2.5.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="ReInit"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/ReInit.xsd"
  xmlns:self="http://www.bitxml.org/v2/ReInit.xsd"
>
  <xs:element name="reinit" type="self:ReInitNodeType" >
    <xs:annotation>
      <xs:documentation>
        ReInitialization Node for remote device
      </xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:complexType name="ReInitNodeType">
    <xs:attribute name="id-device" type="xs:string" use="required" />
    <xs:attribute name="datetime-utc" type="xs:dateTime" use="optional" />
    <xs:attribute name="type" type="self:ReInitAttrType" use="required" />
  </xs:complexType>

  <xs:simpleType name="ReInitAttrType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="H" />
      <xs:enumeration value="C" />
      <!-- Awakened server applicative disconnection -->
      <xs:enumeration value="S" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

The **reinit@id-device** attribute contains the (target) gateway application logical name. The **reinit@datetime-utc** attribute contains the UTC date and time when the request has been generated.

The **reinit@type** attribute defines the reinitialization type to be applied:

- H: hot reinitialization: the gateway must issue a BOOT system event in order to request a new **<init>** configuration and reinitialize itself.
- C: cold reinitialization: the gateway should physically reboot itself, and then act as if a hot reinitialization has been issued.
- S: server monitor restart: forces a disconnection from the server monitor (silently ignored if no server monitor is available)

2.5.3 Return values

<error> value with 0 error code (no error)

2.5.4 Semantic

None

2.5.5 Examples

Hot reinitialization:

```
<?xml version="1.0" encoding="utf-8" ?>  
<reinit xmlns="http://www.bitxml.org/v2/ReInit.xsd" id-device="PC_01" type="H" />
```

2.6 State Retrieval Command

2.6.1 Description

The **<getstate>** element defines a command request to retrieve the current value of one or more I/O ports.

2.6.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="GetState"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/GetState.xsd"
  xmlns:self="http://www.bitxml.org/v2/GetState.xsd"
>

  <xs:element name="getstate">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="self:a-io"/>
          <xs:element ref="self:d-io"/>
          <xs:element ref="self:ps-o"/>
          <xs:element ref="self:s-io"/>
          <xs:element ref="self:user-io"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="id-device" type="xs:string" use="required" />
      <xs:attribute name="datetime-utc" type="xs:dateTime" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:complexType name="GetStateType">
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="self:a-io"/>
        <xs:element ref="self:d-io"/>
        <xs:element ref="self:ps-o"/>
        <xs:element ref="self:s-io"/>
        <xs:element ref="self:user-io"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="a-io" type="self:ioType" />
  <xs:element name="d-io" type="self:ioType" />
  <xs:element name="ps-o" type="self:ioType" />
  <xs:element name="s-io" type="self:sioType" />
  <xs:element name="user-io" type="self:userIoType" />

  <xs:complexType name="ioType">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
```

```

<xs:complexType name="sioType">
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
  <xs:attribute name="readchar-timeout" type="xs:positiveInteger"
    use="optional" />
  <xs:attribute name="at-most" type="xs:positiveInteger" use="optional" />
</xs:complexType>

<xs:complexType name="userIoType">
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
</xs:complexType>

</xs:schema>

```

The `getstate@id-device` attribute contains the (target) gateway application logical name.
 The `getstate@datetime-utc` attribute contains the UTC date and time when the request has been generated.

The `getstate/a-io` elements define state value request for analog I/O ports.
 The `getstate/a-io@id` attribute contains the identifier of the analog port whose state value must be retrieved.

The `getstate/d-io` elements define state value request for digital I/O ports.
 The `getstate/d-io@id` attribute contains the identifier of the digital port whose state value must be retrieved.

The `getstate/ps-o` elements define state value request for positioning devices.
 The `getstate/ps-o@id` attribute contains the identifier of the positioning device port whose state value must be retrieved.

The `getstate/s-io` elements define state value request for serial I/O ports.
 The `getstate/s-io@id` attribute contains the identifier of the serial port whose state value must be retrieved.
 The `getstate/s-io@binary-result` attribute defines the data format type of the generated result.
 The `getstate/s-io@readchar-timeout` attribute defines the override for the read timeout (in milliseconds) applied to recognize the end of the data retrieval.
 The `getstate/s-io@at-most` attribute defines the maximum number of characters to be retrieved.

The `getstate/user-io` elements define state value request for user I/O ports.
 The `getstate/user-io@id` attribute contains the identifier of the user port whose state value must be retrieved.
 The `getstate/user-io@binary-result` attribute defines the data format type of the generated result: if true, the result must be assumed binary, and returned in binary (hex binary encoded) format, if false the result must be assumed textual, and optionally may be recognized and returned as an Xml value.

2.6.3 Return value

<**state**> element containing the current states (if available) for requested ports, or an <**error**> element if some error occurred while processing the request.

2.6.4 Semantic

1. Every port description element (<**ps-o**>, <**a-io**>, <**d-io**>, <**s-io**> and <**user-io**>) must have a unique logical identifier, different from every other name used to identify any other port element (of any type) - it is not possible to request multiple times the state of the same port within a single request.
2. For serial ports, the state retrieval operation implies a simple read operation from the port.
3. If a port state is unavailable, the returned state value must be ERROR.
4. If a port is initialized as an input only port, an <**error**> value must be returned.

2.6.5 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<getstate
  xmlns="http://www.bitxml.org/v2/GetState.xsd"
  id-device="PC_01" >
  <a-io id="A-21" />
  <s-io id="COM1" />
</getstate>
```

2.7 History Retrieval Command

2.7.1 Description

The **<gethistory>** element defines the operation of history retrieval: current history content is retrieved and returned to the caller.

2.7.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="GetHistory"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/GetHistory.xsd"
  xmlns:self="http://www.bitxml.org/v2/GetHistory.xsd"
>

  <xs:element name="gethistory">
    <xs:complexType>
      <xs:attribute name="id-device" type="xs:string" use="required" />
      <xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

The `gethistory@id-device` attribute contains the target gateway logical name.

The `gethistory@datetime-utc` attribute contains the UTC date and time when the request has been generated.

2.7.3 Return value

<history> element containing the whole content of the history monitor storage, or an **<error>** element if some error occurred while processing the request.

2.7.4 Semantic

1. If the history monitor has not been enabled at initialization time, the returned value must be a valid, empty **<history>** element.

2.7.5 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<gethistory
  xmlns="http://www.bitxml.org/v2/GetHistory.xsd"
  id-device="PC_01" />
```

2.8 State Setup Command

2.8.1 Description

The **<setstate>** element defines the operation of gateway state setup. The request may involve a subset or the whole set of configured I/O ports.

2.8.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="SetState"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/SetState.xsd"
  xmlns:self="http://www.bitxml.org/v2/SetState.xsd"
>

  <xs:element name="setstate">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="self:a-io" />
          <xs:element ref="self:d-io" />
          <xs:element ref="self:s-io" />
          <xs:element ref="self:user-io" />
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="id-device" type="xs:string" use="required" />
      <xs:attribute name="datetime-utc" type="xs:dateTime" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:complexType name="SetStateType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="self:a-io" />
        <xs:element ref="self:d-io" />
        <xs:element ref="self:s-io" />
        <xs:element ref="self:user-io" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="a-io" type="self:aioType" />

  <xs:complexType name="aioType">
    <xs:choice minOccurs="1">
      <xs:element name="val" type="xs:double" />
      <xs:element ref="self:a-script" />
    </xs:choice>
    <xs:attribute name="id" type="xs:string" use="required" />
    <xs:attribute name="auto-get" type="xs:boolean" use="optional"
      default="true" />
  </xs:complexType>
```

```

<xs:element name="a-script" type="self:AnalogScriptNodeType" />

<xs:group name="AnalogScriptOps">
  <xs:choice>
    <xs:element name="delay" type="self:AnalogScriptDelayType" />
    <xs:element name="exit" type="self:AnalogScriptExitType" />
    <xs:element name="set" type="self:AnalogScriptSetType" />
    <xs:element name="get" type="self:AnalogScriptGetType" />
    <xs:element name="ret" type="self:AnalogScriptRetType" />
    <xs:element name="if" type="self:AnalogScriptIfType" />
    <xs:element name="while" type="self:AnalogScriptWhileType" />
  </xs:choice>
</xs:group>

<xs:complexType name="AnalogScriptNodeType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice minOccurs="1">
      <xs:element name="delay" type="self:AnalogScriptDelayType" />
      <xs:element name="exit" type="self:AnalogScriptExitType" />
      <xs:element name="set" type="self:AnalogScriptSetType" />
      <xs:element name="get" type="self:AnalogScriptGetType" />
      <xs:element name="ret" type="self:AnalogScriptRetType" />
      <xs:element name="if" type="self:AnalogScriptIfType" />
      <xs:element name="while" type="self:AnalogScriptWhileType" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AnalogScriptDelayType">
  <xs:attribute name="ms" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:simpleType name="AnalogScriptExitType">
  <xs:union memberTypes="xs:double">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<xs:complexType name="AnalogScriptSetType">
  <xs:simpleContent>
    <xs:extension base="xs:double" />
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="AnalogScriptGetType">
</xs:complexType>

<xs:complexType name="AnalogScriptRetType">
  <xs:choice minOccurs="1">
    <xs:element name="set" type="self:AnalogScriptRetOpType" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="AnalogScriptRetOpType">
  <xs:simpleContent>
    <xs:extension base="xs:double" />
  </xs:simpleContent>
</xs:complexType>

