ACTAtek Callback SOAP API Manual

Version 1.1

December 15, 2005

Hectrix Limited

Revision History

Revision	*Date*	*Description*	*Author*
1.1	2005/12/15	Initial Release	Sammy
1.2	2005/12/15	Log functions should throw fault	Sammy

HECTRIX LTD.

ACTAtek

ACTAtek Callback SOAP API Manual

Offices:

Asia and Rest of the World

Unit 906-911 9/Fl. Stanhope House

734-738 King's Road, North Point, Hong Kong.

Phone: 852 2319 1333

Fax: 852 2776 8997

Support Hotline: 852 2153 1131

E-mail: sales-row@hectrix.com (Sales Enquiries)

America

13372 Newport Ave suite A ,

Tustin, CA 92780, USA.

Phone: 714 505 0433

Fax: 714 544 5077

E-mail: sales-US@hectrix.com (Sales Enquiries)

Europe, Middle East & Africa

351 Pershore Road,

Edgbaston, Birmingham. B5 7RY.

U.K.

Phone +44 121 472 3991

Fax: +44 121 472 3990

E-mail: sales-EU@hectrix.com (Sales Enquiries)

Table of Contents

Chapter 1. Introduction

1.1. Prerequisite

This document is intended for software engineers who have basic knowledge in programming, XML and TCP/IP network.

1.2. Coverage

This document will cover the callback SOAP API to be implemented by an ACTAtek web service client.

1.3. SOAP

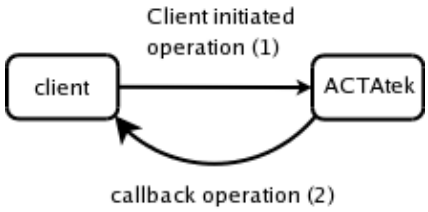
SOAP provides a XML based infrastructure for exchanging structured and typed information between peers in a decentralized, distributed environment.

1.4. Web Service Definition Language (WSDL)

WSDL, in general, is an XML file describing the details of how to call the Web Service it provides. In ACTAtek, the WSDL file can be accessed through the following URL:
http://<deviceIP>/ACTAtek.wsdl

Chapter 2. Concepts

2.1. Terminologies



Client is a program using the web service of ACTAtek. ACTAtek is said to play the role of a *server*. Hereafter, ACTAtek and server will be used interchangeably.

Under the hood, client and server has proxy classes to convert between operation calls and SOAP messages. The proxy of a client is called a stub and that of server a skeleton.

A client that makes use of the call back web service feature is also called an 'agent' (as ACTAtekAgent does so), typically for synchronizing logs.

What is web service callback?

Normally it is the client who calls the server. However, there are times when the server calls back on the client. *ACTAtek SOAP API Manual* describes the API interface of ACTAtek (1), which is used by the client. Callback web service is used by ACTAtek. In order to allow ACTAtek to call back the client, the client has to define a web service endpoint. One of the uses of ACTAtek initiating the operation is that it allows client to receive data by 'push' instead of the 'poll' model.

2.2. Conversational web service

The interaction between the client and server is also considered conversational in style[1]. The conversation starts with client's registering itself in the server and ends with its unregistering. The server keeps states about the client, such as IP, port, the registration time, time of last log sending (to avoid resending previously sent logs), connection states, etc.

As of this writing, server does not store the client endpoint URL and assumes it to be: http://[client address]:[client port]/axis/services/ACTAtekAgent. This will be fixed in future

1 In BEA's speak. See: http://dev2dev.bea.com/pub/a/2002/06/SOAPConversation.html

2.3. How do client and server identify each other?

Since the interaction is conversational, client and server must be able to uniquely identify each other. The server uses (IP, port) pairs to identify a client. The client uses the parameters passed in the specific callback operations which requires differentiating servers. For example, sync() uses registrationID to identify the server.

