XML-RPC SDK

(Version 1.6)

1412 4618

1. Introduction

This document describes the XML-RPC API used for controlling endpoints connected to a Polycom KWS. Currently, KIRK DECT handsets connected to a KWS300/KWS6000 or KWS2500/8000 can be controlled.

The XML-RPC API enables a PC or other device to control endpoints connected to a KWS. The PC can receive status events from endpoints, send and receive SMS, send broadcast messages and establish text calls.

A typical setup is depicted in Figure 1 where a PC communicates with a KWS via XML-RPC. The communication between the KWS and the endpoints are carried via DECT.

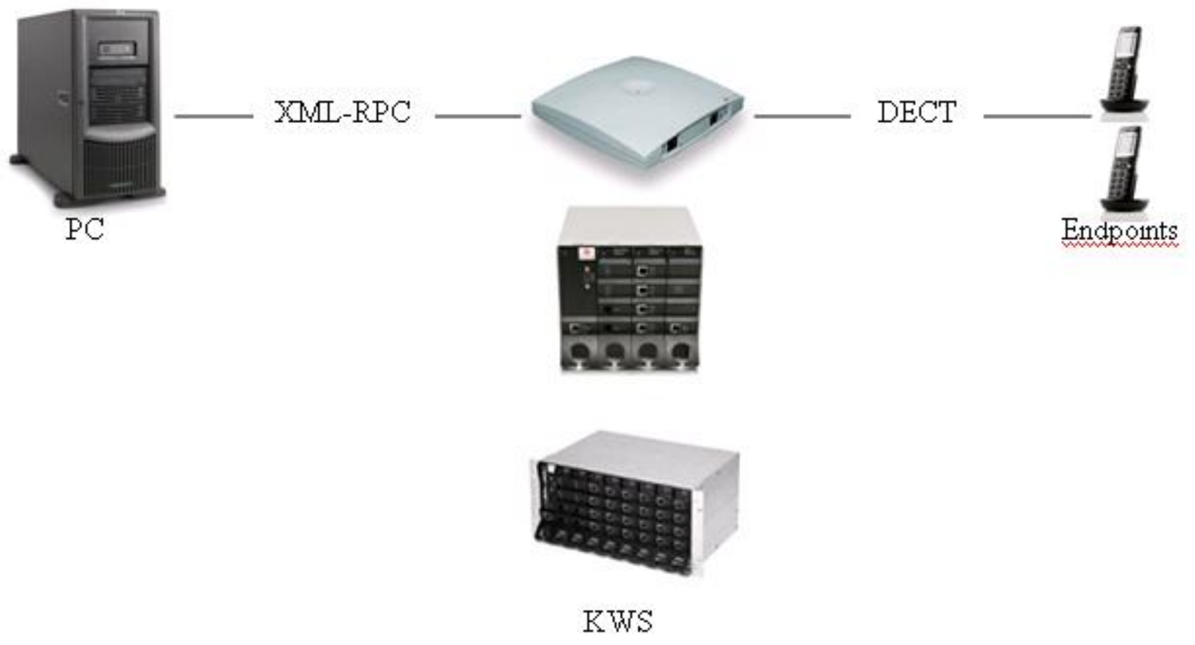


Figure 1 Setup for XML-RPC based applications

This document can be used as a quick start guide and for reference.

2. Revision history

Revision	Date	Changes
1.0		Initial revision.
1.1		Added functionality for upcoming handset. Added endpoint_release event. Restructure document.
1.2		Added endpoint_broadcast event. Release reasons updated.
1.3		Proprietary discriminators updated Updated LED control in endpoint_sms_advanced()
1.4		Added endpoint_base_stations function and event.
1.5		Sort functions and events alphabetically. More details added to the README for the examples. Added guidelines for programming for KWS6000 redundancy. Added guidelines for performance considerations. Added version requirement information. Updated endpoint_hardware_extension() and endpoint_sms_advanced().
1.5.1	2011-03-31	Typographical corrections.
1.6	2011-07-05	Added KWS2500 and KWS8000

3. Contents

4. XML-RPC

XML-RPC is essentially Remote Procedure Calls transmitted as XML via HTTP/HTTPS. Remote procedure calls is a paradigm where a client is executing procedures on a server.

XML-RPC is a standard specified at http://www.xmlrpc.com . The standard specifies what types of parameters that can be used and how the procedure calls and responses are formatted in XML.

XML-RPC is easy to use. Libraries for easy access to XML-RPC are provided for most programming languages meaning that the developer will not need to bother how functionality is implemented in the system. Some libraries for accessing XML-RPC are listed at http://www.xmlrpc.com .

If however, the developer prefers to implement the XML-RPC application from scratch the low-level format is clear text and easy to understand. The following example is a request for sending a SMS to a handset:

```
POST /RPC2 HTTP/1.0
Host: 172.29.198.51
User-Agent: xmlrpclib.py/1.0.1 (by www.pythonware.com)
Content-Type: text/xml
Content-Length: 326

<?xml version='1.0'?>
<methodCall>
  <methodName>endpoint_sms</methodName>
  <params>
    <param><value><int>160160</int></value></param>
    <param><value><string>2632</string></value></param>
    <param><value><string>3000</string></value></param>
    <param><value><string>Hello World!</string></value></param>
  </params>
</methodCall>
```

The following is the response from the KWS:

```
HTTP/1.1 200
Content-Type: text/xml
Content-Length: 113
Date: Mon, 31 Aug 2009 10:28:20 GMT
Server: KirkHTTPServer/1.0

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param><value><int>0</int></value></param>
  </params>
</methodResponse>
```

For more information about the low-level format refer to http://www.xmlrpc.com.

4.1 Security considerations

Passwords and other data is transmitted in clear text if XML-RPC is accessed via HTTP. Therefore, it is recommended to use HTTPS for XML-RPC.

