IControlTM Services Gateway Version 1.14
226-05M00-200

Programming Reference (API)

of the

iControl™ Services Gateway

Table of Contents

1. Document Scope

The document's first mission is to present a comprehensive view of the functioning of an iControl™ Gateway that manages the access to Miranda Services registered to it, for the benefit of an application programmer having to implement a Miranda Service' control application. The technologies used to achieve this mission are briefly identified. The key concepts required to understand the operation of an iControl™ Gateway and the implementation of application controlling Miranda Services through an iControl™ Gateway are introduced and described.

The document's second mission is to concisely specify the activities and sequences an application is required to handle in order to identify, assess and control Miranda Services through an iControl ™ Gateway.

The document's third mission is to provide through case studies concrete examples of control scenarios of Miranda Services registered to an iControl™ Gateway, using the iControl ™ Gateway XML API, and the sequences described in this document.

2. Reference Documents, Acronyms and Conventions

2.1 CONVENTIONS

In this document:

- The character '?' identifies a white space.
- Text within braces [ ] identifies a text value whose content may vary.
- Text in *italic* identifies an XML stream issued by an application to a Miranda Service Node.
- Text in ***bold italic*** identifies an XML stream returned by a Miranda Service Node.
- *{…}* identifies a time delay.
- The character '|' within XML tags identifies a choice of valid values.
- The acronym **TBD** identifies information To Be Defined in a later version of this document.
- The acronym **TBC** identifies information To Be Confirmed in a later version of this document.

3. Getting Started

3.1 XML™ USAGE RATIONAL

Miranda has chosen XML™ as the command language for the remote control its devices to offer programmers:

- Easy understanding of each Service capability.
- Seamless evolution of Services capabilities.
- Platform independence.
- Open Source Licensing.

Refer to Appendix A for more rationales and references.

3.2 iCONTROL™ GATEWAY ENTITIES

To program an application that controls Services through an iControl™ Gateway, a programmer shall understand the meaning of, and affect the following entities:

- iControl™ Directory Services
- iControl™ Services Gateway
- Registered Services List
- Service Node
- Service Identifier
- Service Access Key
- Access Key Parameters List

3.2.1 iCONTROL™ DIRECTORY SERVICES

A Miranda Service, when it starts up, register itself to a listening iControl™ Gateway through a portal referred to as the iControl™ Directory Services.

A Miranda Service, by default, attempts to register itself to all listening iControl™ Gateways. The Service Registration process is handled by the iControl™ Gateway, and requires no interventions whatsoever from any user or application.

An application shall, in order to identify and locate Miranda Services through an iControl™ Gateway, query the iControl™ Directory Services for the list of the currently registered Services.

An application will use the information returned by the iControl™ Directory Services to identify, locate, assess, and control the Miranda Services.

An application will access the iControl™ Directory Services to query the list of Registered Services within an iControl™ Gateway by establishing a TCP/IP-XML session through socket port 10001 of this iControl™ Gateway' Host.

3.2.2 iCONTROL™ SERVICES GATEWAY

Once an application has obtained the list of currently registered Miranda Services, it can use it to identify, locate, assess, and control the Miranda Services currently registered.

An application shall, in order to control Miranda Services through an iControl™ Gateway, establish a TCP/IP-XML session to the iControl™ Services Gateway through socket port 13000 of the iControl™ Gateway Host.

Using that connection and the Registered Services List, an application will be able to dialog with and control the registered Miranda Services through XML format commands and queries.

3.2.3 REGISTERED SERVICES LIST

Cornerstone of iControl™ operation is the Registered Services List.

An application, to operate Miranda Services through an iControl™ Gateway shall obtain it, update it, and manage it.

Refer to section ***iControl™ Directory Services*** on the mechanisms to establish a connection with the iControl™ Directory Services.

An application shall send the XML stream *<listNodes/>\r* to query an iControl™ Directory Services the Registered Services List.

The listening iControl™ Directory Services will return the following XML stream (where '*?*' identifies a white space):

<listNodesIs>[Service Node 1 Attributes] [Service Node 2 Attributes]…[Service Node N Attributes]</listNodesIs>\r

3.2.4 SERVICE NODE

An iControl™ Directory Services fully specify a Miranda Service through a Service Node XML stream.

Following is an example of a Registered Service List returned to an application by an iControl™ Directory Services, containing only one SDM-712 id 93 Service Node, belonging to the logical group 'Incoming Feeds 1", with short Identifier 'Aerospatiale Unitee 1', Miranda Service Name SDM_712 leased until 10:35:00 of January 19 2003 (translated to relative count 5000), and located in slot '3' of frame '1' reachable through socket port 13000 of host 192.168.103.195.

