Oktopous™ PIK 1.0

Copyright © 2004 Phonologies (India) Pvt Ltd.

For help with support, documentation and integration please contact:

Technical Support (Paid Support)
+91-22-22029732
oktopous@phonologies.com

Community support:
http://www.sourceforge.net/projects/oktopous/

Credits:

1. Tracy Boehrer: tboehrer@calltower.com
2. Rohan Hathiwala: rohan@phonologies.com
3. Sorabh Sodhani: sorabh@phonologies.com
4. Raj Kiran Talusani: raj@phonologies.com

License

The Phonologies Public License - Software, Version 1.0

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Phonologies (India) Pvt Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. For written permission contact Phonologies (I) Pvt Ltd, G-46 Dhanraj Mahal, Chattrapati Shivaji Marg, Mumbai -INDIA-400039
- Products derived from the software may not be called "Phonologies" or “Oktopous", nor may "Phonologies" or "Oktopous" appear in their name, without prior written permission of Phonologies.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL PHONOLOGIES OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

1. Overview

Oktopous PIK is a CCXML Interpreter and Platform Integration Kit that fully implements the CCXML 1.0 specification (Call Control Extensible Markup Language) published by the W3C. A copy of the specification can be found at http://www.w3c.org/TR/ccxml

Oktopous PIK provides a high level abstract C++ implementation of the CCXML specification that can be integrated with any underlying telephony platform to provide advanced call control and conferencing services. This document assumes that you are familiar with the activities of W3C Voice Browser Working Group.

Oktopous PIK in its 1.0 release supports Linux platform. Though the code is fully platform independent, it is not tested on other platforms (Solaris, Windows etc). You can expect a platform independent PIK very soon.

Oktopous PIK is only an interpreter interface, and does not do the actual call control operations. You need a telephony platform that supports making and receiving calls. If you want to have dialog support then you need a VoiceXML browser (ex. Phonologies InterpreXer™), SALT browser or any proprietary dialog engine. Conferencing support requires an Audio mixer or an MCU.

Oktopous PIK borrows some of the modules like data types definition and SpiderMonkey ECMAScript implementation, from the OpenVXI project (http://www.sourceforge.net/projects/openvxi).

2. Software requirements

Oktopous depends on a few third party products for XML document parsing, ECMAScript integration etc. As mentioned in the previous chapter, it also depends on the telephony platform for actual call control.

Third party software requirements

Oktopous depends on the following 3rd party software.

+ **Apache Xerces**
  Oktopous uses Xerces for XML file parsing. Xerces is developed by the Apache Software Foundation. Oktopous has been tested with Xerces 2.2.0. You may download it from http://xml.apache.org/dist/xerces-c

  Xerces C++. Copyright © 1999–2001 The Apache Software Foundation.

+ **Mozilla SpiderMonkey**
  SpiderMonkey is a ECMAScript engine developed by Mozilla.org and is released under Mozilla Public License V1.1.
  Oktopous has been tested with SpiderMonkey-1.5rc5a. You may download it from ftp://ftp.mozilla.org/pub/js

+ **PCRE**
  PCRE (Perl Compatible Regular Expressions) is used for validating transition elements in the CCXML document. PCRE is licensed under a LGPL like license. You may download it from http://www.pcre.org.

+ **Tornado**
  Tornado is an Internet I/O Library for downloading documents from the internet. It is also responsible for caching the downloaded documents. It's available under a LGPL like license. You may download it from SourceForge.net
  http://www.sourceforge.net/projects/oktopous

+ **Libcurl**
  Libcurl is the underlying HTTP client library used by Tornado. You may download it from http://curl.haxx.se

Platform requirements
Platform requirements for using Oktopous PIK.

- Redhat Linux 7.2 / 8.0 / AS 2.1
- GCC 3.0.2 or above
- Autoconf 2.52+
- Telephony Platform (SIP / H.323 / ISDN etc)
- VoiceXML Platform (Optional)
- Conferencing Mixer or MCU (Optional)

2. Installation

- Unpack the source distribution
- This will create the directory Oktopous_PIK
- Set the following environment variables
  - i. OKTOPOUSROOT points to the Oktopous_PIK directory
  - ii. XERCESCROOT points to Xerces installation directory
  - iii. SPIDERMONKEYROOT points to SpiderMonkey installation directory
  - iv. TORNADOROOT points to the Tornado installation directory
    - Now goto $OKTOPOUSROOT and execute "./configure –prefix=/usr/local"
    - Now make the distribution by executing the command "make"

3. Integration

OktoInterpreter class defines the interpreter Interface. It provides API to initialize interpreter resources and create interpreter instances.

We initialize the interpreter global resources by calling the static member function OktoInterpreter::Initialize() (to be called only once). This will give us the output queue on which all instances of interpreter will post their commands. The platform must poll on this queue to check for events.

When the platform receives a call it creates a new Interpreter instance using the function OktoInterpreter::CreateInstance(). After this we can call the Run method where we pass three parameters.

1) The URL of the CCXML application.
2) Platform generated unique session ID.
3) Fetch ID

Please note that we have to specify either one of the URL or Fetch ID. If you specify any one, then the other should be NULL. If both are specified, it will download the URL, parse it, and run that application. If neither of them are specified, Run method will return some error code.

Run() method downloads the URL and starts the EHIA thread which executes the document EHIA algorithm specified in CCXML Specification.

Example Integration

In a simple integration we need two threads.

1. Main thread
2. Polling thread

Main thread initializes the resources; creates a polling thread which polls for events on the queue returned by Initialization routine. Then it waits for incoming calls.

```
Int main(){
        OutputQueue * outQ;
        OktoInterpreter::Initialize(0,&outQ);

        //start the polling thread
        PollingThread pollThread(outQ);
        pollThread.run()

        //wait for incoming call.
        // when we receive a new call ……
        OktoInterpreter * myInterpreter = OktoInterpreter::CreateInstance();
        //store the connection id of the in coming call.
        Int connID = getConnectionID();

        //start the Interpreter
        VXIchar * url, sessionId, FetchID;
        //generate a unique session ID
        //get the url from the configuration file.
        myInterpreter->Run(url,sessionId,NULL);
```

```
        //Now post connection.ALERTING event to the interpreter.
        VXIMap * eventMap;
        Int delay = 0;

        // Now fill the map with appropriate properties and their values
        // for connection.ALERTING it will look something like this
        //  name = connection.ALERTING
        //  connid = connID
        // protocol = SIP
        // Info       = New call
        // Now post the event with delay = 0;
        myInterpreter->PostEvent(eventMap, delay);
}


PollingThread

…….

While ( outQ is not empty){

        // Now, when connection.ALERTING was posted by mainthread, the
        //CCXML application may contain a transition element to accept the call.
        //If so, Interpreter converts this accept element into a command and
        //puts it in the outQ

        // Take this command from outQ and execute it.
        // After the execution post the resulting event to interpreter.
        VXIMap * resultEvent;
        If(call accepted successfully){
                //create a connection.CONNECTED event and post it

        }
        else {
                //create a connection.FAILED event and post it.
        }

}
……
```

TODO

- As of now the Inet interface, "Tornado", supports a very simple caching mechanism.
- Logging needs to be improved a lot
- Windows and Solaris release.