4.2 Persistent HTTP/HTTPS connections

Creating a new HTTPS connection for every XML-RPC request is a relatively time consuming process and may degrade performance of the system. Therefore, it is recommended to use persistent connections when HTTPS is employed. Refer to RFC2616 “Hypertext Transfer Protocol -- HTTP/1.1” section 8.1 for further details.

5. API specification

5.1 Sessions

The KWS requires a session to be established to enable communication. When the session is established the PC can call functions to control the endpoints.

Figure 2 illustrates session establishment and sending a SMS to an endpoint. The dashed lines indicate return values from function calls, e.g. session ID illustrates the return value to the session_login function call.

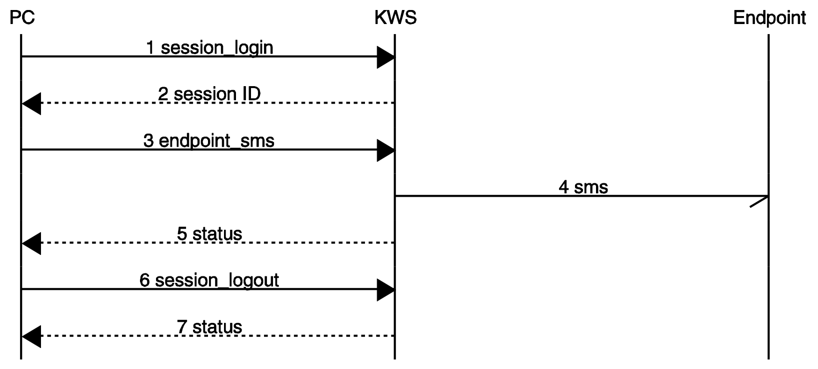


Figure 2 XML-RPC session

If the PC is not periodically communicating with the KWS the KWS will end the session.

Events send from the endpoints or generated internally by the KWS are queued in the KWS and can be received by the PC by calling the session_receive() function.

Figure 3 illustrates a session where the PC logs in to the KWS and waits for events from the KWS by calling the session_receive() function. In the illustration the event is received because an endpoint has sent a endpoint_statu event.

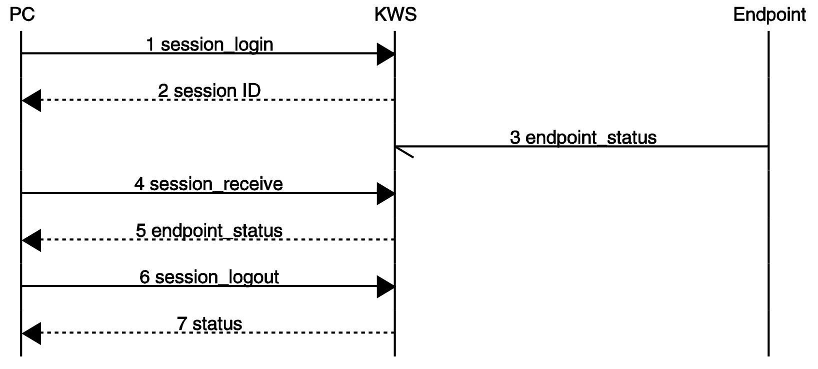


Figure 3 XML-RPC receive events

5.2 Text calls

The KWS uses the concept of text calls. A text call can be established from the PC to an endpoint or the other way. When the call is established the PC can control the display of the endpoint and the end-point can e.g. send user key presses to the PC.

Figure 4 illustrates a text call. The endpoint makes a call setup that is accepted by the PC. The PC sends some text to the display and waits for key presses from the user. The user presses a key and finally, the PC releases the text call.

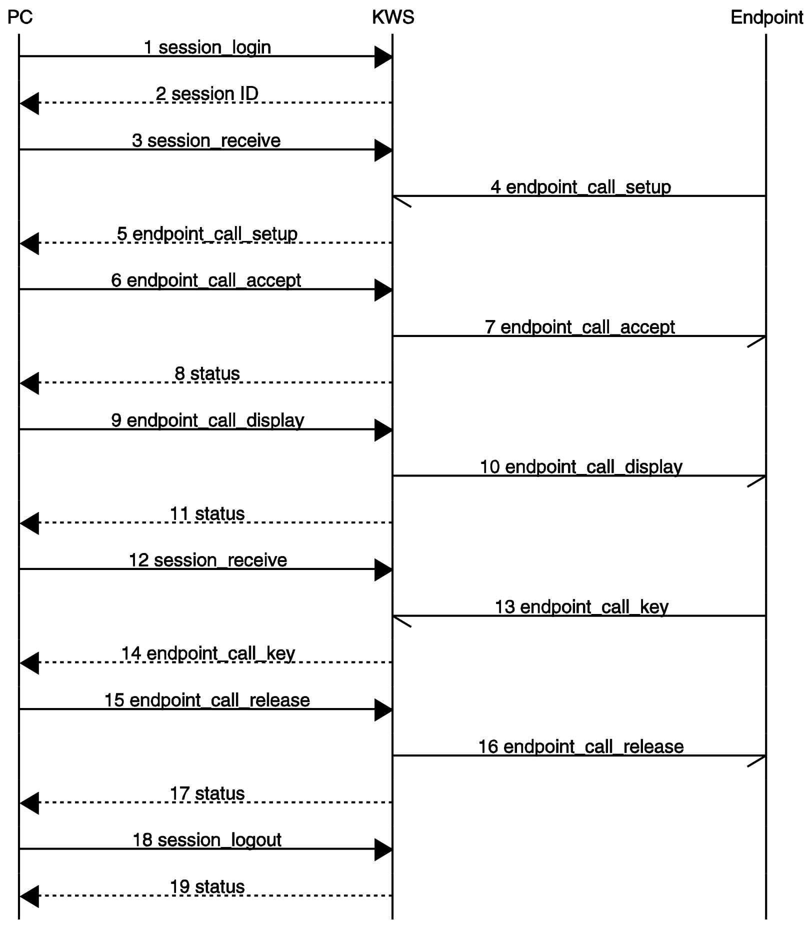


Figure 4 XML-RPC text call

5.3 Functions

The functions available through XML-RPC are listed in the following.