<listNodeIs><id>Aerospatiale?Unitee?1</id><name>SDM_712</name><group>Incoming?Feeds?1</group><frame>SYMPHONIE?3</frame><slot>3</slot><address>192.168.103.195?13000</address><type>SDM_712i_93</type><expiration>5000</expiration><globalStatus>OK</globalStatus></listNodeIs >\r

One Service Node XML stream identifies the following Miranda Service attributes:

- Miranda Service (Unique to iControl™ Gateway) Short Identifier (XML tag (*<id>[Short Identifier String]</id>*). It is a user customizable label.

  Example: ***<id>192.168.103.195_0_3_1_3</id>***

- Miranda Service Name (XML tag *<name>[Miranda Service Name String]</name>*)it is an user label

  Example: ***<name>SDM_712</name>***

- Miranda Logical Group the Service belongs to (XML tag *<group>[Logical Group Name String]</group>*)

  Example: ***<group>Incoming?Feeds?1</group>***

- Service physical location within Miranda, frames, and slots (XML tags *<frame>[Frame Number String]</frame>* and *<slot>[Slot Number String]</slot>*)

  Example: ***<frame>SYMPHONIE?3</frame><slot>3</slot>***

- Service (TCP/IP) network location (XML tag *<address>[IP Address String]?[Port number]</address>*)

  Example: ***<address>192.168.103.195?13000</address>***

- Service Type (XML tag *<type>[Type String]</type>*)

  Example: ***<type>SDM_712i_93</type>***

- Service Expiration date (XML tag *<expiration>[Expiration Count]</expiration>*)

  Example: ***<expiration>5000</expiration>***

- Service Global Status (tag *<globalStatus>[Global Status String]</globalStatus>*)

  Example: ***<globalStatus>OK </globalStatus>***

3.2.5 SERVICE SHORT IDENTIFIER

Miranda Service Short Identifier is certified by the iControl™ Gateway to be unique within that gateway.

Within Miranda Services networks, the Short Identifier is usually constructed from the concatenation of the Service Node IP address, frame identifier, and slot identifier). The XML tag that specifies the Short Identifier is *<id>[Short Identifier String]</id>*.

Examples:

<id>192.168.103.195_0_3_1_3</id>
<id>AppServer_0_3_0_8</id>
<id>m2_1_3_0_1</id>
<id>Aerospatiale?Unitee?1</id>
<id>appserver55_abs_Densite_SLOT_2</id>

3.2.6 SERVICE ACCESS KEY

An application controls a Miranda Service Node by specifying an operation to be performed on a parameter of a Service attribute identified through a Service Access Key.
The following example shows how an application will get the current “vLuma” value of a Miranda Service Node that supports “vLuma”, and increment it of one (presetted) step.

*<getvLuma/>\r**{…}<vLuma>60?[Unit?String]</vLuma>\r***
*<setvLuma>INC?1</setvLuma>\r**<ack/>\r{…}<Luma>61?[Unit?String]</vLuma>\r***

3.2.7 ACCESS KEY PARAMETER INFO

The following example shows how an application will get from a Miranda Service Node the parameters of the “vLuma” Service Access Key identified as RANGE, INC (increment one step), increment STEP, and UNIT.

<getParameterInfo?key="vLuma"/>
<vLuma>0 </vLuma>
<parameterInfo name="Luminance" isActive="true" min="-800.0" max="800.0" step="1.0" fstep="25" nominal="0.0" unit=""/>

The first XML tag returned as response is the “vLuma” value, after which the parameterInfo tag is returned.

4. iControl™ API Specifications

4.1 API GENERIC SPECIFICATIONS

An application exchange information with a Service Node registered within an iControl™Gateway through XML formatted commands.
A Service Node confirms a command by replying to the issuing application the XML token ***<ack/>.*** Some commands will not return the confirmation message ***<ack/>***, but directly the reply message.
A Service Node rejects a syntactically incorrect command by replying to the issuing application the XML tag ***<nack/>.***

All XML messages sent to by iControl™Gateway or Service Node must finish with the character CR (\r)
All XML messages returned by iControl™Gateway or Service Node are finished with characters CR+LF (\r\n).