```

```

<xs:complexType name="AnalogScriptIfType">
  <xs:sequence>
    <xs:group ref="self:AnalogScriptConditions" minOccurs="1" maxOccurs="1"/>
    <xs:element name="then" type="self:AnalogScriptIfThenType" minOccurs="1" />
    <xs:element name="else" type="self:AnalogScriptIfThenType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:group name="AnalogScriptConditions">
  <xs:choice>
    <xs:element name="is-error"/>
    <xs:element name="is-not-error" />
    <xs:element name="and" type="self:AnalogScriptCondComplexType" />
    <xs:element name="or" type="self:AnalogScriptCondComplexType" />
    <xs:element name="eq" type="self:AnalogScriptCondSimpleType" />
    <xs:element name="ne" type="self:AnalogScriptCondSimpleType" />
    <xs:element name="inside" type="self:AnalogScriptCondRangeType" />
    <xs:element name="outside" type="self:AnalogScriptCondRangeType" />
    <xs:element name="gt" type="self:AnalogScriptCondSimpleType" />
    <xs:element name="ge" type="self:AnalogScriptCondSimpleType" />
    <xs:element name="lt" type="self:AnalogScriptCondSimpleType" />
    <xs:element name="le" type="self:AnalogScriptCondSimpleType" />
  </xs:choice>
</xs:group>

<xs:complexType name="AnalogScriptCondComplexType">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="self:AnalogScriptConditions" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="AnalogScriptCondSimpleType">
  <xs:restriction base="xs:double" />
</xs:simpleType>

<xs:complexType name="AnalogScriptCondRangeType">
  <xs:attribute name="lower" type="xs:double" use="required"/>
  <xs:attribute name="upper" type="xs:double" use="required"/>
</xs:complexType>

<xs:complexType name="AnalogScriptIfThenType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="self:AnalogScriptOps" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="AnalogScriptWhileType">
  <xs:sequence>
    <xs:group ref="self:AnalogScriptConditions" minOccurs="0" maxOccurs="1" />
    <xs:group ref="self:AnalogScriptOps" maxOccurs="unbounded"/>
    <xs:element name="on-max" type="self:AnalogScriptIfThenType" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="max" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:element name="d-io" type="self:dioType" />

```

```

<xs:complexType name="dioType">
  <xs:choice minOccurs="1">
    <xs:element name="val" type="xs:boolean" />
    <xs:element ref="self:d-script" />
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="auto-get" type="xs:boolean" use="optional"
    default="true" />
</xs:complexType>

<xs:element name="d-script" type="self:DigitalScriptNodeType" />

<xs:group name="DigitalScriptOps">
  <xs:choice>
    <xs:element name="delay" type="self:DigitalScriptDelayType" />
    <xs:element name="exit" type="self:DigitalScriptExitType" />
    <xs:element name="set" type="self:DigitalScriptSetType" />
    <xs:element name="get" type="self:DigitalScriptGetType" />
    <xs:element name="ret" type="self:DigitalScriptRetType" />
    <xs:element name="if" type="self:DigitalScriptIfType" />
    <xs:element name="while" type="self:DigitalScriptWhileType" />
  </xs:choice>
</xs:group>

<xs:complexType name="DigitalScriptNodeType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice minOccurs="1">
      <xs:element name="delay" type="self:DigitalScriptDelayType" />
      <xs:element name="exit" type="self:DigitalScriptExitType" />
      <xs:element name="set" type="self:DigitalScriptSetType" />
      <xs:element name="get" type="self:DigitalScriptGetType" />
      <xs:element name="ret" type="self:DigitalScriptRetType" />
      <xs:element name="if" type="self:DigitalScriptIfType" />
      <xs:element name="while" type="self:DigitalScriptWhileType" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DigitalScriptDelayType">
  <xs:attribute name="ms" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:simpleType name="DigitalScriptExitType">
  <xs:union memberTypes="xs:boolean">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="" />
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>

<xs:complexType name="DigitalScriptSetType">
  <xs:simpleContent>
    <xs:extension base="xs:boolean" />
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="DigitalScriptGetType">
</xs:complexType>

<xs:complexType name="DigitalScriptRetType">

```

```

<xs:choice minOccurs="1">
  <xs:element name="set" type="self:DigitalScriptRetOpType" />
</xs:choice>
</xs:complexType>

<xs:complexType name="DigitalScriptRetOpType">
  <xs:simpleContent>
    <xs:extension base="xs:boolean" />
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="DigitalScriptIfType">
  <xs:sequence>
    <xs:group ref="self:DigitalScriptConditions" minOccurs="1" maxOccurs="1"/>
    <xs:element name="then" type="self:DigitalScriptIfThenType" minOccurs="1" />
    <xs:element name="else" type="self:DigitalScriptIfThenType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:group name="DigitalScriptConditions">
  <xs:choice>
    <xs:element name="is-error"/>
    <xs:element name="is-not-error" />
    <xs:element name="and" type="self:DigitalScriptCondComplexType" />
    <xs:element name="or" type="self:DigitalScriptCondComplexType" />
    <xs:element name="eq" type="self:DigitalScriptCondSimpleType" />
    <xs:element name="ne" type="self:DigitalScriptCondSimpleType" />
  </xs:choice>
</xs:group>

<xs:complexType name="DigitalScriptCondComplexType">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="self:DigitalScriptConditions" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="DigitalScriptCondSimpleType">
  <xs:restriction base="xs:boolean" />
</xs:simpleType>

<xs:complexType name="DigitalScriptIfThenType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="self:DigitalScriptOps" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="DigitalScriptWhileType">
  <xs:sequence>
    <xs:group ref="self:DigitalScriptConditions" minOccurs="0" maxOccurs="1" />
    <xs:group ref="self:DigitalScriptOps" maxOccurs="unbounded"/>
    <xs:element name="on-max" type="self:DigitalScriptIfThenType" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="max" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:element name="s-io" type="self:sioType" />

<xs:complexType name="sioType">

```

```

<xs:choice minOccurs="1">
  <xs:element name="val">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="binary-request" type="xs:boolean"
            use="optional" default="false" />
          <xs:attribute name="writechar-delay" type="xs:nonNegativeInteger" use="optional"/>
          <xs:attribute name="execution-delay" type="xs:nonNegativeInteger" use="optional" />
          <xs:attribute name="at-most" type="xs:nonNegativeInteger" use="optional" />
          <xs:attribute name="readchar-timeout" type="xs:nonNegativeInteger" use="optional" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element ref="self:s-script" />
</xs:choice>
<xs:attribute name="id" type="xs:string" use="required" />
<xs:attribute name="binary-result" type="xs:boolean" use="optional" default="false" />
<xs:attribute name="auto-get" type="xs:boolean" use="optional"
  default="true" />
</xs:complexType>

<xs:element name="s-script" type="self:SerialScriptNodeType" />

<xs:group name="SerialScriptOps">
  <xs:choice>
    <xs:element name="port" type="self:SerialScriptPortType" />
    <xs:element name="delay" type="self:SerialScriptDelayType" />
    <xs:element name="exit" type="self:SerialScriptExitType" />
    <xs:element name="set" type="self:SerialScriptSetType" />
    <xs:element name="get" type="self:SerialScriptGetType" />
    <xs:element name="ret" type="self:SerialScriptRetType" />
    <xs:element name="if" type="self:SerialScriptIfType" />
    <xs:element name="while" type="self:SerialScriptWhileType" />
  </xs:choice>
</xs:group>

<xs:complexType name="SerialScriptNodeType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice minOccurs="1">
      <xs:element name="port" type="self:SerialScriptPortType" />
      <xs:element name="delay" type="self:SerialScriptDelayType" />
      <xs:element name="exit" type="self:SerialScriptExitType" />
      <xs:element name="set" type="self:SerialScriptSetType" />
      <xs:element name="get" type="self:SerialScriptGetType" />
      <xs:element name="ret" type="self:SerialScriptRetType" />
      <xs:element name="if" type="self:SerialScriptIfType" />
      <xs:element name="while" type="self:SerialScriptWhileType" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SerialScriptPortType">
  <xs:attribute name="bps" type="xs:positiveInteger" use="required" />
</xs:complexType>

<xs:complexType name="SerialScriptDelayType">
  <xs:attribute name="ms" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

