

addUPI Specification

URL programming interface

Version 1.3

Version:	1.3
Date:	13.09.2011
Status:	FINAL
Author:	M. Wittmann

Proprietary Notice:

The Adcon logo, the A720 series and addIT™, the A730 series and addWAVE, A8x0 series and addVANTAGE®, are trademarks or registered trademarks of Adcon Telemetry GmbH. All other registered names used throughout this publication are trademarks of their respective owners.

This publication contains confidential information, property of Adcon Telemetry GmbH. Disclosure to third parties of the information contained herein is prohibited. Neither the whole nor any part of the information contained in this publication may be reproduced in any material form except with the prior written permission of Adcon Telemetry GmbH.

Table of contents

1. Scope.....	4
2. Nodes and their local / remote ID.....	4
3. Quick introduction.....	5
4. addUPI Reference.....	9
4.1 General Format of a Request.....	9
4.2 Response Format.....	9
4.3 addUPI Functions.....	9
4.3.1 Authentication Functions.....	10
4.3.2 Configuration functions.....	12
4.3.3 Data transfer functions.....	15
4.3.4 Node specific functions.....	18
4.4 Data types.....	19
4.4.1 Standard data types.....	19
4.4.2 Enumeration data types.....	20
4.5 Date/Time Handling.....	21
4.6 Templates.....	22
5. XSD (XML Schema Definition)	23

Change log

1.3	2011-08-10	<ul style="list-style-type: none">• Rewrote the specification to make it easier to understand it.<ul style="list-style-type: none">◦ Added quick introduction◦ Removed error codes and list of templates.◦ Re-ordered chapters to move less important chapters to the back.◦ Put the XML examples to the functions.◦ Using XSD instead of DTD (better code generator tools available)• Removed notifications and thus the events• Removed da (date anchor) parameter – it can be calculated by the duration/offset of the value• Removed host-id parameter from login function• Removed cache parameter• Removed setattrib function
-----	------------	--

1. Scope

This paper defines the communication protocol between various TCP/IP capable components of Adcon Telemetry's system. Some examples of components that may implement this protocol are given below:

- an addVANTAGE Pro client and an addVANTAGE Pro server
- an addVANTAGE Pro client and a Telemetry Gateway (e.g. an A850)
- A2A (addUPI to ASCII) tool to either an addVANTAGE Pro server or Telemetry Gateway

The specification described herein belongs to the application layer according to the ISO OSI layered system.

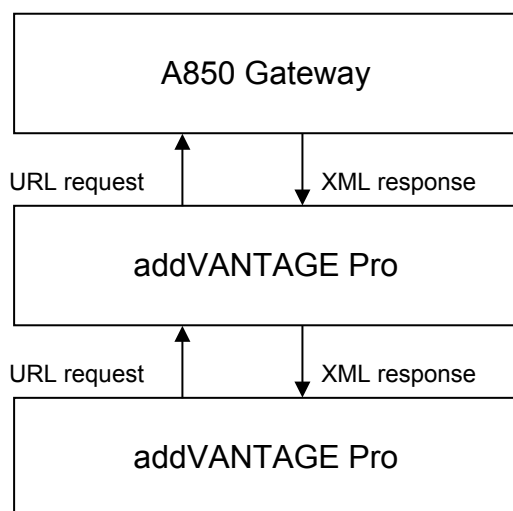


Figure 1: Typical use case for addUPI

2. Nodes and their local / remote ID

Adcon's addUPI servers are handling various information grouped together in entities called "nodes". These nodes can have child nodes, making the data structure a tree. Each node has an ID that is to be unique on this server. Thus when requesting the addUPI server using a node ID, this node (and its subnodes – depending on request) is used for the response, or none if this ID is unknown.

The addUPI client on the other hand can import nodes, giving them a new ID that is unique locally, but also it needs to keep the ID on the addUPI server, called the “remote ID”.