2.4. How to implement a client endpoint?

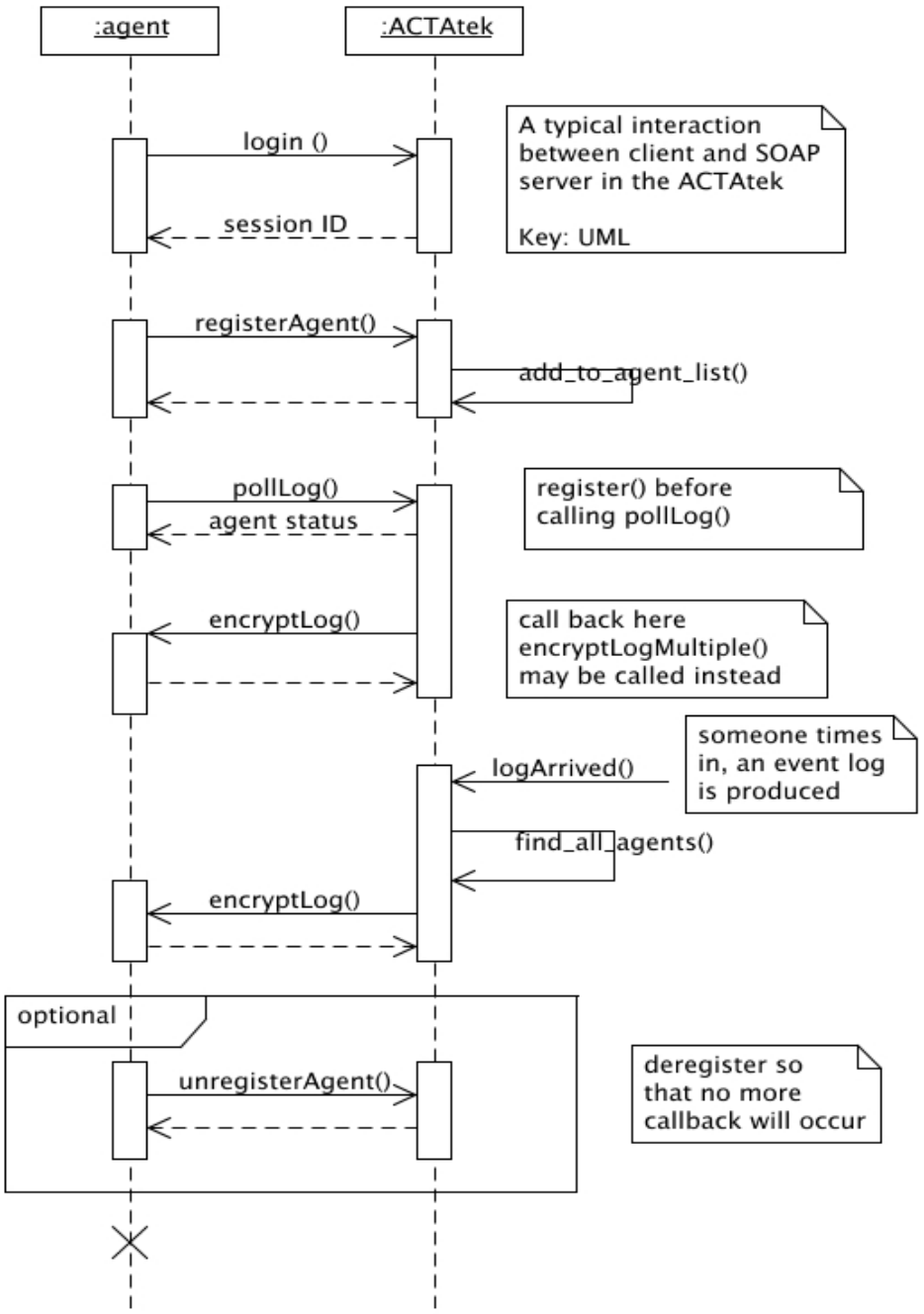
It is not necessary to implement the callback endpoint in order to use any of the ACTAtek SOAP API, except “register”. However, if one desires to retrieve logs in real-time, by the call-back mechanism, the client has to provide such an endpoint. This means setting up a web service server and implementing functions expected by ACTAtek, which will be specified in the Chapter 3. Currently, ACATekAgent has exactly implemented this interface. For code sample in different languages, see the “Getting started guide”.