```

```

<xs:simpleType name="SerialScriptExitType">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:complexType name="SerialScriptSetType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="binary-request" type="xs:boolean"
        use="optional" default="false" />
      <xs:attribute name="writechar-delay" type="xs:nonNegativeInteger" use="optional"
        default="0"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="SerialScriptGetType">
  <xs:attribute name="at-most" type="xs:positiveInteger" use="optional" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
  <xs:attribute name="readchar-timeout" type="xs:positiveInteger"
    use="optional" />
</xs:complexType>

<xs:complexType name="SerialScriptRetType">
  <xs:choice minOccurs="1">
    <xs:element name="clean"/>
    <xs:element name="set" type="self:SerialScriptRetOpType" />
    <xs:element name="append" type="self:SerialScriptRetOpType" />
    <xs:element name="prepend" type="self:SerialScriptRetOpType" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="SerialScriptRetOpType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <!-- from and to only used when current value is available and used -->
      <xs:attribute name="from" type="xs:nonNegativeInteger" use="optional"
        default="0"/>
      <xs:attribute name="to" type="xs:nonNegativeInteger" use="optional" />
      <!-- flag telling if the content is binary or textual -->
      <xs:attribute name="binary-value" type="xs:boolean" use="optional"
        default="false"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="SerialScriptIfType">
  <xs:sequence>
    <xs:group ref="self:SerialScriptConditions" minOccurs="1" maxOccurs="1"/>
    <xs:element name="then" type="self:SerialScriptIfThenType" minOccurs="1" />
    <xs:element name="else" type="self:SerialScriptIfThenType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:group name="SerialScriptConditions">
  <xs:choice>
    <xs:element name="and" type="self:SerialScriptCondComplexType" />
    <xs:element name="or" type="self:SerialScriptCondComplexType" />
    <xs:element name="is-error"/>
    <xs:element name="is-not-error" />
    <xs:element name="eq" type="self:SerialScriptCondSimpleType" />
    <xs:element name="ne" type="self:SerialScriptCondSimpleType" />
  </xs:choice>
</xs:group>

```

```

<xs:element name="starts-with" type="self:SerialScriptCondStartWithType" />
<xs:element name="not-starts-with" type="self:SerialScriptCondStartWithType" />
<xs:element name="ends-with" type="self:SerialScriptCondSimpleType" />
<xs:element name="not-ends-with" type="self:SerialScriptCondSimpleType" />
<xs:element name="contains" type="self:SerialScriptCondSimpleType" />
<xs:element name="not-contains" type="self:SerialScriptCondSimpleType" />
</xs:choice>
</xs:group>

<xs:complexType name="SerialScriptCondComplexType">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="self:SerialScriptConditions" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="SerialScriptCondSimpleType">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:complexType name="SerialScriptCondStartWithType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="from" type="xs:nonNegativeInteger" use="optional" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="SerialScriptIfThenType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="self:SerialScriptOps" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SerialScriptWhileType">
  <xs:sequence>
    <xs:group ref="self:SerialScriptConditions" minOccurs="0" maxOccurs="1" />
    <xs:group ref="self:SerialScriptOps" maxOccurs="unbounded"/>
    <xs:element name="on-max" type="self:SerialScriptIfThenType" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="max" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:element name="user-io" type="self:userIoType" />

<xs:complexType name="userIoType">
  <xs:choice minOccurs="1">
    <xs:element name="val">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="binary-request" type="xs:boolean"
              use="optional" default="false" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:choice>
  <xs:element ref="self:u-script" />
</xs:complexType>

```

```

</xs:choice>
<xs:attribute name="id" type="xs:string" use="required" />
<xs:attribute name="binary-result" type="xs:boolean" use="optional" default="false" />
<xs:attribute name="auto-get" type="xs:boolean" use="optional"
  default="true" />
</xs:complexType>

<xs:element name="u-script" type="self:ScriptNodeType" />

<xs:group name="ScriptOps">
  <xs:choice>
    <xs:element name="delay" type="self:ScriptDelayType" />
    <xs:element name="exit" type="self:ScriptExitType" />
    <xs:element name="set" type="self:ScriptSetType" />
    <xs:element name="get" type="self:ScriptGetType" />
    <xs:element name="ret" type="self:ScriptRetType" />
    <xs:element name="if" type="self:ScriptIfType" />
    <xs:element name="while" type="self:ScriptWhileType" />
  </xs:choice>
</xs:group>

<xs:complexType name="ScriptNodeType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:choice minOccurs="1">
      <xs:element name="delay" type="self:ScriptDelayType" />
      <xs:element name="exit" type="self:ScriptExitType" />
      <xs:element name="set" type="self:ScriptSetType" />
      <xs:element name="get" type="self:ScriptGetType" />
      <xs:element name="ret" type="self:ScriptRetType" />
      <xs:element name="if" type="self:ScriptIfType" />
      <xs:element name="while" type="self:ScriptWhileType" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ScriptDelayType">
  <xs:attribute name="ms" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

<xs:simpleType name="ScriptExitType">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:complexType name="ScriptSetType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="binary-request" type="xs:boolean"
        use="optional" default="false" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="ScriptGetType">
  <xs:attribute name="at-most" type="xs:positiveInteger" use="optional" />
  <xs:attribute name="binary-result" type="xs:boolean" use="optional"
    default="false" />
</xs:complexType>

<xs:complexType name="ScriptRetType">
  <xs:choice minOccurs="1">

```

```

    <xs:element name="clean"/>
    <xs:element name="set" type="self:ScriptRetOpType" />
    <xs:element name="append" type="self:ScriptRetOpType" />
    <xs:element name="prepend" type="self:ScriptRetOpType" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="ScriptRetOpType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <!-- from and to only used when current value is available and used -->
      <xs:attribute name="from" type="xs:nonNegativeInteger" use="optional"
        default="0"/>
      <xs:attribute name="to" type="xs:nonNegativeInteger" use="optional" />
      <!-- flag telling if the content is binary or textual -->
      <xs:attribute name="binary-value" type="xs:boolean" use="optional"
        default="false"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="ScriptIfType">
  <xs:sequence>
    <xs:group ref="self:ScriptConditions" minOccurs="1" maxOccurs="1"/>
    <xs:element name="then" type="self:ScriptIfThenType" minOccurs="1" />
    <xs:element name="else" type="self:ScriptIfThenType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:group name="ScriptConditions">
  <xs:choice>
    <xs:element name="and" type="self:ScriptCondComplexType" />
    <xs:element name="or" type="self:ScriptCondComplexType" />
    <xs:element name="is-error"/>
    <xs:element name="is-not-error" />
    <xs:element name="eq" type="self:ScriptCondSimpleType" />
    <xs:element name="ne" type="self:ScriptCondSimpleType" />
    <xs:element name="starts-with" type="self:ScriptCondStartWithType" />
    <xs:element name="not-starts-with" type="self:ScriptCondStartWithType" />
    <xs:element name="ends-with" type="self:ScriptCondSimpleType" />
    <xs:element name="not-ends-with" type="self:ScriptCondSimpleType" />
    <xs:element name="contains" type="self:ScriptCondSimpleType" />
    <xs:element name="not-contains" type="self:ScriptCondSimpleType" />
  </xs:choice>
</xs:group>

<xs:complexType name="ScriptCondComplexType">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="self:ScriptConditions" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="ScriptCondSimpleType">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:complexType name="ScriptCondStartWithType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="from" type="xs:nonNegativeInteger" use="optional" />
    </xs:extension>
  </xs:simpleContent>

```

```

</xs:complexType>

<xs:complexType name="ScriptIfThenType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="self:ScriptOps" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ScriptWhileType">
  <xs:sequence>
    <xs:group ref="self:ScriptConditions" minOccurs="0" maxOccurs="1" />
    <xs:group ref="self:ScriptOps" maxOccurs="unbounded"/>
    <xs:element name="on-max" type="self:ScriptIfThenType" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="max" type="xs:nonNegativeInteger" use="required" />
</xs:complexType>

</xs:schema>

```

The `setstate@id-device` attribute contains the target gateway logical name.

The `setstate@datetime-utc` attribute contains the UTC date and time when the request has been generated.

`setstate/a-io` elements contain state change requests for analog I/O ports

`setstate/a-io@id` attribute contains the analog port logical identifier

`setstate/a-io@auto-get` attribute controls the automatic retrieval of the port value after the set-up (true by default)

`setstate/a-io/val` element contains the constant value to set for the port

`setstate/a-io/a-script` element contains the piece of logic (as a BiTXML script) to be applied to the port to set it up (BiTXML scripts are explained in appendix A)

`setstate/d-io` elements contain state change requests for digital I/O ports

`setstate/d-io@id` attribute contains the digital port logical identifier

`setstate/d-io@auto-get` attribute controls the automatic retrieval of the port value after the set-up (true by default)

`setstate/d-io/val` element contains the constant value to set for the port

`setstate/d-io/d-script` element contains the piece of logic (as a BiTXML script) to be applied to the port to set it up (BiTXML scripts are explained in appendix A)

`setstate/s-io` elements contains state change requests for serial I/O ports

`setstate/s-io@id` attribute contains the serial port logical identifier

`setstate/s-io@auto-get` attribute controls the automatic retrieval of the port value after the set-up (true by default)

The `setstate/s-io@binary-result` attribute defines the data type of the result value (if any): if true, the result is assumed binary, and hex binary encoded, otherwise it is assumed to be textual, and no encoding is performed. As a side effect, if the result is binary and a script is used to set up the port, all constants values defined into the script code must be expressed in hex binary encoding.