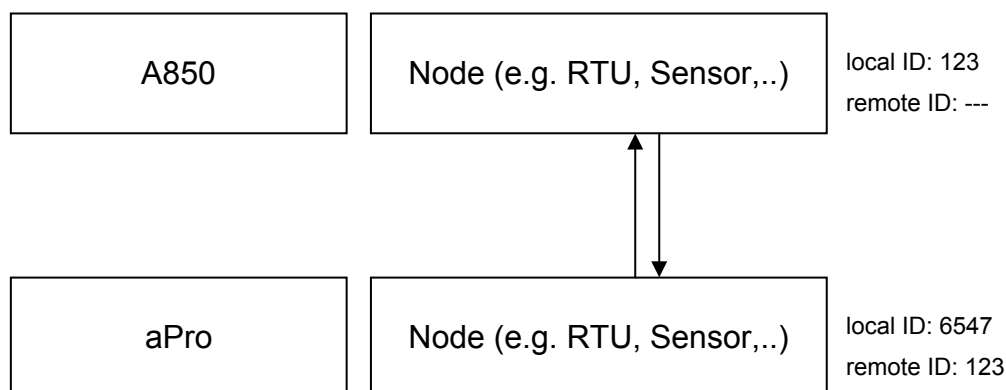


Figure 2: addVANTAGE Pro holding the remote ID as reference

3. Quick introduction

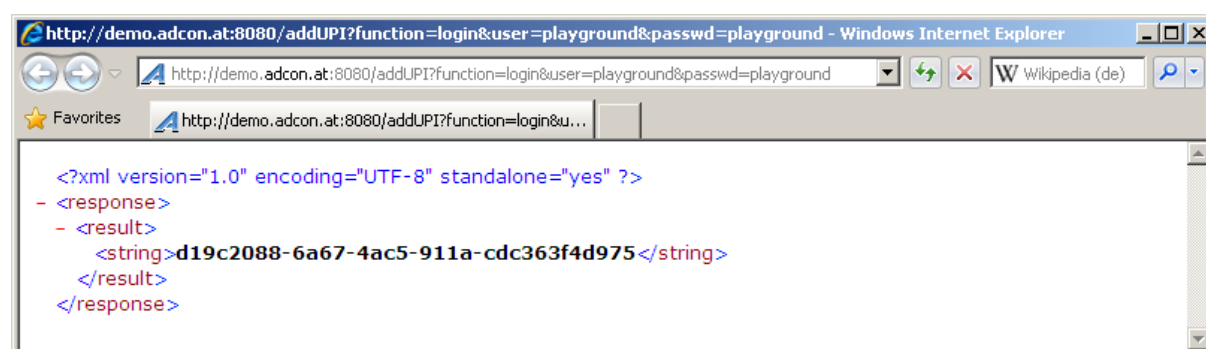
For this small introduction, you can use your web browser (Mozilla Firefox, Microsoft Internet Explorer,...). We're sending request information by the URL and get XML as response. Both information is easy to display in the browser.

addUPI is stateful, this means the client must login first:

request:

<http://demo.adcon.at/addUPI?function=login&user=playground&passwd=playground>

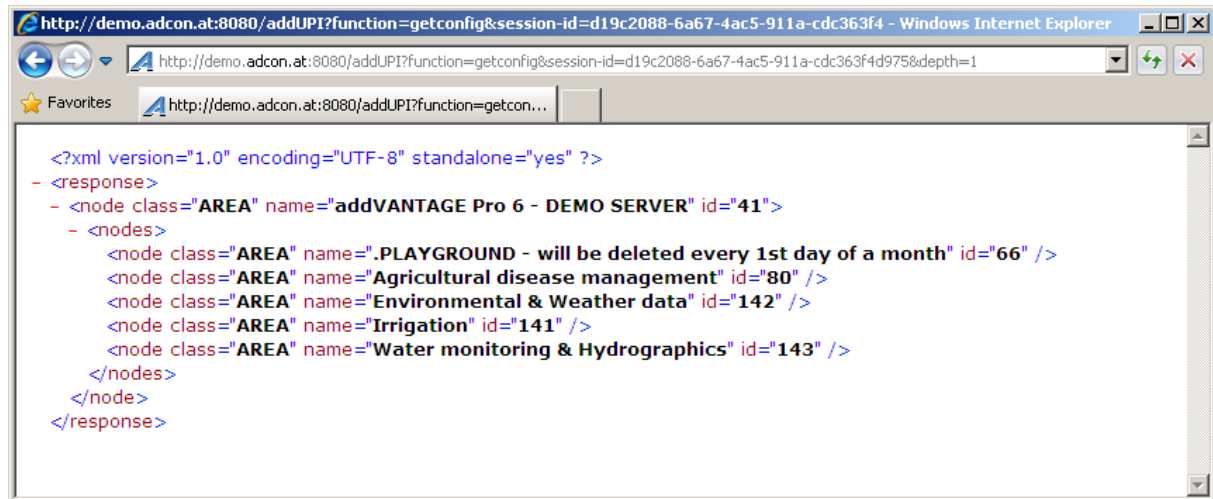
response:



This response is further used in the “session-id” parameter.

Most probably, the next step will be to get the node tree of this server. You can pass a node ID as parameter and also the depth, this means how many levels you would like to have returned. As default, you get all nodes on this server, thus we use depth=1 to have a better overview. Note that this time you need to pass the session-id parameter that you obtained when logging in:

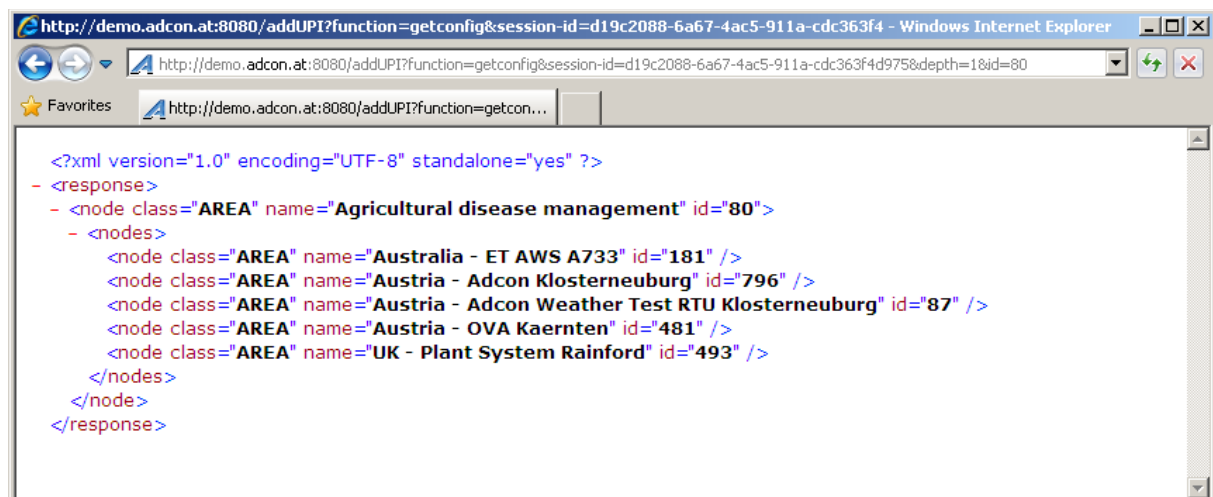
<http://demo.adcon.at:8080/addUPI?function=getconfig&session-id=...&depth=1>



```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <response>
- <node class="AREA" name="addVANTAGE Pro 6 - DEMO SERVER" id="41">
- <nodes>
  <node class="AREA" name=".PLAYGROUND - will be deleted every 1st day of a month" id="66" />
  <node class="AREA" name="Agricultural disease management" id="80" />
  <node class="AREA" name="Environmental & Weather data" id="142" />
  <node class="AREA" name="Irrigation" id="141" />
  <node class="AREA" name="Water monitoring & Hydrographics" id="143" />
</nodes>
</node>
</response>
```