When a parameter is written with a trailing "=<value>" this indicates a default value and that the parameter is optional.

For some functions and events firmware version requirements are listed. When no firmware requirements are listed the function is available for all relevant products in all firmware versions.

5.3.1 endpoint_base_stations()

Query an endpoint for its set of base stations including the current base station. This feature can be used for locating the endpoint. The set of base stations is returned in the response end-point_base_stations event.

int endpoint_base_stations(int session, string to)

- session : The session ID
- to : The endpoint to receive the request

Requirements:

Handset	**Firmware version**
5020 and 5040 series handsets	PCS08Ja (or newer) released 2010-Q3
6020 and 6040 series handsets	PCS06Da (or newer) released 2010-Q3
7020 and 7040 series handsets	PCS06Da (or newer) released 2010-Q3
KWS	**Firmware version**
KWS300/6000	PCS05C_ (or newer) released 2010-Q3
KWS2500/8000	PCS06__(or newer) released 2011-Q3

5.3.2 endpoint_broadcast()

Send a broadcast message to one or more handsets. The status of the broadcast is received in a end-point_broadcast event.

int endpoint_broadcast(int session, string display, int repeat=3, string type=”all”, string to=””, int pro-prietary_discriminator=0x82, base64 data=””)

- session : The session ID
- display : Text to display in endpoints. Maximum length is 19 characters.
- repeat : The number of times to repeat the broadcast.
- type : The type of broadcast “all”, “group” or “endpoint”
- to : The destination for the broadcast
- proprietary_discriminator : Proprietary discriminator describing the format of the broadcast. The values are listed in Appendix – Proprietary discriminators

- data : Binary data send to endpoint. Only used for special values of the proprietary discriminator. Maximum length is 19 characters.
- return : The status of the operation. The status codes are listed in the section Status codes

5.3.3 endpoint_call_accept()

Accept a text call from an endpoint. This is used in response to an endpoint_call_setup event.

int endpoint_call_accept(int session, int call)

- session : The session ID
- call : The call ID of the call
- return : Status of operation. The status codes are listed in the section Status codes

5.3.4 endpoint_call_display()

Send information to an endpoint during a text call. The setup specifications are listed in the section Appendix – Setup specifications.

int endpoint_call_display(int session, int call, string callback, string display, int setupspec1=0x00)

- session : The session ID
- call : The call ID of the call
- callback : The number to present as callback
- display : Text to show in the display of the endpoint. Maximum length is 72 characters but it depends on the specific endpoints.
- setupspec1 : Setup specification for the endpoint
- return : Status of operation. The status codes are listed in the section Status codes

5.3.5 endpoint_call_menu()

Display a menu in an endpoint during a text call. The index of the selected menu item is returned in a endpoint_call_menu event.

int endpoint_call_menu(int session, int call, array of string items)

- session : The session ID
- call : The call ID of the call
- items: An array of strings with the items to display in the menu.
- return : Status of operation. The status codes are listed in the section Status codes

5.3.6 endpoint_call_prompt()

Display a text prompt in an endpoint during a text call. The text entered is returned in a end-point_call_prompt event.

int endpoint_call_prompt(int session, int call, string display, string mode=”alphanumeric”, int min=0, int max=72)

- session : The session ID
- call : The call ID of the call
- display : Text to show in the display of the endpoint. Maximum length is 72 characters but it depends on the specific endpoints.
- mode : Specifies what characters are allowed to be entered. Valid modes are “alphanumeric” or “numeric”.
- min : The minimum length of the entered text.
- max : The maximum length of the entered text.
- return : Status of operation. The status codes are listed in the section Status codes

5.3.7 endpoint_call_release()

Release a text call. Release reasons are listed in the section Release reasons.

int endpoint_call_release(int session, int call, int reason=0)

- session : The session ID
- call : The call ID of the call
- reason : Call release reason
- return : Status of operation. The status codes are listed in the section Status codes

5.3.8 endpoint_call_setup()

Establish a new text call to a endpoint. If the call is accepted by the endpoint a endpoint_call_accept event is received otherwise a endpoint_call_release event is received. The setup specifications are listed in the section Appendix – Setup specifications.

int endpoint_call_setup(int session, string to, string callback, string display, int setupspec1=0x81, int setupspec2=0x00, int setupspec3=0x00, int setupspec4=0x00, int alertpattern=0x00, int alert-tone=0x00, int alerttimeout=0x00, int displaytimeout=0x00, dateTime.iso8601 datetime=0x00)

- session : The session ID
- to : The endpoint to receive the request

- callback : The endpoint to present as callback
- display : Text to show in the display of the endpoint. Maximum length is 72 characters but it depends on the specific endpoints.
- setupspec1 : Setup specification for the endpoint
- setupspec2 : Setup specification for the endpoint
- setupspec3 : Setup specification for the endpoint
- setupspec4 : Setup specification for the endpoint
- alertpattern : Alert pattern for the text call
- alerttone : Alert tone for the text call
- alerttimeout : Alert timeout for the text call
- displaytimeout : Display timeout for the text call
- datetime : Timestamp of the message in local time
- return : A call positive ID to identify the call or a negative status code on failure. The status codes are listed in the section Status codes

5.3.9 endpoint_hardware_extension()

Access hardware extensions build into the endpoint. This function can either be used to access the hardware in one endpoint, in a group of endpoints or all endpoints. Responses for this function are received in endpoint_hardware_extension events.