`setstate/s-io/val` element contains the constant value to write to the port

`setstate/s-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus hex binary encoded, otherwise it is assumed to be textual (the default value is false - textual value).

`setstate/s-io/val@writechar-delay` attribute defines the delay in milliseconds to be applied between any two transmitted bytes (the default value is defined by the port configuration at initialization time - this value works as an override).

`setstate/s-io/val@execution-delay` attribute defines the delay in milliseconds to be applied before starting to retrieve data from the port (applied if `auto-get` is true) (the default value is defined by the port configuration at initialization time - this value works as an override).

`setstate/s-io/val@at-most` attribute defines the maximum number of bytes to retrieve from the port (applied if `auto-get` is true).

`setstate/s-io/val@readchar-timeout` attribute defines the timeout in milliseconds to be applied before declaring terminated the retrieval of data from the port (applied if `auto-get` is true) (the default value is defined by the port configuration at initialization time - this value works as an override).

`setstate/s-io/s-script` element contains the piece of logic (as a BiTXml script) to be applied to the port to set it up (BiTXml scripts are explained in appendix A)

`setstate/user-io` elements contain state change requests for user I/O ports

`setstate/user-io@id` attribute contains the serial port logical identifier

`setstate/user-io@auto-get` attribute control the automatic retrieval of the port value after the set-up (true by default)

The `setstate/user-io@binary-result` attribute defines the data type of the result value (if any): if true, the result is assumed binary, and hex binary encoded, otherwise it is assumed to be textual, and no encoding is performed. As a side effect, if the result is binary and a script is used to set up the port, all constants values defined into the script code must be expressed in hex binary encoding.

`setstate/user-io/val` element contains the constant value to write to the port

`setstate/user-io/val@binary-request` attribute defines the data type of the constant value: if true, the value to set-up is assumed binary, thus hex binary encoded, otherwise it is assumed to be textual.

`setstate/user-io/u-script` element contains the piece of logic (as a BiTXml script) to be applied to the port to set it up (BiTXml scripts are explained in appendix A)

2.8.3 Return value

`<state>` value containing the new values for the modified ports where `auto-get` is true, or an `<error>` value if some error occurred while processing the request.

2.8.4 Semantic

1. Every port description element (`<ps-o>`, `<a-io>`, `<d-io>`, `<s-io>` and `<user-io>`) must have a unique logical identifier, different from every other name used to identify any other port element (of any type) - it is not possible to request multiple times the state of the same port within a single request.
2. It is possible to change a port value only if the port type has been configured (at initialization time) as I (input port) or IO (input/output port)
3. When a request involving multiple I/O ports is issued, there must be no guaranteed execution order.
4. For serial ports set-up requests, using a constant value, termination conditions driven by attributes `at-most` and `readchar-timeout` must be applied together, and the data retrieval must be terminated by the first recognized condition.

2.8.5 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<setstate
  xmlns="http://www.bitxml.org/v2/SetState.xsd"
  id-device="fds2343fsd5f1d3s1f"
>
  <a-io id="hsja-78"><val>8.5</val></a-io>
  <d-io id="d-23"><val>true</val></d-io>
  <d-io id="d-23">
    <d-script>
      <set>true</set>
      <get/>
      <while max='150'>
        <set>true</set>
        <delay ms='2' />
        <set>false</set>
        <delay ms='2' />
      </while>
      <if>
        <eq>false</eq>
        <then>
          <ret><set>true</set></ret>
        </then>
        <else>
          <ret><set>false</set></ret>
        </else>
      </if>
    </d-script>
  </d-io>
</setstate>
```

2.9 Event value

2.9.1 Description

The **<event>** element defines an asynchronous event value.

Events may be of the following types:

- Condition event: the event will contain the condition identifier and the state of every I/O port involved in the originating condition
- System event: the event will contain a system identifier, corresponding to different run-time conditions (boot, startup, ...)
- History event: the event will contain the list of states collected by the history monitor
- Error event: the event will contain the error generated by the originating unit (may be the gateway itself, or whatever other unit)

2.9.2 Syntax

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Event"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="http://www.bitxml.org/v2/Event.xsd"
  xmlns:self="http://www.bitxml.org/v2/Event.xsd"
  xmlns:state="http://www.bitxml.org/v2/State.xsd"
  xmlns:history="http://www.bitxml.org/v2/History.xsd"
  xmlns:error="http://www.bitxml.org/v2/Error.xsd"
>

<xs:import namespace="http://www.bitxml.org/v2/History.xsd" schemaLocation="History.xsd" />
<xs:import namespace="http://www.bitxml.org/v2/Error.xsd" schemaLocation="Error.xsd" />
<xs:import namespace="http://www.bitxml.org/v2/State.xsd" schemaLocation="State.xsd" />

<xs:element name="event" type="self:EventNodeType"/>

<xs:complexType name="EventNodeType">
  <xs:choice minOccurs="1">
    <xs:element ref="self:condition" />
    <xs:element ref="self:system" />
    <xs:element ref="history:history" />
    <xs:element ref="error:error" />
  </xs:choice>
</xs:complexType>

<xs:element name="condition" type="self:CondEventValueType"/>

<xs:complexType name="CondEventValueType">
  <xs:sequence>
    <xs:element ref="state:state" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attributeGroup ref="self:EventAttrs" />
  <xs:attribute name="id-condition" type="xs:string" use="required" />
  <xs:attribute name="level" type="xs:nonNegativeInteger"
    use="optional" default="0" />
</xs:complexType>
```

```

<xs:element name="system" type="self:SysEventValueType"/>

<xs:complexType name="SysEventValueType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attributeGroup ref="self:EventAttrs" />
      <xs:attribute name="id" type="self:SysEvent" use="required" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:simpleType name="SysEvent">
  <xs:restriction base="xs:string">
    <!--
      Initialization request (expected error or init values)
    -->
    <xs:enumeration value="BOOT"/>
    <!--
      Reconfiguration request (expected error or init values)
    -->
    <xs:enumeration value="REINIT"/>
    <!--
      Startup event (expected error value)
    -->
    <xs:enumeration value="STARTUP"/>
    <!--
      Reset event (expected error value)
    -->
    <xs:enumeration value="RESET"/>
  </xs:restriction>
</xs:simpleType>

<xs:attributeGroup name="EventAttrs">
  <xs:attribute name="id-device" type="xs:string" use="required" />
  <xs:attribute name="datetime-utc" type="xs:dateTime" use="required" />
</xs:attributeGroup>

</xs:schema>

```

The `event/condition` element contains the data pertaining to a matching condition.

The `event/condition@id-device` attribute contains the originating gateway logical name.

The `event/condition@datetime-utc` attribute contains the UTC date and time when the event has been generated.

The `event/condition@level` attribute contains the severity level defined for the condition (at initialization time).

The `event/condition@id-condition` attribute contains the identifier of the condition that generated the event.

The `event/condition/state` element contains the state value related to the ports involved into the condition evaluation, together with the explicitly defined return ports (which might not be used while evaluating the condition)

The `event/system` element contains information pertaining to a system condition

The `event/system@id-device` attribute contains the originating gateway logical name.

The `event/system@datetime-utc` attribute contains the UTC date and time when the event has been generated.

The `event/system@id` attribute contains the system event identifier: may be one of the following

- BOOT - used at initialization time to request the start-up configuration
- REINIT - used by the reconfiguration checker to request the relative (modified) start-up configuration
- STARTUP - used at initialization time to signal the correct start-up termination (gateway is properly configured and running);
- RESET - used at run time to signal the gateway reset;

The `event/system` content may be freely filled by the gateway application

The `event/history` element contains the current history content (event sent by the history monitor).

The `event/error` element contains the generated error event (event sent by the system at initialization or run time).

2.9.3 Return values

1. No return value is defined for condition events
2. For BOOT system event type, return values must be either **<init>** or **<error>** values
3. For REINIT system event type, return values must be either **<init>** or **<error>** values
4. For STARTUP system event type, return value must be the **<error>** element
5. For RESET system event type, return value must be the **<error>** element

2.9.4 Semantic

None

2.9.5 Examples

```
<?xml version="1.0" encoding="utf-8" ?>
<event
  xmlns="http://www.bitxml.org/v2/Event.xsd"
>
  <condition
    id-device="010101011102034" id-condition="MY ERROR" level="5"
    datetime-utc="2006-03-03T14:27:00Z"
  >
    <state id-device="010101011102034">
      <a-io id="dss-46">23</a-io>
      <d-io id="ere-89">true</d-io>
    </state>
  </condition>
</event>
```

3. BITXML V2 Protocol Definition: Data Flows

In the following, a set of UML sequence diagrams is given in order to show how main data flows between the controller and the gateway application must happen.