As you can see, the local ID of the “Agricultural disease management” node is 80. You can use this ID to further explore the demo server:

<http://demo.adcon.at:8080/addUPI?function=getconfig&session-id=...&depth=1&id=80>



```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <response>
- <node class="AREA" name="Agricultural disease management" id="80">
- <nodes>
  <node class="AREA" name="Australia - ET AWS A733" id="181" />
  <node class="AREA" name="Austria - Adcon Klosterneuburg" id="796" />
  <node class="AREA" name="Austria - Adcon Weather Test RTU Klosterneuburg" id="87" />
  <node class="AREA" name="Austria - OVA Kaernten" id="481" />
  <node class="AREA" name="UK - Plant System Rainford" id="493" />
</nodes>
</node>
</response>
```

Now we'd like to display all the nodes in the "Adcon Klosterneuburg" area:

<http://demo.adcon.at:8080/addUPI?function=getconfig&session-id=...&id=796>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <response>
- <node class="AREA" name="Austria - Adcon Klosterneuburg" id="796">
- <nodes>
+ <node class="AREA" name="Apfel 2010" id="2799">
+ <node class="AREA" name="Apfel 2011" id="2800">
+ <node class="AREA" name="Wein 2010" id="2802">
+ <node class="AREA" name="Wein 2011" id="2801">
- <node subclass="DEFAULT" class="DEVICE" name="Adcon Klosterneuburg" template="A850_A733_V4" id="797">
- <nodes>
<node subclass="BATT" class="TAG" name="Batt Adcon KlbG" template="A850_ANALOG_TAG_V2" id="800" />
<node subclass="DIGITAL" class="TAG" name="Digital A" template="A850_DIGITAL_TAG_A733_V2" id="807" />
<node subclass="RF" class="TAG" name="Incoming RF Level" template="A850_ANALOG_TAG_V2" id="809" />
<node subclass="LW" class="TAG" name="Leaf Wetness" template="A850_ANALOG_TAG_A733_V2" id="804" />
<node subclass="RF" class="TAG" name="Outgoing RF Level" template="A850_ANALOG_TAG_V2" id="798" />
<node subclass="RAIN" class="TAG" name="Precipitation 0.1mm lmbt" template="A850_ANALOG_TAG_V2" id="801" />
<node subclass="GR" class="TAG" name="PyranometerCM3" template="A850_ANALOG_TAG_A733_V2" id="803" />
<node subclass="RAIN" class="TAG" name="rain B" template="A850_ANALOG_TAG_V2" id="810" />
<node subclass="RAIN" class="TAG" name="rain D" template="A850_ANALOG_TAG_V2" id="814" />
<node subclass="RAIN" class="TAG" name="RAIN_01mm 1" template="A850_ANALOG_TAG_V2" id="808" />
<node subclass="RH" class="TAG" name="Relative Humidity" template="A850_ANALOG_TAG_A733_V2" id="811" />
<node subclass="SOLARCELL" class="TAG" name="Solar Cell" template="A850_ANALOG_TAG_V2" id="806" />
<node subclass="TEMP" class="TAG" name="Temperature" template="A850_ANALOG_TAG_A733_V2" id="802" />
<node subclass="WINDDIR" class="TAG" name="Wind Direction" template="A850_ANALOG_TAG_A733_V2" id="812" />
<node subclass="WINDSPEED" class="TAG" name="wind max value slot c1" template="A850_ANALOG_TAG_A733_V2" id="805" />
<node subclass="windmax" class="TAG" name="wind max value slot c3" template="ANALOG_TAG_A7X3_V1_4" id="799" />
<node subclass="WINDSPEED" class="TAG" name="Wind Speed" template="A850_ANALOG_TAG_A733_V2" id="813" />
</nodes>
</node>
+ <node subclass="eto" class="EXTENSION" name="ETo" template="" id="3084">
+ <node subclass="statistic" class="EXTENSION" name="Statistic - Rain" template="" id="2790">
+ <node subclass="statistic" class="EXTENSION" name="Statistic - RH" template="" id="2786">
+ <node subclass="statistic" class="EXTENSION" name="Statistic - Temp" template="" id="2782">
</nodes>
</node>
</response>
```

The interesting part of this response is the list of tags. A tag is a node that contains measuring values (either physically measured or calculated).

Thus we'd like to get measuring values from the temperature tag, using the getdata function:

<http://demo.adcon.at:8080/addUPI?function=getdata&session-id=...&id=802>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <response>
- <node id="802">
<v s="0" t="20110810T11:50:00">17.63126</v>
</node>
</response>
```

Note that the time stamp is in ISO 8601 format, in its local timezone (Vienna) and the measuring value is in the tag's engineering unit (here: °C). The addUPI client must take care to convert the timestamps and values to the necessary format.