Please refer to the relevant handset documentation for further details.

int endpoint_extended_hardware(int session, int hardware, int action, int repeat=3, string type=”all”, string to=””, base64 data=””)

- session : The session ID
- hardware : The hardware element to access

  0x01: Tear off sensor

  0x02: Motion sensor no move

  0x03: Motion sensor fast run

  0x04: Motion sensor not vertical
- action : The action to perform on the hardware element

0x01: Write configuration

0x02: Read configuration

0x03: Read sensor values

- repeat : The number of times to repeat the request. This is only relevant for type “group” and “all”
- type : The type of request “all”, “group” or “endpoint”
- to : The destination for the request
- data : Data specific for the hardware element.
- return : Status of operation. The status codes are listed in the section Status codes

Requirements:

Handset	**Firmware version**
6040 series handsets	All versions
7040 series handsets	All versions
KWS	**Firmware version**
KWS300/6000	PCS05__(or newer) released 2010-Q1
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.3.10 endpoint_sms()

Send a SMS to an endpoint. This is a fire and forget function.

int endpoint_sms(int session, string to, string callback, string display, int alerting=0x01)

- session : The session ID
- to : The endpoint to receive the request
- callback : The number to present as callback
- display : Text to show in the display of the endpoint. Maximum length is 72 characters but it depends on the specific endpoints.
- alerting : The alert pattern for the SMS. Refer to Appendix – Setup specifications
- return : Status of operation. The status codes are listed in the section Status codes

5.3.11 endpoint_sms_advanced()

Send an advanced SMS to an endpoint. This function can either be used to send an SMS to one endpoint, to a group of endpoints or to all endpoints. Responses for this function are received in endpoint_sms_advanced events.

The function can be used either to send a new advanced SMS or modify an advanced SMS already delivered to an endpoint.

Please refer to the relevant handset documentation for further details.

nt endpoint_sms_advanced(int session, int action, int response, int id=0, int repeat=3, string type=”all”, string to=””, led=0x00, icon=0x00, color=0x00, setup=0x00, int alerttone=0x00, int alertpattern=0x00, int alertvolume=0, int alerttimeout=0, int displaytimeout=0, string display=””, string callback=””)

- session : The session ID
- action : Action to perform with the message.

  0x01: New normal SMS

  0x02: New alarm SMS and put in front

  0x10: Ask for status (Last byte is response enabler/mask) (if SMS do not exist, the endpoint will return deleted SMS)

  0x12: Delete message. If the selected message is active in the display, the endpoint leaves the message menu.

  0x13: LED control only (Last byte is LED control), only if the selected message is active in display.

  0x14: Icon number only (Last byte is Color control)
- response : Bit mask specifying what responses are allowed for this message.

Response enabler mask

Bit	7	6	5	4	3	2	1	0
Enable Response Action	Allow Task-Delete	*AllowTask Com-pleted*	*AllowTask Al-mostDone*	*AllowTask Started*	Al-lowAlarm Rejected	Al-lowAlarm Callback	Al-lowAlarm Accepted	Auto-Delete Rejected
Soft key text	Delete	Done	Nearly	Started	Reject	No text	Accept	No text

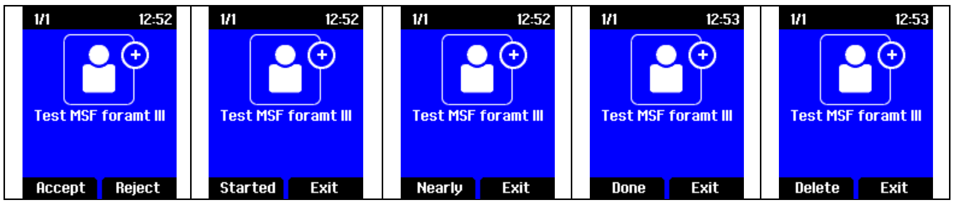
AutoDeleteRejected: If the user select reject the message is deleted.

AllowAlarmRejected and **AllowAlarmAccepted** are perceived as a pair when the user activates one of them the other is deleted.

AllowAlarmCallback: If bit is on and the user activates the hook key. A message "Alarm acknowledge by call back" is send to the system.

AllowTaskDelete: User can delete the message.

Example message send all response enabler bit on user press left soft key:



- id : 16-bit message ID identifying the message
- repeat : The number of times to repeat the request. This is only relevant for type "group" and "all"
- type : The type of request "all", "group" or "endpoint"
- to : The destination for the request
- led : Control the status LEDs on the endpoint
  - Bit 0,1,2 Selected Led
  - Bit 3,4,5 Led command: On, flash slow, flash fast
  - Bit 6: unused.¨
  - Bit 7: If bit is set then don't touch the LED's.

Bit no	7	6	5	4	3	2	1	0
Bit pattern	Ignore all	Unused	Led cmd	Led cmd	Led cmd	Blue	Green	Red

  - Led command:

Bit no	5	4	3
Selected led on	0	0	0
Selected Led flash slow	0	0	1
Selected Led flash fast	0	1	0
Switch slow	0	1	1
Switch fast	1	0	0

- icon : The icon to display for the message. Depends on the endpoint but 0x00 means no icon.

- color : For future use
- setup : Bit mask with setup specification for the message

7	6	5	4	3	2	1	0
		SIC	IIVC	IC	AV	LV	SIS
TBD	TDB	Silent if PP in charger	Ignore SMS if PP in voice call	Ignore SMS if PP in charger	Always vibrate	Use local Alert Volume	Save in stack

- alerttone : Alert tone for the message. Refer to Appendix – Setup specifications for the specification
- alertpattern : Alert pattern for the message. Refer to Appendix – Setup specifications for the specification
- alertvolume : Alert volume for the message

  0x00 is silent
  0x01 is lowest non silent level
- alerttimeout : Alert timeout in units of 0.5 second. 0 means no timeout
- displaytimeout : Display timeout in units of 0.5 second. 0 means no timeout
- display : Text to show in the display of the endpoint. Maximum length is 52 characters but it depends on the specific endpoints.
- callback : The endpoint to present as callback
- return : Status of operation. The status codes are listed in the section Status codes

Requirements:

Handset	**Firmware version**
6020 and 6040 series handsets	All versions
6020 and 7040 series handsets	All versions
KWS	**Firmware version**
KWS300/6000	PCS05__ (or newer) released 2010-Q1
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.3.12 endpoint_status()

Send a status request to a handset. The status is received in an endpoint_status event.

int endpoint_status(int session, string to, int status, base64 data="")