While describing the data flows, a number of abbreviations have been used to identify gateway sub process, as shown in the following table:

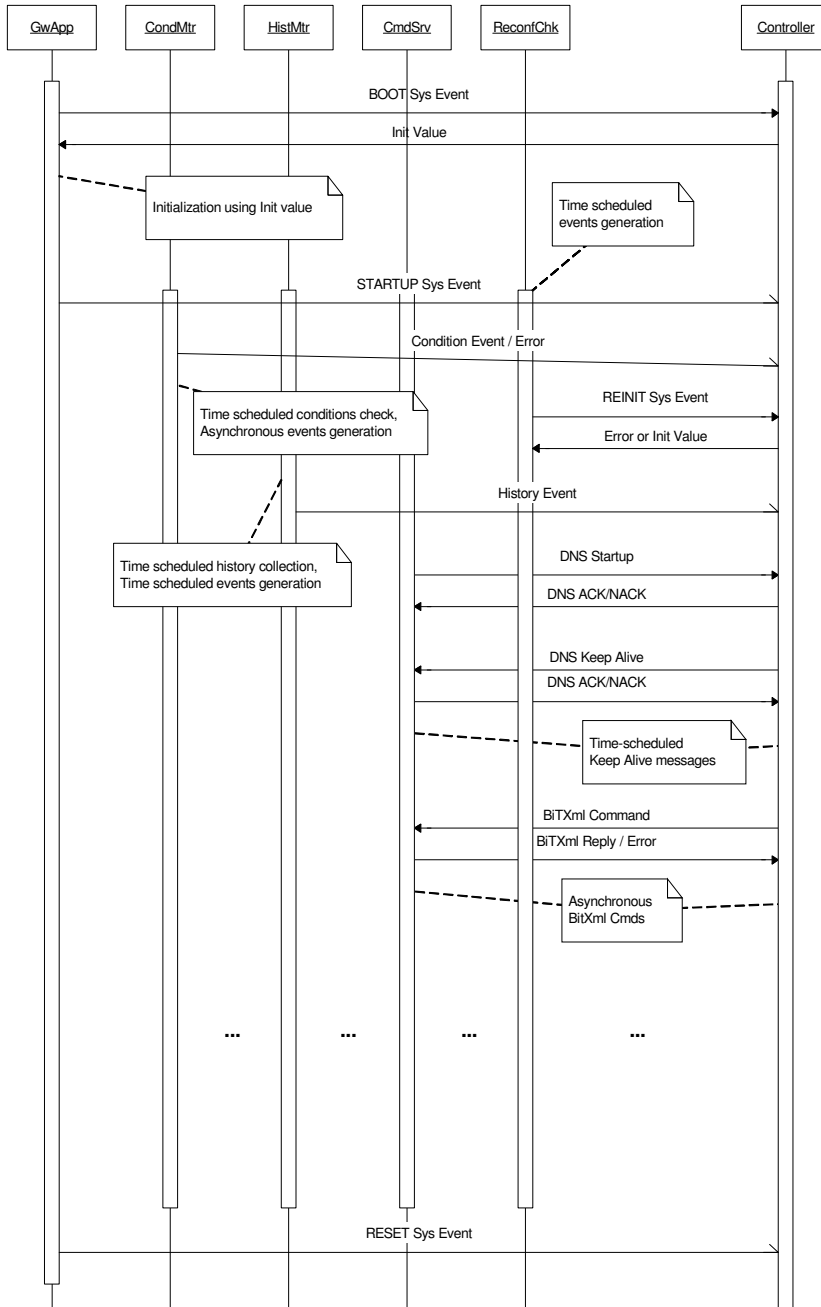
Abbreviation	Definition
GwApp	The main gateway application (or the main execution flow of the application process)
CondMtr	Condition monitor
HistMtr	History monitor
CmdSrv	BiTXML command server
ReconfigChk	Reconfiguration checker
Controller	Controlling unit (master unit) for the gateway

Six scenarios are described, in order to show the main predefined information flow for standard gateway applications:

- complete gateway initialization
- initialization error
- condition monitor data flow
- history monitor data flow
- command server
- reconfiguration checker data flow

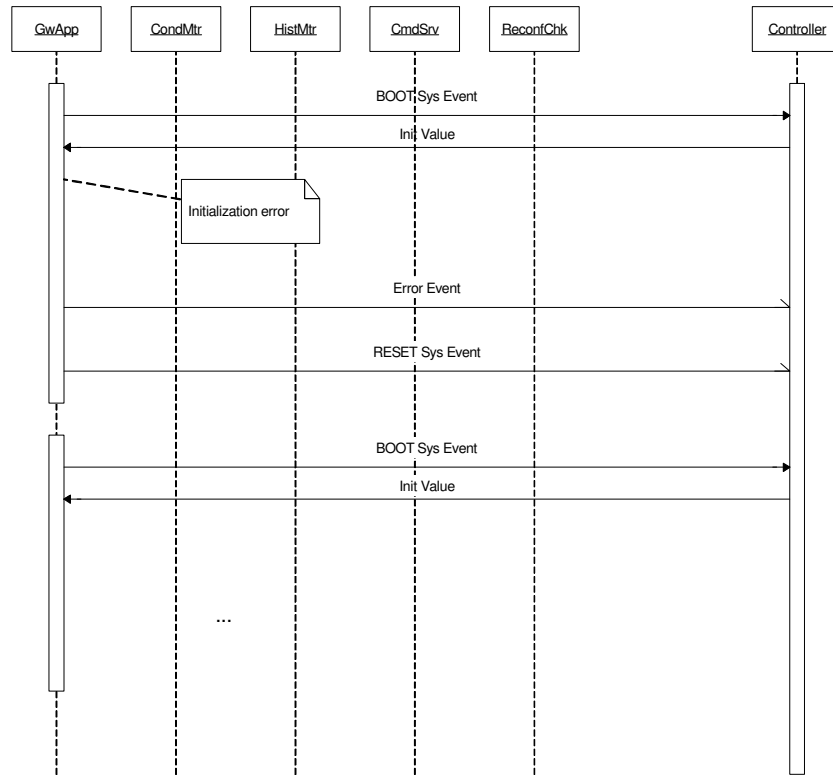
Custom applications are free to handle initialization and sub-processes activation at their wish, but, if some standard sub-process is implemented, they must follow the data flow schemas portrayed in this chapter.

3.1 Complete gateway application initialization



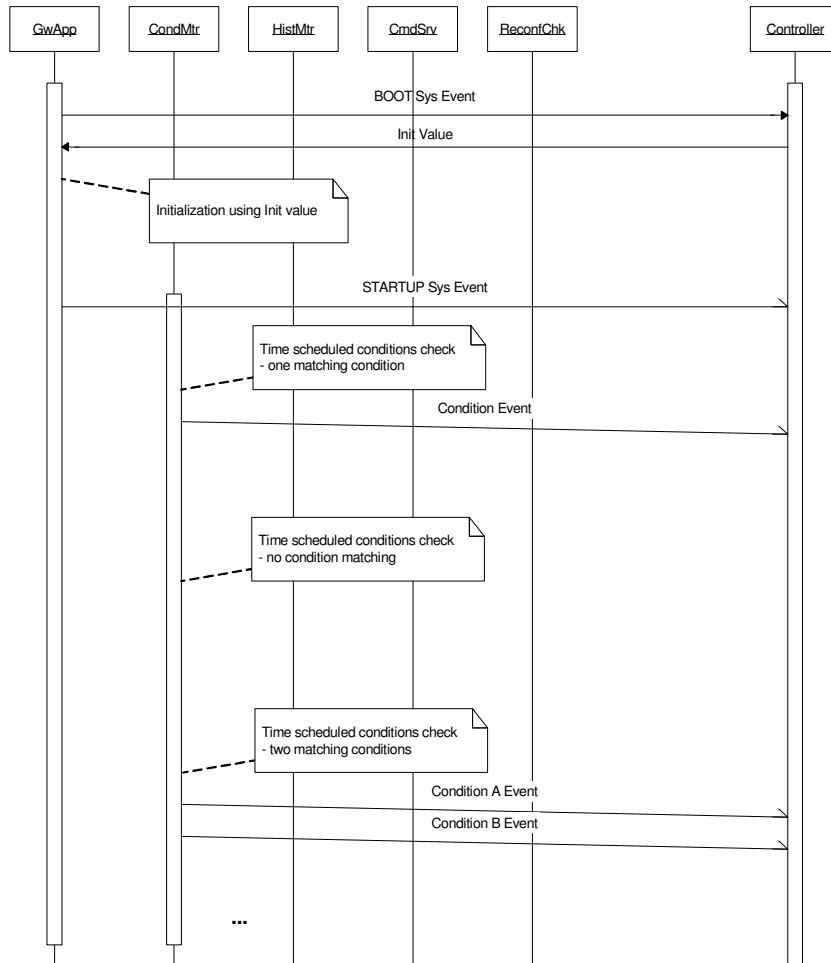
A complete gateway application activation implies the activation of the whole set of sub-processes described in the reference model - meaning we have a connected gateway with every asynchronous event generator initialized (condition monitor, history monitor, reconfiguration checker).