You also can obtain multiple values at once by passing the begin date and the number of values (slots) to be returned:

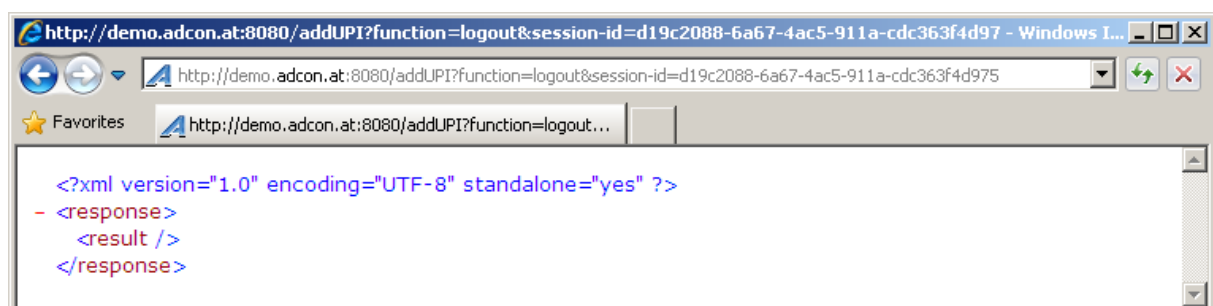
[http://demo.adcon.at:8080/addUPI?function=getdata&session-id=...&
id=802&date=20110101T00:00:00&slots=1000](http://demo.adcon.at:8080/addUPI?function=getdata&session-id=...&id=802&date=20110101T00:00:00&slots=1000)



The first timestamp is showing the time in ISO8601 format, the subsequent time stamps are an offset to the previous ones. If you'd like to have the timestamps in other formats, check out the additional parameters in the getdata chapter.

And finally, NEVER forget to logout when you've finished. Else, the session will take up resources on the server but also reduce the number of parallel logged-in users by 1.

<http://demo.adcon.at:8080/addUPI?function=logout&session-id=...>



4. addUPI Reference

4.1 General Format of a Request

A request formulated by a client has a generic form as given below

```
http://hostname:port/addUPI?function=fn&session-id=...&param1=p1&...paramn=pn
```

where

- hostname is the address of the server
- port is the port number to the server listens upon (if not specified, it is standard 80; it is recommended to use the port 80 for Telemetry Gateways and 8080 for addVANTAGE servers)
- addUPI is the name of the handler on the server that will be invoked to handle the request; this may be a servlet, a cgi, etc.
- function is the name of the invoked method
- session-id is a text identifying a certain client (obtained after authentication)
- param<i>=pn are the parameters requested by a particular function

4.2 Response Format

The response will return an XML document. There are two main ways to code the result: in plain ASCII, or binary (ZIP compressed).

4.3 addUPI Functions

A list of currently defined addUPI functions follows. They are divided in four classes:

- authentication functions
- configuration functions
- data transfer functions
- node specific functions (custom).

The authentication, configuration and data transfer functions are mandatory, while the node specific functions differ from device to device and may or may not be implemented.

Each function call returns an XML document. The XSD (XML schema definition) for the responses are specified in the addUPI protocol. Each XSD defines two possible returns: one is the valid response (in case no error has occurred) and the other is the error response. The error response is defined as an integer/string pair. The integer is the error number and the string is the human-readable explanation of the error.

4.3.1 Authentication Functions

login

FORMAT	<code>addUPI?function=login&user=username&passwd=password&timeout=xxx&mode=t/z&version=1.2</code>
DESCRIPTION	Request authentication from server
PARAMETERS	<p><code>user</code> is the user to be authenticated (mandatory).</p> <p><code>passwd</code> is the password (mandatory).</p> <p><code>timeout</code> is the session duration (in seconds); if missing, the session is either valid until logout or a default timeout is used depending on the server implementation. See the remarks for additional information.</p> <p><code>mode</code> specifies how the response should be returned; it can be <code>t</code> (ASCII text) or <code>z</code> (compressed text) — if missing, <code>t</code> is implied.</p> <p><code>version</code> specifies the addUPI version to use for further communication (default: 1.0, supported values: 1.0 and 1.2)</p>
REMARKS	<p>The session-id string must be unique in the context of a given server; its generation could be for example based on the requesting client's address plus a serially incremented number and some random added part.</p> <p>The timeout starts counting after the last request, that is, the timeout counter will be reset with each valid request (invalid or malformed requests are not considered for resetting the timeout counter). In normal</p>

communication cases, a client will always execute a logout, so the timeout should never expire; it is however required in cases where the communication breaks up unexpectedly.

A server may (and it is desirable to) have its own timeout. In this case, and if the server built-in timeout is shorter than the timeout proposed by the client, then the server timeout will override the client timeout.

If the client specifies an addUPI protocol version that is not supported by the server, it may choose to return an error instead of a session ID. However, both the A840 and addVANTAGE versions 4.x and 5.x will ignore this parameter and just use protocol version 1.0

RESPONSE EXAMPLE

```
<response>
<result>
<string>34e04b88-1c4e-45da-8dd5-c1216048b162</string>
</result>
</response>
```

logout

FORMAT `addUPI?function=logout&session-id=nnnn&mode=t/z`

DESCRIPTION Request de-authentication from the server

PARAMETERS `session-id` is the result returned by the login function (mandatory).

`mode` specifies how the response should be returned; it can be t (ASCII text) or z (compressed text) — if missing, t is implied.

RESPONSE EXAMPLE

```
<response>
<result/>
</response>
```

4.3.2 Configuration functions

getConfig

FORMAT	<code>addUPI?function=getconfig&session-id=nnnn&id=nnnn&depth=m&df=date-format&flags=[a][f]&mode=t/z</code>
DESCRIPTION	Returns the configuration of a node and of its underlying nodes, depending on the specified depth.
PARAMETERS	<code>session-id</code> is the result returned by the login function (mandatory).

`id` is the node's ID — if missing, then the root tree will be returned.