4.2 iCONTROL™ GATEWAY DIRECTORY SERVICES COMMANDS

In order for an application to obtain the list of iControl™ Service Nodes registered in an iControl™Gateway, the application shall establish a TCP/IP-XML session with the iControl™ Gateway Directory Services, over a TCP/IP socket on the port 10001 of the iControl™ Gateway.

The application can then send any of the following XML commands to the iControl™ Gateway Directory Services:

- *listNodes*
- *getGatewayVersion*
- *nodeCount*
- *getNode*
- *getNextNode*

The XML command must be finished with the \r (Carriage Return) character.**The XML response returned by** iControl™ Gateway Directory Services will be finished by the \r\n symbol

Example: <listNodes/>\r

4.2.1 COMMAND LISTNODES

Purpose:
Query an iControl™ Directory Services the list of Service Nodes registered to an iControl™ Gateway.

Pre-Requisites:

- IP address of the iControl™ Gateway host is known.
- TCP/IP socket port 10001 has been successfully opened on iControl™ Gateway host IP address.

XML Syntax:
<listNodes/>\r

Replies:

- ***{…}<listNodesIs>[Node 1 Attributes][Node 2 Attributes]…[Node N ttributes]</listNodesIs>\r\n***
- ***{…}<nack/>\r\n***

The format of *[Node 1 Attributes]* is *<id>[value]</id><[Attribute 1 Name]>* [Value]<[Attribute 2 Name]>[Value]… <[Attribute N Name]>[Value] where <id> identifies the Miranda Service Node Short Identifier.

The supported *[Node Attributes]* currently supported by the iControl™ Gateway version 1 are:

- *<name>[Miranda Service Name]</name>*
- *<group>[Miranda Services Logical Group]</group>*
- *<frame>[Miranda Frame hosting the Service Node]</frame>*
- *<slot>[Slot in Frame hosting the Service Node]</slot>*
- *<address>[IP address of the Service Node]?[Port Number]</address>*
- *<type>[Service Type]</type>*
- *<expiration>[Service Expiration Value]</expiration>*
- *<globalStatus>[Service Global Status]</globalStatus>*

Example of a reply message reporting one node:

<listNodesIs><id>m2_1_3_0_1</id><name>ASD-231i ID 81</name><group>Grp0</group> <frame>SYMPHONIE 0</frame><slot>1</slot><address>10.10.20.10 13000</address><type>ASD-231i_81</type><expiration>300</expiration><globalStatus>OK</globalStatus></listNodesIs>\r\n

4.2.2 COMMAND GETGATEWAYVERSION

Purpose:
Query an iControl™ Directory Services the identify its runtime version

Pre-Requisites:

- IP address of the iControl™ Gateway host is known.
- TCP/IP socket port 10001 has been successfully opened on iControl™ Gateway host IP address.

XML Syntax:
<getGatewayVersion/>\r

Replies:

- ***{…}<gatewayVersion version="x.xx" />\r\n***
- ***{…}<nack/>\r\n***

4.2.3 COMMAND *GETNODE*

Purpose:
Query an iControl™ Directory Services for a specific service.

Pre-Requisites:
- IP address of the iControl™ Gateway host is known.
- TCP/IP socket port 10001 has been successfully opened on iControl™ Gateway host IP address.
- We know the serviceId we are interested in.

XML Syntax:

<getNode><id>serviceId</id></getNode>\r

Replies:
- ***<getNodeIs>[Node Attributes]</getNodeIs>\r\n***
- ***{…}<nack/>\r\n***

Example:

<getNode><id>m2_1_3_0_1</id></getNode>
{…}<getNodeIs><id>m2_1_3_0_1</id><name>ASD-231i ID 81</name><group>Grp0</group><frame>SYMPHONIE 0</frame><slot>1</slot><address>10.10.20.10 13000</address><type>ASD-231i_81</type><expiration>300</expiration><globalStatus>OK</globalStatus></getNodeIs>

When no service attributes are found for a serviceID, the XML tag returns no content:
<getNodeIs></getNodeIs>

4.2.4 COMMAND NODECOUNT

Pre-Requisites:
- IP address of the iControl™ Gateway host is known.
- TCP/IP socket port 10001 has been successfully opened on iControl™ Gateway host IP address.

XML Syntax:
- *<nodeCount/>\r*

Replies:
- ***<nodeCountIs>[Node number]</nodeCountIs>\r\n***
- ***{…}<nack/>\r\n***

A Node number equal to 0 means that no service is registered.

4.2.5 COMMAND GETNEXTNODE

Purpose:
Query an iControl™ Directory Services without knowing the registred services.