2.5. A typical scenario of using callback API

As an example, ACATtekAgent interacts with ACTAtek through successive API calls (see the figure below):

1. login(): The agent first login to get authorization (sessionID) for the subsequent API calls.
2. registerAgent(): provides details such as IP, port to server to make callback possible. ACTAtek will add the agent to a list. (On registration, server will try to connect to the agent to test connectivity. If connection fails, registration will fail too.).
3. pollLog(): force ACTAtek to try to call back (in case some new event logs has not been able to be sent, due to network problems for example)
4. encryptLog(): when new event logs arrive, ACTAtek will check the list of agents registered and send the logs to them in encrypted form.
5. unregisterAgent(): when the agent no longer wants to receive logs real-time, it unregisters itself from ACTAtek.

Note that the picture is a bit of simplification. It only shows the significant interaction. At the server side, much more complicated processing takes place.


:agent
:ACTAtek
login ()
session ID
A typical interaction between client and SOAP server in the ACTAtek
Key: UML
registerAgent()
add_to_agent_list()
pollLog()
agent status
register() before calling pollLog()
encryptLog()
call back here encryptLogMultiple() may be called instead
logArrived()
someone times in, an event log is produced
find_all_agents()
encryptLog()
optional
unregisterAgent()
deregister so that no more callback will occur

Chapter 3. SOAP API Reference

Due to the language-independent nature of web services, the calling syntax would depend on the actual language for client implementation. Instead of binding to a particular language, we would describe the API in terms of the function name, parameter name, parameter type, return parameter name, return type. In case of fault, the fault string will describe the what the actual problem is. For the exact definition, please refer the actual WSDL file (agent.wsdl).

3.1. Data type Definition

3.1.1. Log

Element	Type	Nullable	Description	Example
logID	xsd:long	No	ID of event Log	32
userID	xsd:string	No	ID of the user logging in.	999
timestamp	xsd:dateTime	No	time of logging	2005-12-13T07:28:13Z
trigger	ns:eventType	No	trigger type of the event. See eventType	IN
terminalSN	xsd:string	No	ACTAtek serial number	00111DA0009D
sender	xsd:string	No	serial number of the ACTAtek from which the photo originates from. It is usually the same as 'terminalSN', it may differ only in a primary-secondary setup. In this case, 'sender' of logs originating from a secondary ACTAtek is its serial number, instead of the primary, even though it is the primary ACTAtek who actually sends the log. This field is useful to tell where the photo in 'photoPart' orginates from.	00111DA000DB
photoPart	ns:PhotoPart	No	Photo accompanying this log, if any (see PhotoPart)	

Table 1.Log Data Type Definition

3.1.2. eventType

Element	Type	Description

eventType	enum:
	{
	UNKNOWN,
	IN, OUT,
	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10,
	ID_UNKNOWN, REJECTED, ID_RESERVED,
	RESET_ALL,
	RESET_ACC, RESET_DEVICE,
	RESET_PASSWORD, RESET_DEFAULT,
	SYSTEM_LOGIN, ADMIN_LOGIN, USER_LOGIN,
	NETWORK_CHANGE,
	PATCHED, BACKUP_DB, RESTORE_DB,
	SYSTEM_RSVD, ENROLL_USER,
	UPDATE_USER, REMOVE_USER,
	SUSPEND_USER, EDIT_EVENT,
	RESET_ADMIN_PW, ADMIN_RSVD
	}	

Table 2.eventType Data Type Definition

3.1.3. PhotoPart

It contains data related to the photo of an event log.

Element	Type	Nullable	Description	Example
hasPhoto	xsd:boolean	No	Whether the log has photo.	
photoData	xsd:base64Binary	Yes	Photo accompanying this log, if any (see PhotoPart). Photo is in JPEG format.	If hasPhoto is true, this is not null and vice versa.