- session : The session ID
- receiver : The handset to receive the request
- status : The kind of status to request.
- data : Data for the status request
- return : Status of operation. The status codes are listed in the section Status codes

5.3.13 session_login()

Establish a new session between the application and the KWS.

int session_login(string user, string password)

- user : The user to authenticate (default : GW-DECT/admin)
- password (the password for the user):
  - KWS300/6000-default: ip6000
  - KWS2500/8000-default: kws8000
- return : A unique random session ID greater than 0 or RPC_LOGIN_INCORRECT if not authenticated

5.3.14 session_logout()

End the session.

boolean session_logout(int session)

- session : The session ID
- return : The status of the operation. The status codes are listed in the section Status codes

5.3.15 session_receive()

Poll for new or queued data/events relevant for this session on the server. This function will block until data is available or the timeout is expired. For a list of events refer to the section Events.

array of struct session_receive(int session, int timeout=0)

- session : The session ID

- timeout : Time in seconds to wait before return
- return : A vector/list/array with events

5.3.16 system.listMethods()

XML-RPC introspection function for listing available methods.

array of string system.listMethods()

- return: A vector/list/array containing the names of the methods

5.3.17 system.methodHelp()

XML-RPC introspection function for getting a short help for a method.

string system.methodHelp(string methodname)

- methodname: Method name of the method to get help for
- return: A text describing the method

5.3.18 system.methodSignature()

XML-RPC introspection function for getting the signature for a method.

array of string system.methodSignature(string methodname)

- methodname: Method name of the method to get signature for
- return: A vector/list/array containing the parameter types of the method

5.4 Status codes

When some of the functions return a status it will be one of the following status codes.

RPC_OK	0	Operation is performed OK.
RPC_UNKOWN_SESSION	-1	The session requested is not valid or has expired.
RPC_USER_BUSY	-2	The endpoint that is contacted is busy and the operation cannot be performed.
RPC_RECV_ACTIVE	-3	A session_receive request is active while another one is requested.
RPC_LOGIN_INCORRECT	-4	The username or password provided for login is incor-rect.

RPC_UNKOWN_CALL	-5	The call requested is not valid or is released.

5.5 Events

The events that can be received via the session_receive() function are listed in the following.

All events returned are structures with a member called "type" that tells the name of the type of event.

5.5.1 endpoint_base_stations

This event is received in response to endpoint_base_stations() and can be used for locating an end-point.

struct endpoint_base_stations {string type="endpoint_base_stations", string from, array of struct rfp_map {int rpn, int rssi}}

- type : "endpoint_base_stations"
- from : The endpoint from which the base station set is received
- rfp_map[] : An array of up to 5 elements containing the Radio Part Number (RPN) and Received Signal Strength Indication (RSSI) of the base stations set of the endpoint. The first element is the current base station of the endpoint meaning the one that the handset is currently connected to.

Requirements:

Handset	**Firmware version**
5020 and 5040 series handsets	PCS08Ja (or newer) released 2010-Q3
6020 and 6040 series handsets	PCS06Da (or newer) released 2010-Q3
7020 and 7040 series handsets	PCS06Da (or newer) released 2010-Q3
KWS	**Firmware version**
KWS300/6000	PCS05C_ (or newer) released 2010-Q3
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.5.2 endpoint_broadcast

This event informs about status of a broadcast initiated by the endpoint_broadcast() function.

struct endpoint_broadcast {string type="endpoint_broadcast", int status}

- type : "endpoint_broadcast"

- status : The status of the broadcast.
    - NORMAL_RELEASE : The message has been broadcasted.
    - USER_BUSY : The messages has not been broadcasted because the broadcasting system was busy.

Requirements:

KWS	**Firmware version**
KWS300/6000	PCS05A_ (or newer) released 2010-Q1
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.5.3 endpoint_call_accept

Indicates that an endpoint has accepted a text call. This is a possible response to endpoint_call_setup().

struct endpoint_call_accept {string type=”endpoint_call_accept”, int call}

- type : ”endpoint_call_accept”
- call : The call ID of the call

5.5.4 endpoint_call_key

User interaction sent from an endpoint within a text call.

struct endpoint_call_key {string type=”endpoint_call_key”, int call, string key}

- type : ”endpoint_call_key”
- call : The call ID of the call
- key : The key pressed by the user

5.5.5 endpoint_call_menu

User menu selection sent from an endpoint in response to endpoint_call_menu().

struct endpoint_call_menu {string type=”endpoint_call_menu”, int call, int item, int errorcode}

- type : ”endpoint_call_menu”
- call : The call ID of the call
- item : The index of the selected menu item
- errorcode : Currently always 0

5.5.6 endpoint_call_prompt

User entered text sent from an endpoint in response to endpoint_call_prompt().

struct endpoint_call_prompt{string type="endpoint_call_prompt", int call, string text, int errorcode}

- type : "endpoint_call_menu"
- call : The call ID of the call
- text : The text entered
- errorcode : Currently always 0

5.5.7 endpoint_call_release

A text call has been released from the endpoint. Release reasons are listed in the section Release reasons.

struct endpoint_call_release {string type="endpoint_call_release", int call, int reason}

- type : "endpoint_call_release"
- call : The call ID of the call
- reason : The reason for the call release.

5.5.8 endpoint_call_setup

Indicates a text call initiated from an endpoint. The text call is accepted by calling endpoint_call_accept().

struct endpoint_call_setup {string type="endpoint_call_ setup", int call, string from, string to}

- type : "endpoint_call_ setup"
- call : The call ID of the call
- from : The endpoint initiating the text call
- to : The destination of the call

5.5.9 endpoint_hardware_extension

This event is received in response to endpoint_hardware_extension() or when a hardware element needs to deliver information.