Pre-Requisites:
- IP address of the iControl™ Gateway host is known.
- TCP/IP socket port 10001 has been successfully opened on iControl™ Gateway host IP address.

XML Syntax:
<getNextNode/>\r

Replies:
- ***<getNextNodeIs>[Node Attributes]</getNextNodeIs>***
- ***{…}<nack/>***

A repeated request *<getNextNode/>\r* will return the next node recorded in iControl™ Directory Services starting with the first one, until the message ***<getNextNodeIs></getNextNodeIs>\r\n*** is received, signaling that the end of the list was reached, after which the nodes will be again reported one after another, starting with the first one.
This command can be used combined with *<nodeCount/>\r* command to determine the number of registered services.

IMPORTANT NOTE: In a system services may dynamically be added and/or removed. iControl™Gateway is not notifying in an unsolicited way its clients about these changes.
An application connected to iControl™Gateway should periodically query the iControl™ Directory Services to update the status of registered services.

Example: there is only one node in Directory Service. First call will return the node , next call will return an empty tag content, signaling the end of the list.

<getNextNode/>\r
{…}<getNextNodeIs><id>m2_1_3_0_1</id><name>ASD-231i ID 81</name><group>Grp0</group><frame>SYMPHONIE 0</frame><slot>1</slot><address>10.10.20.10 13000</address><type>ASD-231i_81</type><expiration>300</expiration><globalStatus>OK</globalStatus></getNextNodeIs>\r\n
<getNextNode/>\r
{…}<getNextNodeIs></getNextNodeIs>\r\n

4.3 ESTABLISHING A SERVICE NODE SESSION THROUGH ICONTROL™ GATEWAY

Once an application has obtained the list of nodes registered to an iControl™Gateway, the application may establish a TCP/IP-XML session with the iControl™ Services Gateway.

First. the application shall open a TCP/IP socket on the port 13000 of the iControl™ Services Gateway.

Next, the application can then send any of the following XML commands to the iControl™ Services Gateway:

- *openID*
- *getParameterInfo*
- *get[ParameterName]*

- *set[ParameterName]*
- *closeID*

4.3.1 OPENING A SESSION WITH A SERVICE NODE (*OPENID*)

Purpose:
Open a session with a Miranda Service Node over an iControl™ Services Gateway (TCP/IP socket port 13000 of iControl™ Services Gateway).

Pre-Requisites:
- Short Identifier of the Node Service is known.
- IP address of the Service Node is known.
- TCP/IP socket port 13000 has been successfully opened at <*address*> of Service Node.

XML Syntax:
- *<openID>[Service Node Short Identifier]</openID>\r*

Replies:
- ***{…}<ack/>\r\n***
- ***{…}<nack/>\r\n***

  The <nack/> reply is returned when:

  - An openID command was already sent and accepted before.
  - The Service Node Short Identifier is incorrect or the service is not anymore registered in Directory services.
- ***{…}<nack exception="please open the id before"/>***

  After connection to port 13000, the first command to send is openID.

  This message is returned when another command was sent before the openID command, or a syntax error in XML command tags was detected.

Example:

<openID>m2_1_3_12_1</openID>\r

{…}<ack/>\r\n

4.3.2 QUERYING AN ACCESS KEY PARAMETER' FORMAT (*GETPARAMETERINFO*)

Purpose:
To obtain all the available information on parameters of an access key accessible to an application**.**

Pre-Requisites:
- Short Identifier of the Node Service is known.
- IP address of the Service Node is known.
- TCP/IP socket port 13000 has been successfully opened at <*address*> of Service Node.
- A session with the Miranda Service Node we want to query has been successfully opened with the command *<openID>[Service Node Short Identifier]</openID>*

XML Syntax:
<getParameterInfo?key="[Access Key]"/>\r where *[Access Key]* is the parameter's specific key.

All iControl's access keys must start with lower-case letter.

Replies:

- ***{…}<[param>[value]?[unit]</[param]>***
  <parameterInfo><key>[AccessKey]<type>[param]<type>[param]…<type><param><parameterInfo/>\r\n

  where each *[param]* is a Service Node Access Key parameter Name to be used within commands get *[param]* and set *[param]*. The first XML tag returned as response is the current parameter value, after which the parameterInfo tag is returned.

- ***{…}<nack/>\r\n***
  Returned when the command syntax is wrong.