3.2 Initialization error data flow



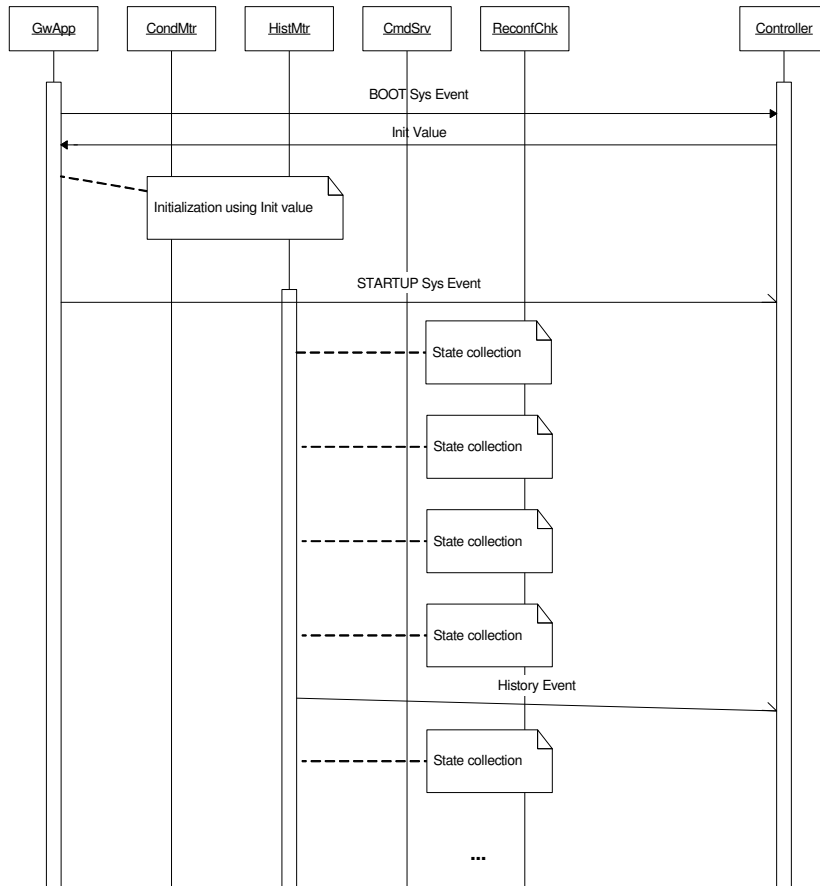
An initialization error may arise for a number of reasons (gateway internal failure, incorrect initialization, ...) and this situation is handled in the simplest way by restarting the initialization sequence, with a unspecified delay (might be minutes) before retrying the boot.

3.3 Condition monitor data flow



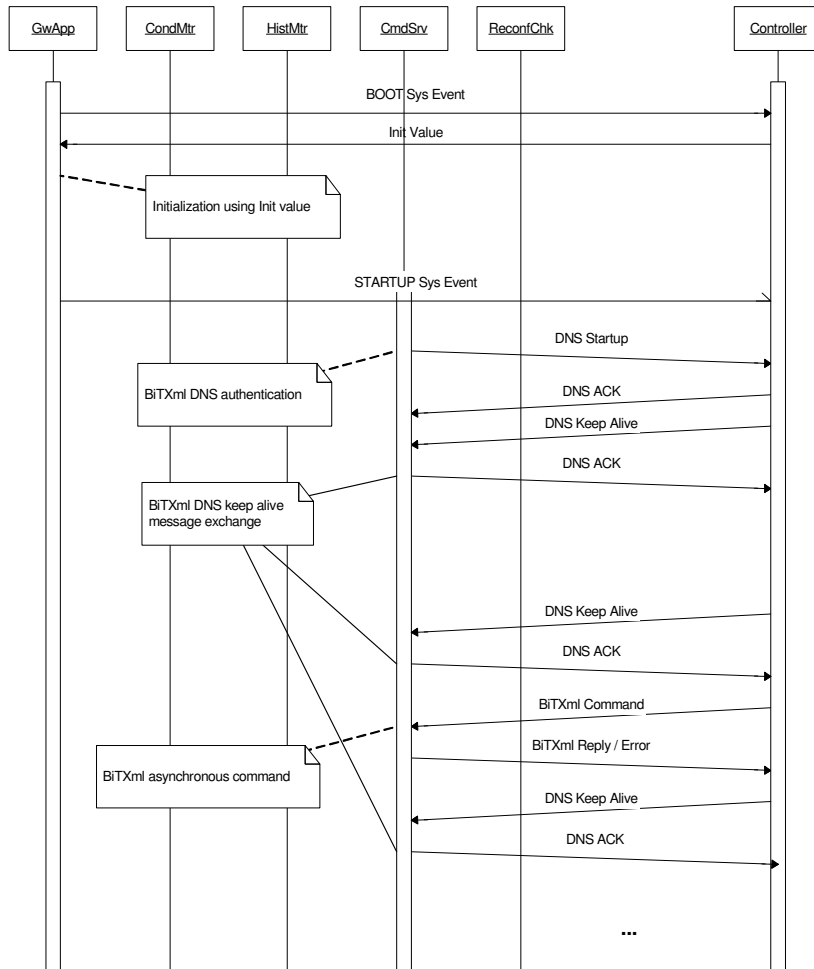
Condition monitor is a sub-process working mainly inside the gateway application, periodically checking configured conditions, and sending events when one or more of the conditions evaluates to true. The sub-process has no termination condition defined, and must explicitly be broken by the main gateway application if required to.

3.4 History monitor data flow



History monitor is a sub-process working mainly inside the gateway application, periodically generating state values, collecting them in a FIFO persistent buffer, and sending history events (a sequence of state values) when the configured time schedule table enables the forwarding. The sub-process has no termination condition defined, and must explicitly be broken by the main gateway application if required to.

3.5 Commands server data flow



The command server is the sub-process distinguishing connected gateway applications. The connection has a twofold meaning in this context:

- from an application point of view, a connection means that at any time commands may be sent to the gateway, thus possibly almost instantaneously activating or retrieving data from the controlled devices
- from a network transport point of view, a connection means that a persistent link is maintained between two parties, the gateway and the controller (or the logical part of it named *bridge* in this document), in order to consistently support the application view previously described.

The protocol does not pose any requirement on the application use of the connection, but defines explicitly how the network transport must be used to build and maintain a BiTXml connection.

Any TCP/IP or equivalent level 4 transport might be used, although the following discussion will be centred on the TCP socket toolset - being perhaps at present the most widely known level 4 abstraction (and API) layer.

The basic connection for a gateway application implementing the commands server is performed by a TCP client socket connection to the controller bridge, acting as the TCP socket server for the whole communication.

Once established the transport connection, namely the TCP client/server between the gateway and the bridge, two distinct protocols must be multiplexed on the same connection: the BiTXml DNS protocol (see appendix B for further details), and the BiTXml protocol itself, specifically the commands (setstate, getstate, reinit, gethistory) and the corresponding replies (state and error).

As a first step, DNS protocol, the gateway application must authenticate on the bridge, by sending its identifier: if the reply is a no-acknowledgment, then the server must close the connection and stop; otherwise the communication can go on.

After DNS authentication, both protocol messages may happen to be read from the connection - they can be distinguished being DNS protocol messages starting with low ASCII (non printable) characters, when BiTXml commands must start with printable characters. The end of a BiTXml command, to avoid unnecessary Xml parsing, must be marked by a NUL (ASCII code 0x00) byte value.

If a DNS protocol message is received, it must be a keep alive message: if it is not, the commands server must send a no-acknowledge message, close the connection, forward an error code 3000 (ERR_COMMUNICATION) to the controller, and shut down, in order to force a gateway application restart; otherwise, an acknowledge message must be sent back to the controller.

If a BiTXml command is completely received, then it must be executed as soon as possible and the reply sent back to the controller - the BiTXml reply too must be terminated with a NUL (ASCII code 0x00) byte value, to allow the controller to detect the end of the message.

The controller must guarantee that in no way two or more different messages, in whatever protocol, are sent concurrently to the commands server.

The controller must guarantee that every BiTXml message sent to the commands server be NUL terminated.