Please refer to the relevant handset documentation for further details.

struct endpoint_hardware_extension {string type="endpoint_hardware_extension", string from, int hardware, int reason, base64 data}

- type : “endpoint_hardware_extension”
- from : The endpoint sending the response
- hardware : The hardware element from which the response comes. Refer to end-point_hardware_extension() for a list of hardware elements
- reason : The reason for the response

  0x01: Confirm on write configure action in ExtenHwReq signal.
  0x02: Confirm on read configure action in ExtenHwReq signal.
  0x03: Detected values (response to 0x03)
  0x04: Confirm on activate the device
  0x05: Confirm on deactivate the device
  0x10: Limit exceed
  0x11: Limit normal
  0x12: Timeout
- data : Data specific for the hardware element.

Requirements:

Handset	**Firmware version**
6040 series handsets	All versions
7040 series handsets	All versions
KWS	**Firmware version**
KWS300/6000	PCS05__(or newer) released 2010-Q1
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.5.10 endpoint_release

Indicates that a connection to an endpoint has been released. This event is sent when an endpoint re-lease is not related to a text call.

struct endpoint_release {string type=”endpoint_release”, string from, int reason}

- type : ”endpoint_release”
- from : The endpoint for which the connection is released
- reason : The reason for the release. Refer to Release reasons for the specification.

Requirements:

KWS	Firmware version
KWS300/6000	PCS05__ (or newer) released 2010-Q1
KWS2500/8000	PCS06__ (or newer) released 2011-Q3

5.5.11 endpoint_sms

A SMS sent from an endpoint.

struct endpoint_sms {string type=”endpoint_sms”, string from, string to, string message}

- type : ”endpoint_sms”
- from : The endpoint sending the SMS
- to : The destination of the SMS
- text : The message contained in the SMS

5.5.12 endpoint_sms_advanced

This event is received when the user of an endpoint responds to an advanced SMS sent using the end-point_sms_advanced() function.

Please refer to the relevant handset documentation for further details.

struct endpoint_sms_advanced {string type=”endpoint_sms_advanced”, string from, int id, int action, array of int status, array of struct rfp_map {int rpn, int rssi}, int datadescriptor, base64 data}

- type : ”endpoint_sms_advanced”
- from : The endpoint sending the response
- id : ID identifying the message that response is related to
- action : Action performed by the user in response to the message

  0x80: Sms status is normal sms (set state = 00)
  0x81: Alarm accepted (I am responding; set state = 01)
  0x82: Alarm accepted by pressing the hook key and using the call back number (set state = 02)
  0x83: Alarm rejected. (User rejects; set state = 03)
  0x84: Started on task (User is starting / have started on the assigned task; set state = 04)
  0x85: Task almost completed (User is ready for a new job in a few moments; state = 05)
  0x86: Task completed. (User has finished the assigned task; state = 06)

0x88: Deleted Sms, this action should always be allowed. (SMS does no longer exist in PP; User has deleted this sms)

- status[0] : Enhanced status information from the endpoint

  Bit 7: In charger
  Bit 6: Speech active (active voice call)
  Bit 5: Hands free active
  Bit 4: Microphone muted. (from PP key pad)
  Bit 3: Vibrator enabled. (via PP setup menu)
  Bit 2..0: Battery level 0=Empty, 7=Full
- status[1] : Enhanced status information from the endpoint

  Bit 7: Silent Mode (set via key pad)
  Bit 6: Absent Mode (PPStatus 0x20)
  Bit 5..3: Not used
  Bit 2..0: Start volume setting.
- rfp_map[] : An array of 5 elements containing the Radio Part Number (RPN) and Received Signal Strength Indication (RSSI) of the base stations with the strongest signal.

- datadescriptor : Data descriptor describing additional data if present.

  0x00: No extra data.
- data : Additional data

Requirements:

Handset	**Firmware version**
6020 and 6040 series handsets	All versions
6020 and 7040 series handsets	All versions
KWS	**Firmware version**
KWS300/6000	PCS05__(or newer) released 2010-Q1
KWS2500/8000	PCS06__(or newer) released 2011-Q3

5.5.13 endpoint_status

Status sent from an endpoint. Either initiated by the endpoint or by a call to endpoint_status(). The status types are listed in the section Appendix – Status types.

struct endpoint_status {string type=”endpoint_status”, string from, int status, base64 data}

- type : ”endpoint_status”

- from : The endpoint sending the status

- status : The type of status
- data : Status data from the endpoint

5.6 Release reasons

The following table covers the release reasons sent when a text call is released.

NORMAL_RELEASE	0	Normal release reason.
USER_NOT_IN_RANGE	3	The requested user is not in the range of the system.
USER_BUSY	4	The requested user is busy.
UNKNOWN_USER	5	The requested user is not known to the system.
USER_UNKNOWN_REASON	6	Unknown release reason send from endpoint.
INCOMPATIBLE_SERVICE	33	Incompatible service or not a Polycom Kirk endpoint.
UNKNOWN_REASON	84	Unknown release reason.

6. API examples

This section contains a few simple examples on how to use the XML-RPC API. For more examples refer to the samples included in the SDK. The examples in the SDK demonstrates Python, Java and Visual Basic .NET and covers more complex scenarios.

6.1 Python – send SMS

This is a simple example implemented using the built-in XML-RPC module of Python. The program logs in to a KWS and sends a SMS to an endpoint.

```
import xmlrpclib

host = "172.29.198.51"
to = "2632"
callback = "3000"
display = "Hello World!"

server = xmlrpclib.Server("http://" + host)

session = server.session_login("GW-DECT/admin", "ip6000")

server.endpoint_sms(session, to, callback, display)

server.session_logout(session)
```

6.2 Python – receive SMS

This is a simple example implemented using the built-in XML-RPC module of Python. The program logs in to a KWS and receives SMS messages from endpoints.

```
import xmlrpclib

host = "172.29.198.51"

server = xmlrpclib.Server("http://" + host)

session = server.session_login("GW-DECT/admin", "ip6000")

while True:
  events = server.session_receive(session, 60)
  for event in events:
    if event["type"] == "endpoint_sms":
      print "SMS '"+event["display"]+"' to "+event["to"]+" from 
"+event["from"]
    else:
      print "Unknown event: "+str(event)

server.session_logout(session)
```