Example:
Getting the information about the parameters of the access Key "vLuma", where the values of "vLuma" are float numbers and there is no associated measurement unit:

<getParameterInfo?key="vLuma"/>\r
<vLuma>0 </vLuma>
<parameterInfo name="Luminance" isActive="true" min="-800.0" max="800.0" step="1.0" step="5" nominal="0.0" unit=""/>\r\n

Returned information for the "vTest" parameter, where the values of "vTest" are a choice list:

<getParameterInfo key="vTestMode"/>\r
<vTestMode>OFF</vTestMode>
<parameterInfo name="Test Signal" type="choice" isActive="true"><choice label="OFF" rcpval="OFF" active="true"/><choice label="100% Bars" rcpval="100%" active="true"/><choice label="75% Bars" rcpval="75%" active="true"/><choice label="Black" rcpval="BLCK" active="true"/></parameterInfo>\r\n

where:

- ***type="choice"*** informs that the parameter is implemented as a choice list
- ***isActive*** informs if the parameter is enabled ("true") or disabled ("false"). An application should use this information to activate/deactivate the control of a parameter.

Each choice is defined by a tag like the following:
<choice label="100% Bars" rcpval="100%" active="true"/>

- ***choice label*** is the long value label displayed only in the IControl service panel;
- ***rcpval*** is the short value label displayed on RCP-100 panel and returned to any client using the Gateway to control a card;
- ***active="true"*** or ***"false"*** informs if the choice label is a valid choice. An application should use this information to activate/deactivate the use of this choice

The parameterInfo XML message is generally sent by service also as an unsolicited message, if the parameterInfo properties have changed. An application may interrogate the parameterInfo properties right before trying to change the parameter value or keep track of the changes reported by the unsolicited parameterInfo messages.

4.3.3 GETTING A PARAMETER VALUE (*GET[PARAMETER]*)

Purpose:
Obtain the values associated to a parameter of a Service Access Key.
Generally all iControl access keys start with lower-case letter. A get command for a specific parameter will be constructed using the next rule:

"<get"+ Service Access key with first letter in capital (lowercase is also accepted) + "/>"
Example: Service Access Key = "luma"
Get command = <getLuma/>\r

Pre-Requisites:

- Short Identifier of the Node Service is known.
- IP address of the Service Node is known.
- TCP/IP socket port 13000 has been successfully opened on Service Node IP address.
- A session with the Miranda Service Node we want to query has been successfully opened with the command *<openID>[Service Node Short Identifier]</openID>*

XML Syntax:
<get[param]/>

Replies:

- ***{...}*<[param]>[value]?[unit]</[param]>*r\n***
- ***{…}<nack/>\r\n***

(Ex. <setup>7.5 IRE</setup>)

Examples:
<getvLuma/>\r
{…}<vLuma/>60 </vLuma>\r\n

<getSetup/>\r
<setup>7.5 IRE</setup>\r\n

If the [param] passed to a get[param] request is not recognized by service (syntax error or not supported) no message is returned as response.

4.3.4 SETTING A PARAMETER TO A VALUE (*SET[PARAMETER]*)

Purpose:
To affect a parameter of a Service Node Access Key.

Pre-Requisites:

- Short Identifier of the Node Service is known.
- IP address of the Service Node is known.
- TCP/IP socket port 13000 has been successfully opened at *<address>* of Service Node..
- A session with the Miranda Service Node we want to control has been successfully opened with the command *<openID>[Service Node Short Identifier]</openID>*

XML Syntax:

<set[param]>[type]?[value]></set[param>\r

Replies:

- ***{…}<ack/>{…}<[param/][value]</param>\r\n***
- ***{…}<nack/>\r\n***

[param] is any of the parameter tags . Ex. “Luma”
[type] – possible valuesare: “inc” or “INC. ”dec” or “DEC”, ”set” or “SET”
[value] – depending of [type] can be 1 to n for “inc” and “dec” or the absolute value for “set”, represented as an integer or decimal number..

A set command for a specific parameter will be constructed using the next rule:

“<set”+ Service Access key with first letter in capital (lowercase is also accepted) + “>” + [type] + “?”+”value”+”</set”+ Access key with first letter in capital +“>

Example:
Service Access Key: “luma”
set command using the absolute value: “<setLuma>set 60</setLuma>\r
or set command using a relative value: “<setLuma>inc 2</setLuma>\r

For the numeric parameters, when *[type]* is “inc” or “dec” , the effect will be the increase/decrease of the current parameter value with a quantity equal to *{value]*step* , where step is given by the ***parameterInfo*** message. For example, in the case of a parameter with step = 0.5, “inc 3” will increase the current value with 1.5.