3.1.4. eLog

Element	Type	Nullable	Description	Example
encryptedLog	xsd:base64Binary	No	An encryped and 'serialized' Log without photo data. Photo data is in the photo field. A symmetric encryption algorithm is used, which is **Blowfish in CFB mode without padding, using an all-zero 8-byte as initial vector**. The password for encryption is the 'magic' supplied to ACTAtek when calling registerAgent(). The encrypted string is encoded in base64 format since the XSD type if base64Binary. The unencrypted string has to be parsed to reconstruct the original log. The string consists of 5 tab-separated fields: 1. timestamp: number of seconds since UTC epoch, eg: 1134626361 2. userID. eg: 1 3. eventType, eg: IN 4. terminalSN 5. sender	8iUUG/JMASGnI4Z6RGYuXhYMk6kh1pbfz7xHoCgkLENnDSvzD/QBP0g=*

Element	Type	Nullable	Description	Example
photo	ns:PhotoPart	Yes	Photo accompanying this log, if any (see PhotoPart).	If hasPhoto is true, this is not null and vice versa.

*After decrypting using magic: the log message is:

"1134626361 1 IN 00111DA001DB 00111DA001DB"

3.1.5. eLogArray

An array of eLog

Element	Type	Nullable	Description	Example
elogs	ns:eLog	Yes	zero or more eLogs. See eLog	

3.2. SOAP API

The API is documented below to advise implementors on how to process callback SOAP request from the server: how to use the parameters passed, what values to return to the server, what action the server supposes the client to take, etc.

For each function, code example written in Java is provided, using classes generated using Apache Axis 1.2 from agent.wsdl and ACTAtek.wsdl. To avoid blurring the main logic, error processing and other trivial details are not shown.

3.2.1. log

3.2.1.0.1. Semantics

This call is used solely for feeding data to the client. No processing is required. However, a typical client will use the magic to check that the log is intended for it. If so, process the log, eg. storing it in database. Besides, it is strongly advised that the client throws a SOAP fault when it fails to receive or process the log. When the server receives the fault, it will try to resend the logs later. Otherwise it will assume the client has successfully received them.

3.2.1.0.2. Remarks

Currently not used by ACTAtek. Instead encryptLog and encryptMultiple is used. Implementors may skip it.

3.2.1.0.3. In parameters

Parameters	Type	Nullable	Description	Example
magic	xsd:string	No	Client should check that it matches its own magic to prevent spoofing.	ACTAtek123
Log	ns:Log	No	Incoming log	

3.2.1.0.4. Return

Parameters	Type	Nullable	Description	Example
status	`xsd:string`	Yes	Currently not used by server. Return null is fine.	

3.2.1.0.5. Example Implementation

```
String mymagic = getConfig().getMagic().
if (mymagic != null) {
        if (!mymagic.equals(magic)) {
                throw new BadMagicFault("Magic does not match");
        }
} else {
        boolean result = processIncomingLog(log);
        if (!result) {
                throw new AgentFault("Agent Log Error");
        }
}
return null;
```

3.2.1.0.6. Related Calls

encryptLog

3.2.1.0.7. Change History

Version	Released Date	Change
1.1	15-12-2005	Initial version

3.2.2. encryptLog

3.2.2.0.1. Semantics

This call is used solely for feeding data to the client. No processing is required. However, a typical client will decrypt the log using its magic, then parse the resulting string to reconstruct the log for further processing. Besides, it is strongly advised that the client throws a SOAP fault when it fails to receive or process the log. When the server receives the fault, it will try to resend the logs later. Otherwise it will assume the client has successfully received them.

3.2.2.0.2. Remarks

Called by ACTAtek when the client is connected and a new event log is produced.

3.2.2.0.3. In parameters

Parameters	Type	Nullable	Description	Example
encryptedLog	xsd:base64Binary	No	See 3.1.4 eLog	See 3.1.4 eLog
photo	ns:PhotoPart	Yes	See 3.1.4 eLog	See 3.1.4 eLog

3.2.2.0.4. Return

Parameters	Type	Nullable	Description	Example
status	xsd:string	Yes	Currently not used by server. Return null is fine.	