6.3 Java – send SMS

This is a simple example implemented using the Apache Java XML-RPC package. The program logs in to a KWS and sends a SMS to an endpoint.

```
import java.util.*;
import java.net.*;
import org.apache.xmlrpc.client.*;

public class SendSMS {

  private static String host = "172.29.198.51";
  private static String to = "2632";
  private static String callback = "3000";
  private static String display = "Hello World!";

  public static void main (String [] args) {
    try {
      XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
      config.setServerURL(new URL("http://" + host + "/RPC2"));
      XmlRpcClient client = new XmlRpcClient();
      client.setConfig(config);

      Object[] params = new Object[]{new String("admin"), new
String("ip6000")};
      Integer session = (Integer) client.execute("session_login", params);

      params = new Object[]{session, to, callback, display};
      client.execute("endpoint_sms", params);

      params = new Object[]{session};
      client.execute("session_logout", params);

    } catch (Exception exception) {
      System.err.println("SendSMS: " + exception);
    }
  }
```

7. Performance considerations

This chapter covers performance limitations that need to be taken into account when designing an application that will send a high number of messages to endpoints.

7.1 Background

Sending a message to an endpoint will occupy one DECT channel while the message is being sent. A text call will occupy a DECT channel for the duration of the text call. If the endpoint is in range a text message is typically delivered to the endpoint in less than two seconds.

Establishing the required DECT connection from the KWS to the endpoint requires the KWS to page the endpoint and wait for an answer. The KWS retries for 12 seconds and afterwards the endpoint is reported out of range (release reason: USER_NOT_IN_RANGE). This means that endpoints not in range will occupy system resources for a longer time than endpoints in range and thereby reduce the message throughput greatly.

7.2 Recommendation

To avoid overloading the system with text messages, it is recommended to limit the number of pending message requests. By sending a limited number of messages and waiting for a release before sending the next message, the load of the system can be controlled, and the messages are delivered to the endpoints as fast as possible.

The table below lists the recommended maximum of pending message requests.

Server	**Channels per base station**	**Maximum pending messages**
KWS300	4	2
KWS6000	12	16
KWS2500	4	4
KWS8000	4	4

Be aware that the system uses the same messaging facilities to send Message Waiting Indication (MWI) to endpoints and hence PBX MWI activity will reduce the actual limits further.

7.3 Broadcast

As an alternative to the above recommendation broadcast can be considered for sending messages to all or a group of endpoints. This scales well and reduces the load on the system. Furthermore messages are delivered instantly to all recipients.

8. KWS6000 redundancy

The KWS6000 can be configured in a redundant setup. A KWS6000 redundant setup contains a primary KWS6000 and a secondary KWS6000. Special care must be taken when developing applications for such a setup.

In normal operation where the primary KWS6000 and the secondary KWS6000 is active the XML-RPC based application must communicate with the primary KWS6000. The secondary KWS6000 will not accept XML-RPC access.

In a failover scenario where only the secondary KWS6000 is active the XML-RPC based application must communicate with the secondary KWS6000. When normal operation is resumed the secondary KWS6000 will disconnect the XML-RPC based application and it must re-connect to the primary KWS6000.

In other words, the application must monitor the connection to the current KWS6000. When the connection fails the application must connect to the other KWS6000. Always switching to the other KWS6000 on a failure will ensure that the application is connected to the correct KWS6000.

9. Appendix – Status types

This section documents some of the status types that can be received in the endpoint_status event. The available status types depend on the type of endpoint.

PPChargerStatusInd	11	The handset is taken out of charger or put in to charger.
PPAlarm	12	The alarm button is pressed on the handset.

9.1.1 PPChargerStatusInd

The data field of the endpoint_status event contains the charger status:

- In to charger = 0x01
- Out of charger = 0x02

9.1.2 PPAlarm

Data description:

7 6 5 4 3 2 1 0	Byte no.
PPStatusByte	1
Current RPN$_0$	2
RSSI for RPN$_0$	3
RPN$_1$	4
RSSI for RPN$_1$	5
RPN$_2$	6
RSSI for RPN$_2$	7
RPN$_3$	8
RSSI for RPN$_3$	9
RPN$_4$	10
RSSI for RPN$_4$	11

PPStatusByte:
0x00: Alarm key released by user.
0x01: Alarm key pressed by user.

RPN$_{1..4}$ referes to OtherBases in the HandoverMap. If an RPN is nonexistent, [RPN$_x$, RSSI for RPN$_x$] will be [0xFF,0x00].

10. Appendix – Alert types

In some of the sub-events described on the following pages there will be an AlertType element.

The AlertType is used as follows:

AlertType value	Description
000b	Silence
001b	Pattern 2 (tone 9 in PP)
010b	Pattern 3 (tone 6 in PP)
011b	Pattern 4 (tone 7 in PP)
100b	Pattern 5 (KeyBeep (known key) in PP)
101b	Pattern 6 (KeyClick (unknown key) in PP)
110b	Pattern 7 (Accept Tone in PP)
111b	Vibrate If MSFDispReq Error Tone in PP

Concerning Key tones 100b, 101b, 110b, 111b these will only sound depending on the local settings in the PP. I.e. if the handset is in silent mode, the tones will not sound.

11. Appendix – Setup specifications

SetupSpec1: Has the following format:

Bit no.

7	6	5	4	3	2	1	0
TC	DRA	DLA	RAE	LAE	Alert Type Index		

AlertType: See clause Appendix – Alert types.

TC - ‘Text Call’: When this is cleared, the PP releases the MSF call immediately after displaying the text and alerting. The display content is kept until new display updates.

Only If TC = 1 the DRA, DLA, RAE and LAE is relevant.

DRA - ‘Display Right Arrow’: Turn on right arrow icon if set otherwise turn it off.

DLA - ‘Display Left Arrow’: Turn on left arrow icon if set otherwise turn it off.