In case of choice parameters, inc/dec will set the next/previous valid choice in the choice list.

Example:
Setting the Luma of a Service Node that supports Luma.

*<setLuma>inc?1</setLuma>\r****{…}<ack/>\r\n{…}<luma>61?</luma>\r\n***
*<setLuma>dec?1</setLuma>\r****{...}<ack/>\r\n{...}<luma>60?</luma>\r\n***
*<setLuma>set?50.0</setLuma>\r****{...}<ack/>\r\n{...}<luma>50?</luma>\r\n***

If following the execution of the set command, the parameter value did not change, only ***<ack/>*** message is returned in response.

If [value] is out of range, the command will not be executed and only ***<ack/>*** message will be returned.

In the case of interdependent parameters, (like Luma being also part of a total gain parameter), a ***set*** command may also change the other related parameters. The Gateway will send in return also the unsolicited messages for the related modified parameters and/or ***parameterInfo*** messages if the infos for the related parameters also changed. The client application should be able to receive and process all these incoming messages.

A real case example illustrating the mentioned possible dependencies:
<setvLuma>set 51.0</setvLuma>\r
<ack/>\r\n
<vAllGain>178 </vAllGain>\r\n
<vAllGain>178 </vAllGain>\r\n

<parameterInfo name="All Gain" isActive="true" min="-673.0" max="736.0" step="1.0" fstep="25" nominal="31.0" unit=""/>\r\n
<vLuma>51 </vLuma>\r\n
<vYGain>51 </vYGain>\r\n

4.3.5 CLOSING A SESSION WITH A SERVICE NODE (*CLOSEID*)

Purpose:

Close a session with a Mirande Service Node over an iControl™ Services Gateway (TCP/IP socket port 13000 of iControl™ Services Gateway).

- **Pre-Requisites:**
- Short Identifier of the Node Service is known.
- IP address of the Service Node is known.
- TCP/IP socket port 13000 has been successfully opened on Service Node IP address.
- The session with the Miranda Service Node we want to close has been successfully opened with the command <*openID>[Service Node Short Identifier]</openID>*

XML Syntax:
The following two commands are accepted by the Gateway, both having the same effect:

<closeID>[Service Node Short Identifier]</closeID>\r
<closeID/>\r

Replies:

- ***{…}<nack/>\r\n***

The Gateway tests only if the command starts with the tags <closeID> or <closeID/> and returns <nack/> if not. Service Node Short Identifier is ignored.
No message is returned if the session was successfully closed

5. Case Studies

5.1 Muting Channels of All SDM712i_93 Services Registered Through an iControl™ Gateway, Every Fridays 00:00.

- The application obtains, lets say from a local database, the IP address of the iControl™ Gateway (199.202.148.220).

- The application knows the Access Key has only parameter mute12

- The application is configured to query for the Registered Services List.
  - The application opens a TCP/IP socket to port 10001 of iControl™ Gateway at IP address 199.202.148.220.
  - The application issues XML data stream: *<listNodes/>.*
  - The application waits for the *<listNodesIs>…</listNodesIs>* response.
  - The application closes the TCP/IP socket at port 10001 of IP address 199.202.148.220

- For each SDM712i Service Node the application identified within *<listNodesIs> … </listNodesIs>* by the tag *<name>* value matching the string 'SDM712i_93 ', the application:
  - Adds to the hash SDM712iHash{}an entry using hash key *<id>* with the value of the related *<address>* tag.
  - Adds to the hash SDM712iLocationHash{} an entry using hash key *<id>* with the *<frame>* and *<slot>* values of the identified SDM712i.
  - Increments a local count of found Services.

- At 00:00 on Friday, for each entry j within the hash SDM712iHash{}:
  - The application opens a TCP/IP socket at port 13000 of SDM712iHash{j}.
  - The application issues XML command: *<openID>j</openID>* and expects XML reply
  - *<ack/>*
  - The application issues XML command: *<setMute12>inc 1</setMute12>* and expects XML reply stream *<ack/>*
  - *Automatically , when the set operation is done, will receive <mute12>ON</mute12>*
  - The application issues XML command: *<getMute12/>* and expects XML reply stream *<mute12>ON </mute12>*
  - If not, the application:
  - issues XML stream: *<closeID>j</closeID>* .
  - Moves entry j of SDM712iHash{} to entry j of SDM712iFooHash{}
  - If so, the application issues XML stream: *<closeID>j</closeID>*
  - The application closes TCP/IP socket at port 13000 of Service Node *<address>* XML
  - tag value.