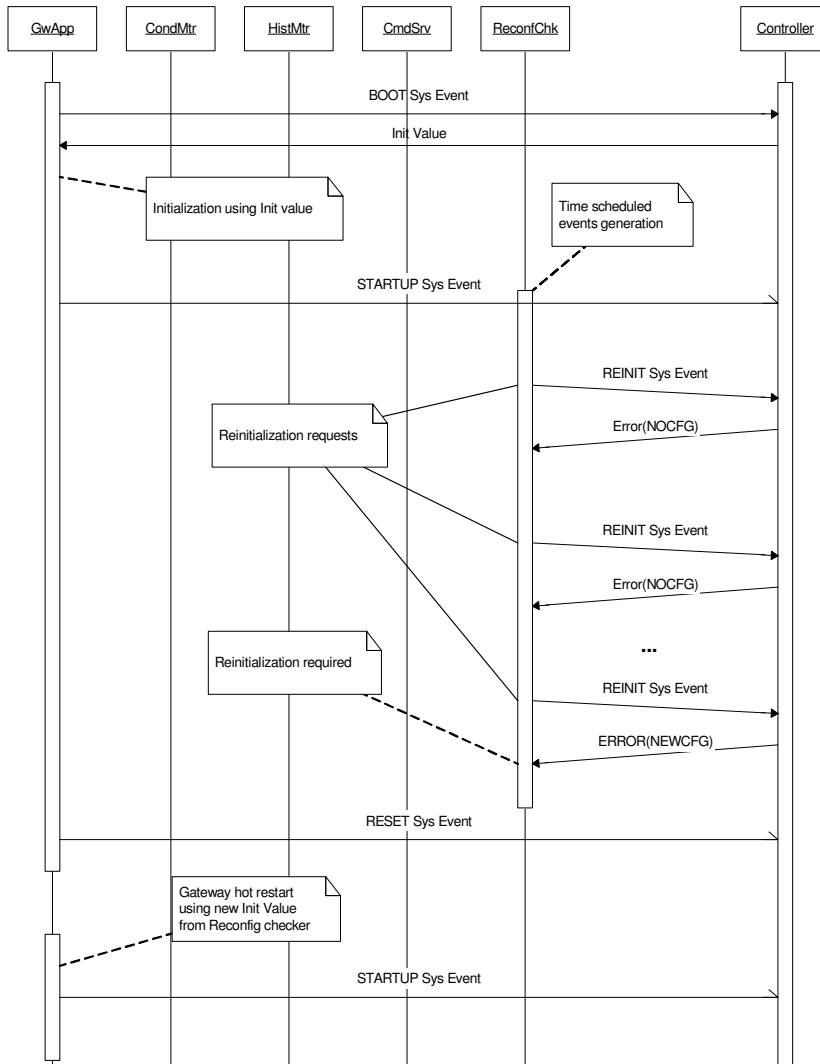
The commands server must guarantee that every BiTXml message sent to the controller be NUL terminated.

Both the controller and the commands server may set custom timeout values in order to detect unterminated BiTXml or DNS protocol messages.

If a controller detects an unterminated message, it must close immediately the connection.

If a commands server detects an unterminated message, it must close the connection, forward an error code 3000 (ERR_COMMUNICATION) to the controller, and shut down, in order to force a gateway application restart.

3.6 Reconfiguration checker data flow



Reconfiguration checker is a sub-process working mainly outside the gateway application, periodically sending REINIT system events when the configured schedule table enables the forwarding. If no updated initialization configuration is available, the controller must reply with either an error value with code 120 (ERR_NOGWCFG), or code 110 (ERR_DISABLEDGW). If some updated initialization is available, the controller must provide error code 130 (ERR_NEWGWCFG). The sub-process has no termination condition defined, and must explicitly be broken by the main gateway application if required to.

Appendix A – BiTXml V2 Scripting Language

The BiTXml scripting language is a protocol feature introduced to extend the gateway application processing power by allowing the execution of small bits of logic written in a simple Xml-based language.

The language semantic has been designed to be syntactically and semantically very simple, in order to be easily implemented, yet powerful enough to significantly enhance the expressive power of the protocol.

General description

The main purpose of the BiTXml scripting language is to generate a value from a port, by executing a set of script commands over that port. As a side effect, a script may be of its own utility apart from the return value: the language may be used as a command language to generate complex actions on the target port.

The processing model is a streamed-value model: a single port value flows through the script, from the first instruction to the last, and each instruction applies zero or some operators in order to change the flowing value, or leave it go while executing other (command specific) tasks. The script return value by default is the stream value: the return value may however be composed by using script operators.

Syntax

<if> Conditional statement

```
<if>
  condition
  <then>
    statements
  </then>
  [<else>
    statements
  </else>]
</if>
```

The <if> conditional statement models a single condition branch: if the condition evaluates to true, the <then> branch is executed, otherwise, if present, the <else> branch is executed.

Conditions may be different depending on the type of port the script is executed upon:

- for analog ports, the following operators are allowed
 - <and> and <or>, as general logical connectors for nested conditions
 - <is-error>, <is-not-error> to test if the port value is available or not
 - <eq>, <ne> to check for equality and inequality relationships between the stream value and some constant value
 - <gt>, <ge>, <lt>, <le> to test for ordering relation between the stream value and some constant value
 - <inside> and <outside> to test the stream value over constant ranges
- for digital ports, the following operators are allowed

- `<and>` and `<or>`, as general logical connectors for nested conditions
- `<is-error>`, `<is-not-error>` to test if the port value is available or not
- `<eq>`, `<ne>` to check for equality and inequality relationships between the stream value and some constant value
- for serial and user ports, the following operators are allowed
 - `<and>` and `<or>`, as general logical connectors for nested conditions
 - `<is-error>`, `<is-not-error>` to test if the port value is available or not
 - `<eq>`, `<ne>` to check for equality and inequality relationships between the stream value and some constant value
 - `<starts-with>`, `<not-starts-with>` to test if the stream value starts (or does not starts, respectively) with the (lexicographically checked) operand value
 - `<ends-with>`, `<not-ends-with>` to test if the stream value ends (or does not ends, respectively) with the (lexicographically checked) operand value
 - `<contains>`, `<not-contains>`

`<then>` and `<else>` branches can contain whatever statement or operator is defined for the kind of port the script is run upon.

`<while>` Iteration statement

```
<while max='non negative value'>
  [condition]
  statements
  [<on-max>
    statements
  </on-max>]
</while>
```

The `<while>` iteration statement models a loop controlled by an optional condition and an upper numeric limit to the cycles count: while the condition evaluates to true and the loop counter is less than the maximum defined value, then the statements contained within the `<while>` statement are sequentially executed. If the condition evaluates to false, the loop is exited; if no condition is defined, the loop is executed exactly the maximum defined value times.

In any case, if the maximum defined value is reached, if defined, the `<on-max>` clause is executed.

`<delay>` Operator

The delay operator makes the script wait for a specified amount of time: the attribute *ms* defines the number of milliseconds to wait for.

The stream value is not affected by this operator.

`<exit>` Operator

The exit operator terminates the script: an optional constant operand will become the script return value.

The stream value is not affected by this operator.

<set> Operator

The <set> operator changes the port value by writing a value to it. It is a polymorphic operator, being its parameters dependent from the type of port used by the script:

- if the port handles analog values, the value to set is a double value;
- if the port handles digital values, the value to set is a boolean (logical) value;
- if the port is a serial interface, two optional attributes control the operator
 - *binary-request* (optional) defines the type of value to be set: if true, the value is binary, and must be hex binary encoded, otherwise (the default) the value is a text string
 - *writchar-delay* (optional) defines the delay in milliseconds to apply between any two consecutive byte write - if specified, overrides the default value stated with the port definition (BiTXml initialization value)
- if the port is a user port, one optional attribute controls the operator
 - *binary-request* (optional) defines the type of value to be set: if true, the value is binary, and must be hex binary encoded, otherwise (the default) the value is a text string

The stream value is not affected by this operator.

<get> Operator

The <get> operator reads the port value and places it on the stream value, overriding the previous value. It is a polymorphic operator, being its parameters dependent from the type of port used by the script:

- if the port handles analog or digital values, no further attribute can be used;
- if the port is a serial interface, three optional attributes control the operator
 - *binary-result* (optional) defines the type of the return value: if true, the value is binary, and must be hex binary encoded, otherwise (the default) the value is a text string
 - *at-most* (optional) defines the maximum number of bytes to read from the port
 - *readchar-timeout* (optional) defines the timeout in milliseconds to declare terminate the read operation - if specified, overrides the default value stated with the port definition (BiTXml initialization value)
- if the port is a user port, one optional attribute controls the operator
 - *binary-result* (optional) defines the type of the return value: if true, the value is binary, and must be hex binary encoded, otherwise (the default) the value is a text string

<ret> Operator

The <ret> operator is used to define or compose the return value generated by the script. If the operator is used, the stream value is not used as the script result. <exit> operator can override both stream and return values by forcing a constant to be the script exit value.

<ret> operator is polymorphic, being its operators dependent from the type of port used by the script:

- if an analog port is used, the operator is a constant double value
- if a digital port is used, the operator is a constant boolean value
- if a serial or a user port is used, four operators are allowed:

- `<clean>` erases any previous result value
- `<set>` absolutely assigns the return value: if a value is specified, that value is used, otherwise the stream value is taken. Optionally, the value can be cut, using *to* and *from* attributes, or hex binary encoded (setting optional attribute *binary-value* value to true)
- `<append>` appends to the already existing return value: if an operand value is specified, that value is used, otherwise the stream value is taken. Optionally, the value can be cut, using *to* and *from* attributes, or hex binary encoded (setting optional attribute *binary-value* value to true). If no return value is defined, the operator is functionally equivalent to `<set>`
- `<prepend>` prepends to the already existing return value: if an operand value is specified, that value is used, otherwise the stream value is taken. Optionally, the value can be cut, using *to* and *from* attributes, or hex binary encoded (setting optional attribute *binary-value* value to true). If no return value is defined, the operator is functionally equivalent to `<set>`

`<port>` Operator

The `<port>` operator can be used only in serial port scripts, and allows the change of the baud rate of the port: the attribute *bps* controls the baud rate to set - a value of 0 resets to the default baud rate defined with the port initialization.