3.2.2.0.5. Example Implementation

```
private void getEncryptedLog(byte[] encryptedLog, com.hectrix.agent.xsd.PhotoPart
photo) throws SoapFault {
                String magic = getConfig().getMagic();


                //decrypt the log
                String logdata = decrypt(encryptedLog, magic);


                //parse the log
                parseLog(logdata, photo);
}

private String decrypt(byte [] data, String magic) throws Exception {
                Cipher c = Cipher.getInstance("Blowfish/CFB/NoPadding");
                AlgorithmParameters parms =
AlgorithmParameters.getInstance("Blowfish");
                parms.init(new IvParameterSpec(new byte[8]));
                SecretKeySpec keyspec = new SecretKeySpec(magic.getBytes(),
"Blowfish");
                c.init(Cipher.DECRYPT_MODE, keyspec, parms);
                return new String(c.doFinal(data));
}

private Log parseLog(String logdata, com.hectrix.agent.xsd.PhotoPart photo){
                StringTokenizer st = new StringTokenizer(logdata, "\t");
                int i = 0;
                boolean success = false;
                boolean breaking = false;
                long timestamp = 0;
                String userID = null;
                String eventID = null;
                String terminalSN = null;
                String sender = null;
```

```
		while (st.hasMoreTokens()) {
			String token = st.nextToken();
			switch (i) {
			case 0:
				timestamp = Long.parseLong(token);
				break;
			case 1:
				userID = token;
				break;
			case 2:
				eventID = token;
				break;
			case 3:
				terminalSN = token;
				break;
			case 4:
				sender = token;
				success = true;
				break;
			default:
				break;
			}
			if (breaking)
				break;
			i++;
		}

		if (success) { //well-formatted log string
			// Reconstruct the log
			com.hectrix.agent.xsd.Log log = new
com.hectrix.agent.xsd.Log();
			Calendar ts = Calendar.getInstance();
			ts.setTimeInMillis(timestamp * 1000);
			com.hectrix.agent.xsd.EventType evt =
com.hectrix.agent.xsd.EventType.fromString(eventID);

			log.setLogID(0); //don't care
			log.setTerminalSN(terminalSN);
			log.setTimestamp(ts);
			log.setTrigger(evt);
```

```
                log.setUserID(userID);
                log.setSender(sender);
                log.setPhotoPart(photo);
            }
}
```

3.2.2.0.6. Related Calls

encryptLogMultiple

3.2.2.0.7. Change History

Version	Released Date	Change
1.1	15-12-2005	Initial version

3.2.3. encryptLogMultiple

3.2.3.0.1. Semantics

This call is used solely for feeding data to the client. No processing is required. However, a typical client will decrypt all the logs using its magic, then parse the resulting string to reconstruct logs for further processing. Besides, it is strongly advised that the client throws a SOAP fault when it fails to receive or process the log. When the server receives the fault, it will try to resend the logs later. Otherwise it will assume the client has successfully received them.

3.2.3.0.2. Remarks

Called by ACTAtek when it tries to send all previous logs, if any, which the client does not have. This occurs when:

1. client is first registered;
2. client has registered before, but later get disconnected and then become connected again.
3. client calls pollLog()

3.2.3.0.3. In parameters

Parameters	Type	Nullable	Description	Example
encryptedLogs	ns:eLogArray	No	An array of encrypted log. See 3.1.5 eLogArray	See 3.1.5 eLogArray

3.2.3.0.4. Return

Parameters	Type	Nullable	Description	Example
status	xsd:string	Yes	Currently not used by server. Return null is fine.	

3.2.3.0.5. Example Implementation

```
public String encryptLogMultiple(ELogArray encryptedLogs)
                                throws RemoteException {
        ELog[] elogs = encryptedLogs.getElogs();
        for (int i = 0; i < elogs.length; i++) {
                                //same processing as encyprtLog()
                                encryptLog(elogs[i].getEncryptedLog(),
elogs[i].getPhoto());
        }
        return null;
}
```

3.2.3.0.6. Related Calls

encryptLog

3.2.3.0.7. Change History

Version	Released Date	Change
1.1	15-12-2005	Initial version

3.2.4. sync

3.2.4.0.1. Semantics

This call serves as a signal sent from the server that new logs are available. No processing is required. However, a typical client may verify the ACTAtek using magic and registration ID and call getLogs() to get the latest logs.