`depth` signifies how many hierarchy levels should be returned.

The depth must be interpreted as follows: for a `depth=0`, only the specified node will be returned, but no details for the subordinate nodes. For a `depth=1`, the specified node plus its children nodes will be returned, and so on. If not specified, the whole tree will be returned.

`df` is the date format. It can be `iso8601` or `time_t`. If a certain date format is specified, then the answer must be returned using that format. If missing, `iso8601` is implied. The `time_t` format represents the number of seconds elapsed since 1 Jan 1970 00:00:00 relative to the UTC meridian. If the answer is known not to include date values, it can be omitted. See also "Date/Time Handling" on page 421.

`flags` specifies what information should be returned together with the node. Following flags are currently defined: `a` (attributes) and `f` (functions). If missing, only the node(s) will be returned. Obviously, any combination of flags may be used, from none to all.

`mode` specifies how the response should be returned; it can be `t` (ASCII text) or `z` (compressed text) — if missing, `t` is implied.

RESPONSE EXAMPLE

```
<response>
```

```

<node class="AREA" name="Austria - Adcon Klosterneuburg" id="796">
  <nodes>
    <node class="AREA" name="Apfel 2010" id="2799"/>
    <node class="AREA" name="Apfel 2011" id="2800"/>
    <node class="AREA" name="Wein 2010" id="2802"/>
    <node class="AREA" name="Wein 2011" id="2801"/>
    <node subclass="DEFAULT" class="DEVICE" name="Adcon Klosterneuburg" template="A850_A733_V4"
id="797"/>
    <node subclass="eto" class="EXTENSION" name="ETo" template="" id="3084"/>
    <node subclass="statistic" class="EXTENSION" name="Statistic - Rain" template="" id="2790"/>
    <node subclass="statistic" class="EXTENSION" name="Statistic - RH" template="" id="2786"/>
    <node subclass="statistic" class="EXTENSION" name="Statistic - Temp" template="" id="2782"/>
  </nodes>
</node>
</response>

```

gettemplate

FORMAT	addUPI?function=gettemplate&session-id=...&template=t&mode=t
DESCRIPTION	Returns the specified template.
PARAMETERS	<p>session-id is the result returned by the login function (mandatory).</p> <p>template points to a specific template — if not specified, all templates known by the inquired server will be returned.</p> <p>mode specifies how the response should be returned; it can be t (ASCII text) or z (compressed text) — if missing, t is implied.</p>
RESPONSE EXAMPLE <pre> <response> <template name="A850_A733_V4"> <attrs> <attrib perm="ro" type="int" name="id"/> <attrib perm="ro" type="string" name="template"> <string>A850_A733_V4</string> </attrib> <attrib perm="ro" type="string" name="name"/> <attrib perm="ro" type="string" name="class"> <string>DEVICE</string> </attrib> <attrib perm="ro" type="string" name="subclass"/> <attrib perm="ro" type="string" name="manufacturer"> <string>Adcon Telemetry</string> </attrib> <attrib perm="ro" type="string" name="type"/> <attrib perm="ro" type="string" name="version"/> ... </pre>	

getattrib

FORMAT	addUPI?function=getattrib&session-id=nnnn&id=nnnn& attrib=name&df=date-format&cache=y/n&mode=t/z
DESCRIPTION	Returns the value of the specified attribute of the specified node.
PARAMETERS	<p><code>session-id</code> is the result returned by the login function (mandatory).</p> <p><code>id</code> is the node's ID—mandatory.</p> <p><code>attrib</code> is the attribute's name—if not specified, all attributes of the node will be returned.</p> <p><code>df</code> is the date format. It can be <code>iso8601</code> or <code>time_t</code>. If missing, <code>iso8601</code> is implied. See also “Date/Time Handling” on page 421.</p> <p><code>mode</code> specifies how the response should be returned; it can be <code>t</code> (ASCII text) or <code>z</code> (compressed text) — if missing, <code>t</code> is implied.</p>
RESPONSE EXAMPLE <pre> <response> <attrib name="active"> <boolean>true</boolean> </attrib> <attrib name="altitude"> <double>177.0</double> </attrib> <attrib name="batteryVoltage"> <double>6.90196</double> </attrib> <attrib name="code"> <int>11142</int> </attrib> ... </pre>	

4.3.3 Data transfer functions

getdata

FORMAT	addUPI?function=getdata&session-id=nnnn&id=nnnn& df=date-format&date=yyyymmddThh:mm:ss&slots=s&cache=y/n& mode=t/z
DESCRIPTION	Returns data from the specified node(s).
PARAMETERS	<p><code>session-id</code> is the result returned by the login function (mandatory).</p> <p><code>id</code> is the nodes' ID — if missing, an error (5 — node not found) will be returned. If the specified node has children, then all the data down the tree until the last node will be returned. The node must be an RTU or sensor (getdata is not supported for other node types, like areas or the server itself).</p> <p><code>df</code> is the date format. It can be iso8601 or time_t. If missing, iso8601 is implied. See also "Date/Time Handling" on page 421.</p> <p><code>date</code> specifies the date of the newest stored slot on the client — the data returned will be strictly newer than this date. If missing, the last stored slot will be returned (see also the remarks section for more details).</p> <p><code>slots</code> is the number of slots the client is prepared to accept (per node). The server may return less slots than requested, but never more — if missing, default one slot is implied. If a server returns less slots for a node than requested by the client, then the client assumes that there are no more data available for that node. The maximum number of slots requested by a client must not be greater than the value of the getdataMaxSlots attribute of the server (the root node in a getconfig response). If a client requests more slots than specified by the getdataMaxSlots attribute, the server will return an error.</p>