- The application exits logging:
  - The iControl™ Gateway Accessed.
  - The total amount of SDM712i_93 registered through that iControl™ Gateway.
  - The amount of (un)successfully muted SDM712i_93.
  - For each (un)successfully muted SDM712i_93, the result, the Service Node Short ID (*<id>*) and the Service Node physical location (*<frame>* and *<slot>*).

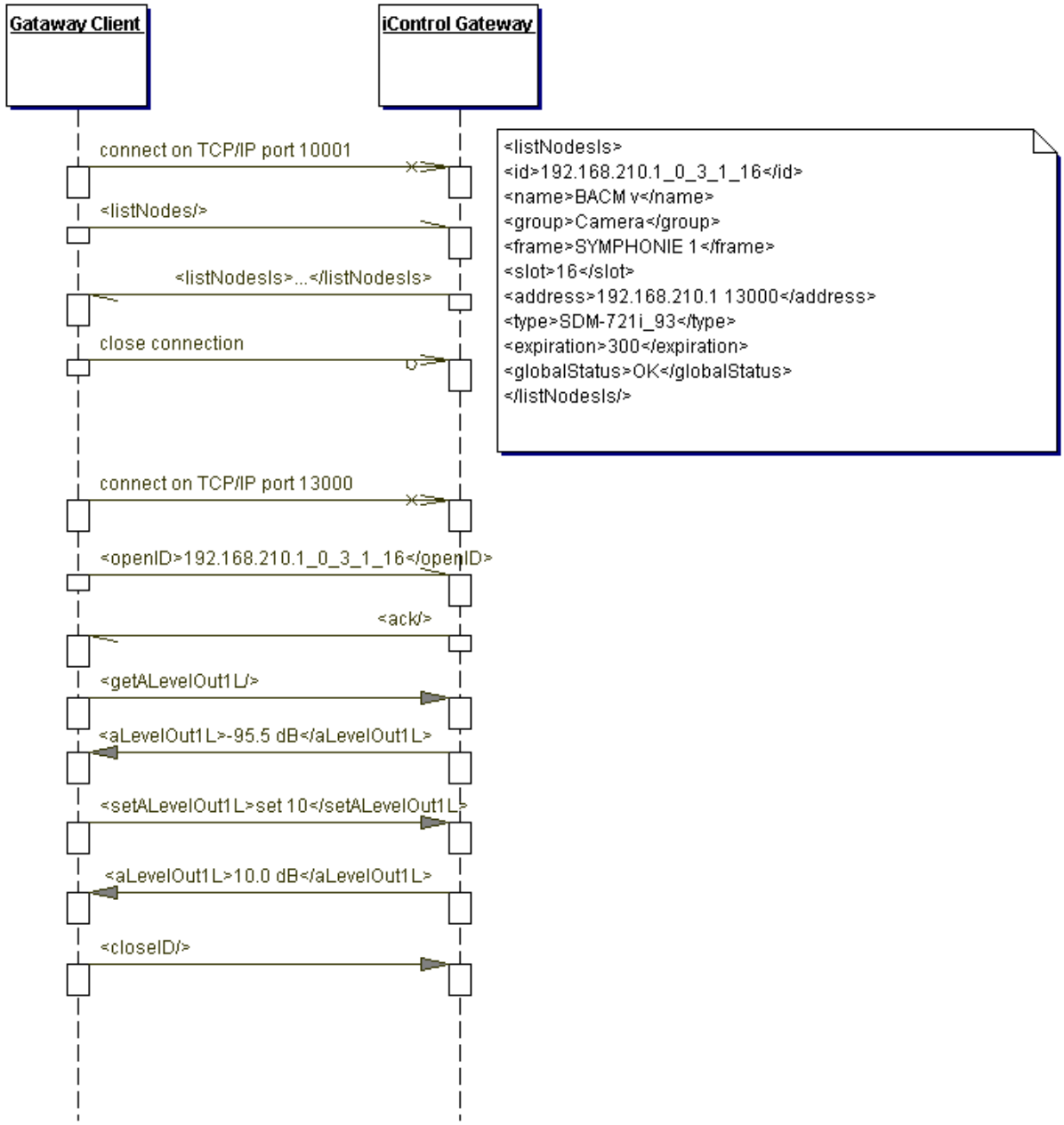
5.2 TELNET SESSION EXAMPLE WITH A SDM-712 SERVICE THROUGH THE ICONTROL GATEWAY.