3.2.4.0.2. Remarks

Currently not used by ACTAtek. API may subject to changes in near future.

3.2.4.0.3. In parameters

Parameters	Type	Nullable	Description	Example
magic	xsd:string	No	Client should check that it matches its own magic to prevent spoofing.	ACTAtek123
registration ID	xsd:string	No	as supplied by client when calling registerAgent()	1

3.2.4.0.4. Return

Parameters	Type	Nullable	Description	Example
status	xsd:string	Yes	Currently not used by server. Return null is fine.	

3.2.4.0.5. Example Implementation

(not available)

3.2.4.0.6. Related Calls

None

3.2.4.0.7. Change History

Version	Released Date	Change
1.1	15-12-2005	Initial version

Chapter 4. Getting Started

In the following chapters, procedures on how to set up and implement an agent will be described for some of the popular languages. Although language differs, the main steps are the same:

1. Getting ACTAtek.wsdl (server web service definition) and agent.wsdl (agent callback web service definition)
2. Create a stub from ACTAtek.wsdl
3. Create a skeleton from agent.wsdl
4. Implement a web service server (agent itself) to serve callback SOAP requests from the ACTAtek.
5. Deploy and start the agent.

Chapter 5. Creating VB.NET application

5.1. Requirements

Visual Studio .NET 2003 or 2005, IIS server 5.0 or later

5.2. Preparation

Download the WSDLs from:

http://<deviceIP>/ACTAtek.wsdl

http://<deviceIP>/agent.wsdl

5.3. Steps

1. Open Visual Studio .NET and create a ASP.NET web service project.
2. In the solution view, add web reference and choose ACTAtek.wsdl just downloaded. It should then list all web service API successfully. A client stub is now created.
3. To create the server skeleton, use wsdl.exe. Issue the following command:

   wsdl.exe /server agent.wsdl /language:VB

   Now an abstract class has been created. Add it to the project.
4. Extends this class and create an implementation for the functions.
5. Build the project and run it. In the Internet explorer that brought up, you should see the web services listed. Try to invoke sync.
6. With the server already, add code to call the server APIs such as registerAgent(), pollLog() like the following:

```
Dim MyPIService As New WebReference.ACTAtek
MyPIService.Url = "http://www.actatek.com/cgi-bin/rpcrouter"
Dim sessionID As Long = MyPIService.login("A999", 1)
```

Chapter 6. Creating JAVA Application

6.1. Requirements

- Apache Axis (Java) 1.1 or later. The following assumes Axis 1.2.
- All libraries needed by Axis

6.2. Preparation

Download the WSDLs from:

http://<deviceIP>/ACTAtek.wsdl

http://<deviceIP>/agent.wsdl

6.3. Steps

1. Create stub classes using WSDL2JAVA

   java org.apache.axis.wsdl.WSDL2Java http://<deviceIP>/ACTAtek.wsdl

   Use the stub classes, residing at the package com.hectrix.www.actatek.service to call server APIs.

   There is a ACTAtekLocator.java generated, which can be used locate the service provided by a particular location (URL, or Service Endpoint). Sample Calling Sequence of the Login function:

   ```
   ACTAtekLocator locator = new ACTAtekLocator();
   ACTAtekPortType api = locator.getACTAtek(new
   java.net.URL("http://192.168.1.100/cgi-bin/rpcrouter"));
       long sessionID = api.login("A123", "123");
   ```

2. Create skeleton classes using WSDL2JAVA

   java org.apache.axis.wsdl.WSDL2Java --server-side
   --skeletonDeploy true http://<deviceIP>/agent.wsdl

Fill in the implementation skeleton class, ACTAtekAgentImpl, residing at the package com.hectrix.www.agent.service to call server APIs.

3. **Sample Agent**

   (see sample/Java/Test.java in the archive accompanied)