	<p>mode specifies how the response should be returned; it can be t (ASCII text) or z (compressed text)—if missing, t is implied.</p>
REMARKS ON RETURNED XML	<p>The date stamp should be interpreted by a server as follows:</p> <ul style="list-style-type: none"> ⌚ If the requested date is older than the oldest entry on the server, then a number of slots starting with the oldest stored slot will be returned; ⌚ If the slot requested has a date stamp newer than the newest entry on the server, then an error will be returned (14 — no more data); ⌚ If data is available, then the slot with the date stamp strictly newer than (but not equal to) the requested date stamp will be returned; as a general rule, a client should always request the next slot with the date stamp retrieved from the last slot it received in a previous request. Example (simplified): <pre>getdata(20010101T00:15:30)</pre> <p>returns the slot with the date stamp</p> <pre>20010101T00:19:55</pre> <p>next request will be</p> <pre>getdata(20010101T00:19:55)</pre> <p>and the response will be</p> <pre>20010101T00:24:20</pre> <p>and so on.</p> <p>The “offset” of a measurement value is the time span between the sample time and the end of the slot interval, the “duration” is the time span between the start and the end of the slot (for example, a wind maximum at 8:14 for a slot started at 8:00 and stored at 8:15 would result in a timestamp of 8:15, a duration of 900 seconds and an offset of 60 seconds). Note that offset and duration are only returned when the client specified to use addUPI version 1.2 in the login functioncall and the addUPI server supports this protocol version.</p> <p>Additionally, these slot elements are not supported by all tag types.</p>

If the duration and/or the offset is unknown or not supported, it is disallowed to send the duration/offset in the response. There is no special placeholder for unknown duration/offset, especially “0” is not meant as placeholder! A duration of 0 means a punctual measurement, like “what is the current state?”.

If the specified node has children (only works for RTUs – see above), then the server will return data from all the children recursively, having the time stamps newer, but not equal to the specified time stamp in the request. For nodes that can’t provide it, an error will be returned (14 — no more data).

The states for slots are: -99 to -1 for partial data (value is percentage of samples missing, i.e. -23 means that 23 percent of the samples were missing and 77 percent of the samples were taken), 0 for OK (100% of samples were taken), 1 for invalid data and 2 for missing data (0% of samples were taken). Note that slot states other than 0,1 and 2 are only returned when the client specified to use addUPI version 1.2 in the login functioncall.

RESPONSE EXAMPLE

```
<response>
<node id="802">
  <v s="0" t="20110905T13:50:00">32.40537</v>
</node>
</response>
```

4.3.4 Node specific functions

All node specific functions are called by the generic function call described below:

functioncall

FORMAT	<code>addUPI?function=functioncall&session-id=nnnn&id=nnnn&df=date-format&name=func-name&paramName1=p1&...&mode=t/z</code>
PARAMETERS	<p><code>session-id</code> is the result returned by the login function (mandatory).</p> <p><code>id</code> is the node's or tag's ID (mandatory).</p> <p><code>df</code> is the date format. It can be <code>iso8601</code> or <code>time_t</code>. If missing, <code>iso8601</code> is implied. See also "Date/Time Handling" on page 421.</p> <p><code>name</code> is the name of the function to be invoked (mandatory).</p> <p><code>paramName1...</code> are the parameters (parameter name and type depend on the function). The parameter names are defined in the template.</p> <p><code>mode</code> specifies how the response should be returned; it can be <code>t</code> (ASCII text) or <code>z</code> (compressed text) — if missing, <code>t</code> is implied.</p>
REMARKS	<p>A special type of function call is the switching of a valve that is connected as a DIGITAL tag. By using the <code>monostableOff</code> and the <code>writeValue</code> function, you can open a valve for a certain time:</p> <pre>addUPI?function=functioncall&session-id=...&id=123&name=monostableOff&onTime=xxx</pre> <p>switches the digital tag with node ID 123 for xxx seconds. xxx must be smaller than 65536 (approx. 18 hours)</p> <pre>addUPI?function=functioncall&session-id=...&id=123&name=writeValue&newValue=true</pre> <p>switches the digital tag with node ID 123 on. Note that the digital tag's status can be reset when the RTUs resets.</p>

4.4 Data types

4.4.1 Standard data types

Following data types are defined in addUPI:

Data type	Description	Example
int	four-byte signed integer	<code><int>-12</int></code>
boolean	false or true	<code><boolean>true</boolean></code>
string	ASCII string	<code><string>hello world</string></code>
double	double-precision signed floating point number	<code><double>-12.214</double></code>
date	date/time	<code><date>20000714T12:05:33</date></code>
base64	base64-encoded binary	<code><base64>eW9lIGNhbid0IHJlYWQgdGhpcyE= </base64></code>
array	a sequence of objects	<code><array></code> <code><int>23</int></code> <code><double>15.23</double></code> <code><string>test</string></code> <code></array></code>
struct	a collection of key-value pairs	<code><struct></code> <code><member name="an integer"></code> <code><int>2345</int></code> <code></member></code> <code><member name="a double"></code> <code><double>6.3</double></code> <code></member></code> <code><member name="a string"></code> <code><string>baburiba</string></code> <code></member></code> <code></struct></code>

4.4.2 Enumeration data types

Since there is no special enumeration data type, the type string is used instead.

samplingMethod

The sampling method specified in tags can be one of:

Value	Meaning
never	This tag does not deliver data at all
event	This tag does not deliver values in a defined slot interval, but on occurrence of tag-specific events (for example, a digital port sensor might store data on port change events). Since this is a single sample value, no calculation takes place.
minimum	The minimum of the samples within a slot
maximum	The maximum of the samples within a slot
sum	The sum of the samples within a slot
arithmetic average	The arithmetic average of the samples within a slot
circular average	The circular average of the samples within a slot (for example, a wind direction or compass heading)
vectorial average	One component of a vectorial average of the samples within a slot. The vector itself is defined by two components: the length (for example, using a wind speed sensor) and the direction (for example, using a wind direction sensor).
first sample	The first sample within a slot
last sample	The last sample within a slot
unknown	The sampling method is not known by the RTU and/or the gateway (for example, for sensors on the A740 or on an SDI bus)

4.5 Date/Time Handling

Date and time can be specified in ISO 8601 or UTC format by means of the `df` parameter.