5.2.1 GET AND SET

=> <getALevelOut1L/>\r
<= <aLevelOut1L>-95.5 dB</aLevelOut1L>\r\n
=> <setALevelOut1L>inc 1</setALevelOut1L>\r
<= <aLevelOut1L>-95.0 dB</aLevelOut1L>\r\n
=> <setALevelOut1L>dec 1</setALevelOut1L>\r
<= <aLevelOut1L>-95.5 dB</aLevelOut1L>\r\n
=> <setALevelOut1L>set 10</setALevelOut1L>\r
<= <aLevelOut1L>10.0 dB</aLevelOut1L>\r\n

where

<= means a response from iControl and => means a request to iControl



5.2.2 GETTING INFORMATION ABOUT ALEVELOUT1L

=> <getParameterInfo key="aLevelOut1L"/>\r

<= <parameterInfo name="CH-1" isActive="true" min="-96.0" max="12.0" step="0.5" fstep="10" nominal="0.0" unit="dB"/>\r\n

6. Appendix A XML Primer (W3C's technical reports page)

What is XML

XML is a method for putting structured data in a text file. For "structured data" think of such things as spreadsheets, address books, configuration parameters, financial transactions, technical drawings, etc. Programs that produce such data often also store it on disk, for which they can use either a binary format or a text format. The latter allows you, if necessary, to look at the data without the program that produced it. XML is a set of rules, guidelines, conventions, whatever you want to call them, for designing text formats for such data, in a way that produces files that are easy to generate and read (by a computer), that are unambiguous, and that avoid common pitfalls, such as lack of extensibility, lack of support for internationalization/localization, and platform-dependency.

XML History

Development of XML started in 1996 and it is a W3C standard since February 1998, which may make you suspect that this is rather immature technology. But in fact the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. And of course HTML, whose development started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, but vastly more regular and simpler to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

XML vs HTML

Like HTML, XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form name="value"), but while HTML specifies what each tag & attribute means (and often how the text between them will look in a browser), XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, if you see "<p>" in an XML file, don't assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person…

XML files are text files, as I said above, but even less than HTML are they meant to be read by humans. They are text files, because that allows experts (such as programmers) to more easily *debug* applications, and in emergencies, they can use a simple text editor to fix a broken XML file. But the rules for XML files are much stricter than for HTML. A forgotten tag, or an attribute without quotes makes the file unusable, while in HTML such practice is often explicitly allowed, or at least tolerated. It is written in the official XML specification: applications are not *allowed* to try to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and issue an error.

Since XML is a text format, and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the XML developers. The advantages of a text format are evident (see 3 above), and the disadvantages can usually be compensated at a different level. Disk space isn't as expensive anymore as it used to be, and programs like zip and gzip can compress files very well and very fast. Those programs are available for nearly all platforms (and are usually free). In addition, communication protocols such as modem protocols and HTTP/1.1 (the core protocol of the Web) can compress data on the fly, thus saving bandwidth as effectively as a binary format.

7. Appendix B The XML Technologies Family

There is ***XML 1.0***, the specification that defines what "tags" and "attributes" are, but around XML 1.0, there is a growing set of optional modules that provide sets of tags & attributes, or guidelines for specific tasks. There is, e.g., ***Xlink*** (still in development as of November 1999), which describes a standard way to add hyperlinks to an XML file. ***XPointer*** & ***XFragments*** (also still being developed) are syntaxes for pointing to parts of an XML document. (An XPointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.) ***CSS***, the style sheet language, is applicable to XML as it is to HTML. ***XSL*** (autumn 1999) is the advanced language for expressing style sheets. It is based on ***XSLT***, a transformation language that is often useful outside XSL as well, for rearranging, adding or deleting tags & attributes. The ***DOM*** is a standard set of function calls for manipulating XML (and HTML) files from a programming language. ***XML Namespaces*** is a specification that describes how you can associate a URL with every single tag and attribute in an XML document. What that URL is used for is up to the application that reads the URL, though. (RDF, W3C's standard for metadata, uses it to link every piece of metadata to a file defining the type of that data.) ***XML Schemas 1*** and ***2*** help developers to precisely define their own XML-based formats. There are several more modules and tools available or under development. Keep an eye on W3C's technical reports page.

XML Benefits

By choosing XML as the basis for some project, you buy into a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs/procedures that manipulate it, but there are many tools available and many people that can help you. And since XML, as a W3C technology, is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. *XML isn't always the best solution, but it is always worth considering.*