RAE - ‘Right Arrow Key Enable’: If 1it allow keypad signalling of right arrow key, otherwise it is disabled.

LAE - ‘Left Arrow Key Enable’: If 1 it allow keypad signalling of left arrow key, otherwise it is disabled.

SetupSpec2: Has the following format:

Bit no.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	NA	SIS

SIS - 'Save In Stack': When this is set DisplayText shall be saved in the message stack of the PP, if TC bit is set ,then Save In Stack will be ignored.

NA- "No Acknowledge": When this is set no MSFRelInd is send to PC.

SetupSpec3: Has the following format:

Bit no.

7	6	5	4	3	2	1	0
F2	F1	F0	0	VT	AA	VA	MU

MU - 'MSF Up' (Automatically sets up a MSF connection) is only valid , if "SetupSpec1, TC"=1 (Text Call). Normally a MSF connection is set up by pressing <Enter> on the PP, when this is set the call is immediately set up.

VA - 'Vibrate Alvays' If 0, the vibrator will only vibrate if the user has enabled the vibrator in the PP. If 1, the vibrator will alvays vibrate independent of the local settings in the PP (if vibrate is specified in following).

AA - 'Alert Alvays' same as VA, but for tones.

VT - 'Vibrate /Tone' The vibrator follows the tone cadence if a tone is specified (both vibrator and tone)

F2, F1, F0 **- 'Function 2-0' Handling of the Light Emitting Diode or the mail icon.**

000 : This event shall not influence the LED or Mail ICON

001 : Turn off the LED

010 : Turn on the LED

011 : Start flashing the LED

100 : Turn off the Mail ICON

101 : Turn on the Mail ICON

110 : Start flashing the Mail ICON

111 : Unused or release the LED and Mail ICON for control by the PP

NOTE: The LED and the Mail ICON can be affected by the local functions !!!

SetupSpec4: Has the following format:

Bit no.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	RVC	NIC

NIC - 'Not In (voice) Call': When this is set the setup will be rejected if a voice call exist for the PP .

RVC - "Release Voice Call": When this is set the voice call will be released and the MSFsetup processed.

AlertPattern: Specifies the alert pattern to be used for the MSF setup. The value 00h indicates not present. If not present the sub-field AlertType of the field SetupSpec1 specifies the alert pattern instead. See PP specification.

🟨	0.5 sec Tone
⬜	0.5 sec Pause

Cadence Name	0		1		2		3		4		5		6		7		8		9		10	
01h - Continuos	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
02h - Internal	🟨	⬜	🟨	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟨	⬜	🟨	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟨	⬜
03h - External	🟨	🟨	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟨	🟨	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟨	🟨
04h - Alarm	Tone1		Tone2		Tone1		Tone2		Tone1		Tone2		Tone1		Tone2		Tone1		Tone2		Tone1	

01h = 'Tone continuos'. A continuos signal of max time defined by AlertTimeout , or to <OK> key is pressed. Only AlertTone 01-09 is allowed..

02h = Internal ringing in PP. Signal of max time defined by AlertTimeout , or to <OK> key is pressed..

03h = External ringing in PP. Signal of max time defined by AlertTimeout , or to <OK> key is pressed..

04h = Alarm in PP. signal of max time defined by AlertTimeout , or to <OK> key is pressed.. Always max volume.

AlertTone: Specifies the alert tone to be used for the MSF setup. The value 00h indicates not present. If not present the sub-field AlertType of the field SetupSpec1 specifies the alert tone instead. See PP specification.

00h = Alerting off.

01h - 09h = equal to tones set-up in PP

0Ah = Use tone chosen in PP.

0Bh = Vibrator.

0Ch = Key Click (Vibration together with the tone not possible). The AlertPattern must be 01h.

0Dh = Key Beep (Vibration together with the tone not possible). The AlertPattern must be 01h.

0Eh = Accept Tone (Vibration together with the tone not possible). The AlertPattern must be 01h.

0Fh = Error Tone. (Vibration together with the tone not possible). The AlertPattern must be 01h.

AlertTimeout: Specifies in seconds the duration of the alerting. The value 00h indicates not present. If not present the duration of the alert is only one cycle of the alert pattern.

DisplayTimeout: Specifies in seconds how long DisplayText shall be shown on the display of the PP before it is replaced by the standby text. Only used if TC = 0. The value 00h indicates not present. If not present the DisplayText will stay shown until a new MSF setup or until the PP is operated by its user.

12. Appendix – Proprietary discriminators

These proprietary discriminators are used to specify how the endpoints will handle broadcasts sent using the endpoint_broadcast() function.

Discriminator	Action	Alert timeout	Display timeout	Volume
80h	Text is displayed and PP is silent		240 sec.	
81h	Text is displayed and PP is alerting with pattern 2	9 sec.	240 sec.	PP setup setting
82h	Text is displayed and PP is alerting with pattern 3	9 sec.	240 sec.	PP setup setting
83h	Text is displayed and PP is alerting with pattern 4	9 sec.	240 sec.	Max.
84h	Text is displayed and PP is alerting with pattern 5	9 sec.	240 sec.	Max.
85h	Text is displayed and PP is alerting with pattern 6	9 sec.	240 sec.	Max.
86h	Text is displayed and PP is alerting with Alarm Pattern As specified under “3.4 MSF-Setup (0xC3)”	12 sec.	240 sec.	Max.
87h	Text is displayed and PP is vibrating only (always)	9 sec.	240 sec.	
D0h	Reserved			
D1h	Reserved for “Alarm user. Used in position.”			
D2h	Reserved for “Alarm position. Used in position.”			
D3h	Reserved for “Alarm resolved. Used in position.”			
D4h	Reserved for “Alarm timeout. Used in position.”			
D5h	Reserved for “You can send a			

	new alarm. Used in position."			
D6h - DBh	Reserved KIRK (for future use)			
DCh - DFh	Used internally for some SMSSetupReq & ExtenHwReq			
E0h - EFh	Reserved (COBS)			
F0h – FFh	Reserved			