- if no parameter or `df=iso8601` is specified, the time is given and displayed in local time, using the timezone of the respective node.

Note: some types of addUPI servers (for example, the A840 and A850 gateways) always use the timezone of the gateway instead of the timezone of the node, which may cause problems if these differ! Thus use this setting only for debugging purposes!

- if `df=time_t` is specified, the time is given and displayed in seconds elapsed since 1 Jan, 1970, 0:0:0 UTC.

The advantage of the `time_t` timestamp over the human-readable `iso8601` format is that the `time_t` format avoids problems associated with time zones and daily saving time (DST) conversions. Thus, it is recommended to always use the `df=time_t` parameter (especially for machine-to-machine communications).

As an additional option, addUPI servers may provide extensions for finer control of the output of date/time specifiers:

- the `df` (date format) parameter may be suffixed with “,incremental” or “,absolute” to specify whether subsequent date/time specifiers are given as number of seconds after the previous one (for example, `df=time_t,incremental`) or whether all date/time values are fully specified (`df=time_t,absolute`). As default, incremental timestamps are printed.

Note: whether these extensions can be used depend on the type and software version of the addUPI server. Currently, these extensions are only supported by the A850 gateway with firmware version 1.5.0 or newer and only used for the answers to a `getdata` addUPI request.

4.6 Templates

With addUPI it is possible to retrieve a node's description in its entirety. That means a client can enumerate the attributes and the functions associated with a node.

The following information is provided for each attribute:

- name: the internal name by which an attribute is referenced (required);
- display name: the attribute's human-readable name (optional);
- type: the type of the attribute value (string, int, etc.—required);
- description: a short description of the attribute (optional);
- permissions: specifies if an attribute is read-only, read-write or write-only (required);
- visible: this flag is intended to be used by a client that presents the attribute list to the user and wants to filter some attributes (optional).

The following information is provided for each function:

- name: the internal name by which a function is referenced (required);
- description: a short description of the function, i.e. what it is intended to do (optional);
- the parameter list; for each parameter, the following is specified:
 - a. name: the internal name by which a parameter is referenced (required);
 - b. type: the value type of the parameter (required);
 - c. description: a short parameter description (optional)
- the return value; the following information is provided about the return value:
 - a. type: the return value type (required);
 - b. description: a short description of the return value (optional).

Note: in older addUPI implementations (specifications), you may find the “events” or “notifications” as part of the node description. These were removed in order to make addUPI easier to understand and implement.

Such a detailed description of a node is required because a client can be a visual tool that can explore and discover an addVANTAGE network at runtime, thus giving the possibility to an user to dynamically get/set attributes or invoke functions.

From the description above it is clear that a lot of information is required in order to define a node. Because a lot of nodes in an addVANTAGE network are similar (i.e. all the A733 devices share the same definition, only some attribute values are different), and in order to reduce the amount of data flowing over the network, addUPI makes use of templates. A template represents the general description of a node (attribute definitions, function definitions) and optionally, some of the nodes' default attribute values.

The node's template is an XML String attribute for each node. It gives the expected structure of the node's attributes/functions and partially their default values. The node itself holds a list of functions and attributes, which overwrite the defaults defined in the template.

5. XSD (XML Schema Definition)

This section presents the XSD for all possible responses supported by addUPI. It can be used for validation or even code generators. You can use for example JAXB to generate Java code. We didn't split it up into multiple XSDs because a lot of XSD references are used by different requests making the XSDs really long and hard to read. Moreover, note that this XSD shows more elements than supported by this specification. These are elements used in older addUPI specifications – in order to have downwards compatibility, we need to list them here, although they are no longer used by newer Adcon devices.

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

  <xsd:element name="response">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref="template" minOccurs="0" maxOccurs="unbounded"/>
        <!-- only can getConfig on one node and only can getData on 1 Tag -->
        <xsd:element ref="node" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="result" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="attrib" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="secresponse" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="error"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="result">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref="boolean"/>
        <xsd:element ref="int"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

        <xsd:element ref="string"/>
        <xsd:element ref="double"/>
        <xsd:element ref="date"/>
        <xsd:element ref="base64"/>
        <xsd:element ref="array"/>
        <xsd:element ref="struct"/>
    </xsd:choice>
</xsd:complexType>
</xsd:element>

<xsd:element name="error">
    <xsd:complexType>
        <xsd:attribute name="code" type="xsd:string" use="required"/>
        <xsd:attribute name="msg" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="boolean" type="xsd:boolean">
</xsd:element>

<xsd:element name="int" type="xsd:integer">
</xsd:element>

<xsd:element name="string" type="xsd:string">
</xsd:element>

<xsd:element name="double" type="xsd:double">
</xsd:element>

<xsd:element name="date" type="xsd:string">
</xsd:element>

<xsd:element name="base64" type="xsd:base64Binary">
</xsd:element>

<xsd:element name="array">
    <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="boolean"/>
            <xsd:element ref="int"/>
            <xsd:element ref="string"/>
            <xsd:element ref="double"/>
            <xsd:element ref="date"/>
            <xsd:element ref="base64"/>
            <xsd:element ref="array"/>
            <xsd:element ref="struct"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