Examples

Generation of a square wave on a digital port, with period 4ms, and duration 600ms:

```
<d-script>
  <while max='150'>
    <set>true</set>
    <delay ms='2' />
    <set>false</set>
    <delay ms='2' />
  </while>
</d-script>
```

Generation of a saw-tooth wave on an analog port, with frequency 83.3Hz, and duration 12 seconds:

```
<a-script>
  <while max='1000'>
    <set>0.0</set>
    <delay ms='2' />
    <set>0.2</set>
    <delay ms='2' />
    <set>0.4</set>
    <delay ms='2' />
    <set>0.6</set>
    <delay ms='2' />
    <set>0.8</set>
```

```
<delay ms='2' />
<set>1.0</set>
<delay ms='2' />
</while>
</a-script>
```

Script to check if a serial port is connected to an AT modem:

```
<s-script>
<port bps='9600' />
<set binary-request='true'>41540D</set>
<delay ms='100' />
<get binary-result='true' readchar-timeout='500' />
<if>
<contains>4F4B</contains>
<then>
<!-- is a modem -->
<exit>OK</exit>
</then>
<else>
<!-- is NOT a modem -->
<exit>KO</exit>
</else>
</if>
</s-script>
```

Appendix B – BiTXml DNS protocol

Description

The BiTXML DNS protocol has been designed to allow the continuous tracking of gateway network configuration in those situations where the gateway network address may vary – typically on a per connection base –.

The implementation of the BiTXML DNS client inside the gateway allows the master unit to instantly be informed when the network configuration of the gateway has changed, and cope with these changes with little or no impact on the application programming logic of the gateway/master interaction.

The BiTXML DNS protocol has been kept very simple in order to minimize the data transfers between the gateway DNS client and the master unit DNS server: the result is an extremely low network band consumption, which translates directly to negligible costs on those networks where the billing depends on the generated traffic.

On specific networks (i.e. GPRS), the BiTXML DNS client may have the (perhaps mandatory) purpose of keeping the network connection up.

Start-up message

The start-up message is the first message a BiTXml DNS client must send to the corresponding BiTXml server in order to register its network address with its logical name. The start-up message may actually be of two different types:

1. If the gateway logical name may be expressed as a 32 bit quantity, the message can be made from five bytes:
 - a. the first byte codes the start-up message,
 - b. the following four contain the 32 bit quantity in big endian byte ordering.

0x01	0xXX	0xXX	0xXX	0xXX
------	------	------	------	------

2. If the gateway logical name must be expressed with an ASCII string, the message can be made from a variable number of bytes, where
 - a. the first one codes the start-up message itself,
 - b. the second one codes the length in bytes of the logical name, and the remaining bytes contain the ASCII representation of the gateway name

0x02	0xNN	0xXX	0xXX	...
------	------	------	------	-----

Start-up reply may be one of:

1. Acknowledge: start-up correctly executed, one byte message is returned.

0x04

2. No acknowledge: error during the start-up phase, one byte message is returned.

0x05

Keep-alive message

The keep-alive message can be sent once the start-up message has been processed by the BiTXML DNS server and an acknowledge message has been returned. From this point on, until a network disconnection happens or a No acknowledge message is received, the BiTXML DNS client is free to send keep-alive messages at whatever required frequency: the DNS client start-up configuration might define such a frequency, which, for currently network applications, may range from dozens of seconds to minutes or more.

The keep-alive message is made from one single byte

0x03

Keep-alive reply may be one of:

1. Acknowledge: keep-alive correctly processed, one byte message is returned.

0x04

2. No acknowledge: keep-alive processing requires a disconnection from the server, one byte message is returned.

0x05

Appendix C - Sample Gateway Application Start-up Meta Code

```
persistent URI masterUri = "http://...";
persistent string gatewayID = "XYZ";
persistent bool requestBoot = true;
persistent XmlDocument lastInitialization = null;

int main()
{
    XmlDocument bootEvent = "<event ... <system id='BOOT' ...";
    XmlDocument reconfigEvent = "<event ... <system id='REINIT' ...";
    XmlDocument& startupEvent = bootEvent;
    XmlDocument& response;
    XmlDocument errorEvent = "<event ... <error code='500' ...";
    int connectionDelay = 60;
    bool reset = false;

    bool errorStartingConditions = false;
    bool errorStartingHistory = false;
    bool errorStartingReconfigChecker = false;
    bool errorStartingDNS = false;
    bool errorStartingServer = false;

    ...

    while (1)
    {
        // Startup request
        //
        response = HTTPPost(masterURI, startupEvent);

        //
        // POST: response can be valid OR an http error occurred
        //

        if (HTTPError == true)
        {
            // Handle http error
            //
            if (requestBoot || lastInitialization == null)
            {
                sleep(connectionDelay);
                continue;
            }
            else
            {
                response = lastInitialization;
            }
        }

        //
        // POST: response is a valid reply from master unit
    }
}
```

```
//
if (response.RootElementName == "error")
{
    // Handle error response
    //
    switch (response.AttributeValue("/error@code"))
    {
        case "100":
            // Gateway is not registered by the master unit
            //
            StandBy();
            break;

        case "110":
            // Gateway is registered but not operable
            //
            StandBy();
            break;

        case "120":
            // No configuration available
            //
            sleep(connectionDelay);
            continue;
            break;

        default:
            sleep(connectionDelay);
            continue;
            break;
    }
}

//
// POST: response is a valid initialization configuration
//

if (!InitializeIOList(response))
{
    // Handle invalid io list (ports configuration)
    // Switch startup event to check configuration changes
    //
    startupEvent = reconfigEvent;

    sleep(connectionDelay);

    continue;
}

//
// POST: ports initialization is valid
//

if (!InitializeConditions(response))
{
```

```
// Handle invalid conditions configuration
// Switch startup event to check configuration changes
//
startupEvent = reconfigEvent;

continue;
}

//
// POST: ports initialization and conditions initialization are valid
//

if (!InitializeHistory(response))
{
    // Handle invalid history configuration
    // Switch startup event to check configuration changes
    //
    startupEvent = reconfigEvent;

    sleep(connectionDelay);

    continue;
}

//
// POST: ports initialization and conditions initialization
// and history initialization are valid
//

if (!InitializeReconfigChecker(response))
{
    // Handle invalid reconfiguration check configuration
    // Switch startup event to check configuration changes
    //
    startupEvent = reconfigEvent;

    sleep(connectionDelay);

    continue;
}

//
// POST: ports initialization and conditions initialization
// and history initialization and reconfiguration check initialization
// are valid
//

if (!InitializeServer(response))
{
    // Handle invalid server configuration
    // Switch startup event to check configuration changes
    //
    startupEvent = reconfigEvent;

    sleep(connectionDelay);
}
```

```
        continue;
    }

    //
    // POST: initialization completed
    //

    lastInitialization = response;

    // Ports values initialization
    //
    InitializePorts();

    // Conditions monitor startup
    //
    if (!StartConditions())
    {
        // Handle conditions monitor startup error
        //
        if (!errorStartingConditions)
            SendErrorEvent(501);

        errorStartingConditions = true;

        sleep(connectionDelay);

        continue;
    }

    errorStartingConditions = false;

    // History monitor startup
    //
    if (!StartHistory())
    {
        // Handle history monitor startup error
        //
        if (!errorStartingHistory)
            SendErrorEvent(502);

        errorStartingHistory = true;

        sleep(connectionDelay);

        continue;
    }

    errorStartingHistory = false;

    // Reconfiguration monitor startup
    //
    if (!StartReconfigChecker())
    {
        // Handle reconfiguration checker startup error
        //
```

```
    if (!errorStartingReconfigChecker)
        SendErrorEvent(503);

    errorStartingReconfigChecker = true;

    sleep(connectionDelay);

    continue;
}

errorStartingReconfigChecker = false;

// DNS startup
//
if (!StartDNS())
{
    // Handle DNS client startup error
    //
    if (!errorStartingDNS)
        SendErrorEvent(505);

    errorStartingDNS = true;

    sleep(connectionDelay);

    continue;
}

errorStartingDNS = false;

// Server startup
//
if (!StartServer())
{
    // Handle server startup error
    //
    if (!errorStartingServer)
        SendErrorEvent(504);

    errorStartingServer = true;

    sleep(connectionDelay);

    continue;
}

errorStartingServer = false;

//
// POST: initialization completed
//

SendSysEvent(STARTUP);
```

```
        while (!reset)
        {
            sleep(10);
        }

        reset = false;

        SendSysEvent(RESET);
    }
}

void StandBy()
{
    while(1)
        sleep(10);
}

bool InitializeIOList()
{
    // ... I/O list check and initialization
}

bool InitializeConditions()
{
    // ... Conditions check and initialization
}

bool InitializeHistory()
{
    // ... History check and initialization
}

bool InitializeReconfigChecker()
{
    // ... Reconfiguration check and initialization
}

bool InitializeServer()
{
    // ... Server check and initialization
}

void InitializePorts()
{
    // ... Ports values initialization
}

bool StartConditions()
{
    // ... Conditions monitor startup
}
```

```
bool StartHistory()
{
    // ... History monitor startup
}

bool StartReconfigChecker()
{
    // ... Reconfiguration monitor startup
}

bool StartDNS()
{
    // ... DNS startup
}

bool StartServer()
{
    // ... Server startup
}
```