<xsd:element name="struct">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="member" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="member">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="boolean"/>
            <xsd:element ref="int"/>
            <xsd:element ref="string"/>
            <xsd:element ref="double"/>
            <xsd:element ref="date"/>
            <xsd:element ref="base64"/>
            <xsd:element ref="array"/>
            <xsd:element ref="struct"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

```



```

    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="node">
  <xsd:complexType>
    <xsd:choice>
      <xsd:choice>
        <xsd:element ref="v" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="error"/>
      </xsd:choice>
      <xsd:sequence>
        <xsd:element ref="attrs" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="functions" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="events" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="nodes" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:choice>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="template" type="xsd:string" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="class" type="xsd:string" use="required"/>
    <xsd:attribute name="subclass" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="attrs">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="attrib" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="attrib">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="attribdef" minOccurs="0" maxOccurs="1"/>
      <xsd:choice>
        <xsd:element ref="boolean"/>
        <xsd:element ref="int"/>
        <xsd:element ref="string"/>
        <xsd:element ref="double"/>
        <xsd:element ref="date"/>
        <xsd:element ref="base64"/>
        <xsd:element ref="array"/>
        <xsd:element ref="struct"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="attribdef">
  <xsd:complexType>
    <xsd:attribute name="dispname" type="xsd:string" use="optional"/>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="boolean"/>
          <xsd:enumeration value="int"/>
          <xsd:enumeration value="string"/>
          <xsd:enumeration value="double"/>
          <xsd:enumeration value="date"/>
          <xsd:enumeration value="base64"/>
          <xsd:enumeration value="array"/>
          <xsd:enumeration value="struct"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:attribute name="descr" type="xsd:string" use="optional"/>
<xsd:attribute name="perm" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ro"/>
      <xsd:enumeration value="rw"/>
      <xsd:enumeration value="wo"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="visible" type="xsd:string" use="optional"/>
</xsd:complexType>
</xsd:element>

<xsd:element name="nodes">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="node" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="functions">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="function" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="function">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="params"/>
      <xsd:element ref="return"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="descr" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="params">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="param" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="param">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="boolean"/>
          <xsd:enumeration value="int"/>
          <xsd:enumeration value="string"/>
          <xsd:enumeration value="double"/>
          <xsd:enumeration value="date"/>
          <xsd:enumeration value="base64"/>
          <xsd:enumeration value="array"/>
          <xsd:enumeration value="struct"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="descr" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="return">
  <xsd:complexType>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="boolean"/>
          <xsd:enumeration value="int"/>
          <xsd:enumeration value="string"/>
          <xsd:enumeration value="double"/>
          <xsd:enumeration value="date"/>
          <xsd:enumeration value="base64"/>
          <xsd:enumeration value="array"/>
          <xsd:enumeration value="struct"/>
          <xsd:enumeration value="void"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="descr" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="events">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="event" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="event">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="descr" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="template">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="attribs" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="attrib" minOccurs="0"
              maxOccurs="unbounded">
              <xsd:complexType>
                <!-- default value -->
                <xsd:choice minOccurs="0"
                  maxOccurs="1">
                  <xsd:element ref="boolean" />
                  <xsd:element ref="int" />
                  <xsd:element ref="string" />
                  <xsd:element ref="double" />
                  <xsd:element ref="date" />
                  <xsd:element ref="base64" />
                  <xsd:element ref="array" />
                  <xsd:element ref="struct" />
                </xsd:choice>
                <xsd:attribute name="name"
                  type="xsd:string" use="required" />
                <xsd:attribute name="dispname"
                  type="xsd:string" use="optional" />
                <xsd:attribute name="type"
                  use="required">
                  <xsd:simpleType>
                    <xsd:restriction
                      base="xsd:string">
                        <xsd:enumeration
                          value="boolean" />

```

```

        <xsd:enumeration
            value="int" />
        <xsd:enumeration
            value="string" />
        <xsd:enumeration
            value="double" />
        <xsd:enumeration
            value="date" />
        <xsd:enumeration
            value="base64" />
        <xsd:enumeration
            value="array" />
        <xsd:enumeration
            value="struct" />
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="descr"
    type="xsd:string" use="optional" />
<xsd:attribute name="perm"
    use="required">
    <xsd:simpleType>
        <xsd:restriction
            base="xsd:string">
            <xsd:enumeration
                value="ro" />
            <xsd:enumeration
                value="rw" />
            <xsd:enumeration
                value="wo" />
        </xsd:restriction>
    </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="visible"
        type="xsd:string" use="optional" />
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<!-- functions and events won't be used - they are only returned with the flags f
or e, which are not used -->
    <xsd:element ref="functions" minOccurs="0"
        maxOccurs="1" />
    <xsd:element ref="events" minOccurs="0"
        maxOccurs="1" />
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"
    use="required" />
</xsd:complexType>
</xsd:element>

<xsd:element name="v">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="t" type="xsd:string" use="required"/>
                <xsd:attribute name="s" type="xsd:integer" use="required"/>
                <xsd:attribute name="d" type="xsd:long" use="optional"/>
                <xsd:attribute name="o" type="xsd:long" use="optional"/>
                <xsd:attribute name="type" type="xsd:integer" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>

<xsd:element name="secresponse">
    <xsd:complexType>
        <xsd:sequence>

```

```
<xsd:element ref="string"/>
</xsd:sequence>
<xsd:attribute name="secalg" type="xsd:string" use="required"/>
<xsd:attribute name="keyindex" type="xsd:integer" use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```