FORUM NOKIA

A Brief Introduction to XML Parsing in MIDlets

Version 1.0, April 1, 2003

Java

NOKIA

Contents

Change History

1 April 2003	V1.0	Document published in Forum Nokia.

Disclaimer

The information in this document is provided "as is," with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

The phone UI images shown in this document are for illustrative purposes and do not represent any real device.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

A Brief Introduction to XML Parsing in MIDlets

Version 1.0; April 1, 2003

1 Introduction

1.1 Purpose

The following document offers an introduction to parsing of XML documents in MIDlets. It assumes familiarity with Java programming and the basics of MIDP programming by, for example, having read the Forum Nokia paper *Brief Introduction to MIDP Programming* **[MIDPPROG].** It also assumes a general knowledge of XML documents and XML parsing.

2 Background

2.1 XML

XML stands for eXtensible Markup Language **[XML]**. It is a meta-language derived from SGML and is used to store and exchange structured information; it offers developers great flexibility to define their own data formats. XML is used in many types of applications, for example, for messaging between applications. XML messages often use either UTF-8 or UTF-16 as the character set and encoding, per the XML specification **[XML] [UTF-8] [UTF-16].**

In general, XML-formatted data tends to be verbose, in order to improve the clarity for human readers. This may be an issue for applications that have access to limited memory and bandwidth resources, such as J2ME™ applications.

2.2 Java and XML

The Java Standard Edition has long offered support for XML with parsers developed by companies or open source groups. The handling of these parsers has been unified since the publication of JAXP **[JAXP],** an API for XML processing that hides the actual parser implementation behind a set of interfaces. Using JAXP, a developer can choose the parser of his or her preference or the one best suited for a particular application using the same application-level code. The parser can be upgraded to include the latest features without breaking existing code.

There are two well-established approaches for processing XML: SAX and DOM. SAX or *Simple API for XML* is an event-driven framework where the parser calls back functions in the client code while processing the XML file. SAX is lightweight and its event-driven model has reduced memory requirements making it suitable for small applications. However, it places a burden on the client code's logic to keep track of the XML parse tree. On the other hand, DOM or *Document Object Model* gives a view of the tree closer to an object model but requires higher memory consumption. A DOM tree can be easily "walked through" using the parent-to-children relationship. At the time of writing, work was underway in the Java Community Process to define an XML parser API for its inclusion in the J2ME Web Services Specification **[JSR-172]**.

A third approach is the pull parser **[XML PULL]** model that is loosely related to SAX, but instead of the API calling the application code, the client drives the parsing process. At least one such XML parser is available for MIDP.

2.3 XML and MIDP

Neither MIDP 1.0 nor MIDP 2.0 specifications provide any support for parsing or creating XML documents. Developers who want to handle XML documents have to rely on third-party libraries. Due to the limited capacity of the platform, most of these parsers support only a limited subset of features that would be expected from a J2SE parser. The use of external libraries also uses space in the jar file that could otherwise be used by the application.

In MIDP there is currently no general framework like JAXP. This means that when a developer chooses a parser, s/he ties his/her application to that specific parser. If possible, the parsing code should be located in only one place to make replacement easier if necessary.

The example developed in this document uses kXML **[KXML]** and NanoXML **[NanoXML]**. Each has benefits and tradeoffs in terms of licensing, portability, performance, and ease of use. Several other libraries have been published and could be used in a similar manner.

3 The BookReader MIDlet

3.1 Overview

A book reader MIDlet can be a useful piece of software to carry books, manuals, or other types of documents in a mobile phone for business or personal reasons. An enterprise could provide this service for technical support staff so they could access the latest version of manuals. A MIDlet could download those books from the network freely or for a fee.

In this example, XML is used to store and distribute the books. The example uses a very simple XML structure, as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <title>My Book</title>
    <metadata key="author">Forum Nokia</metadata>
    <metadata key="date">13-02-2003</metadata>
    <chapter index="0">
        <title>chapter 1</title>
        <content>Content of chapter 1</content>
    </chapter>
    <chapter index="1">
        <title>chapter 2</title>
        <content>Content of chapter 2</content>
    </chapter>
</book>
```

The application demonstrated in this document consists of a MIDlet that will read the book from its own jar file. A more advanced version of the application should be able to download the XML document using a network connection. The MIDlet simply displays a list of book chapters and displays each one as requested.

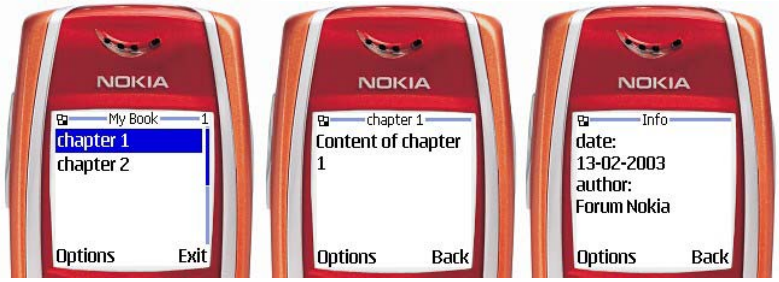

Figure 1: BookReader MIDlet user interface

3.2 BookReader MIDlet Class Diagram

Figure 2 shows the class diagram for the BookReader MIDlet:

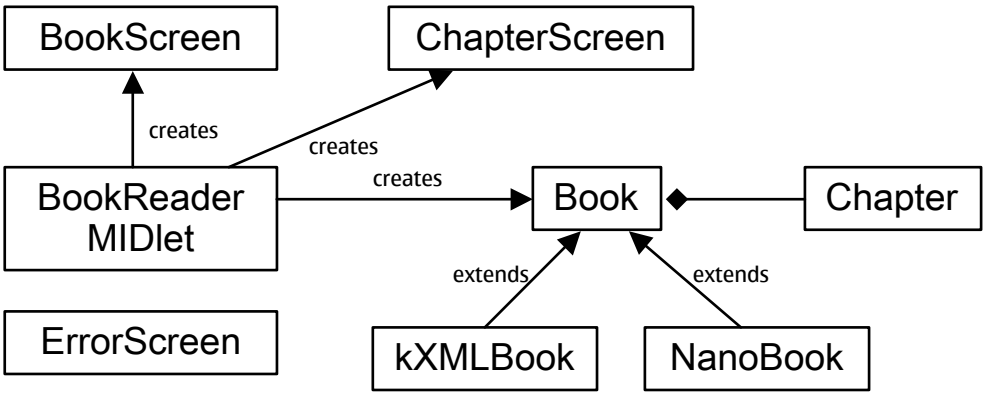

Figure 2: MIDlet class diagram

The BookReader MIDlet reads the XML document and creates an InputStream for it. In this example, the document is stored in the MIDlet's jar file but it could also be downloaded using a network connection. Book is a class that has an abstract method to parse the XML document. Different implementations can choose how to parse the document using a specific parser.

The BookScreen takes a Book object and displays a list of the chapters. Any chapter can be opened to view its contents. It also has an information item to display metadata about the Book, such as author, date, etc. When a particular chapter is open, a ChapterScreen object displays it.

3.3 BookReaderMIDlet.java

The BookReaderMIDlet class takes cares of creating the InputStream for the XML document and delegates the UI handling to the different screens. It also has state model call backs for changes initiated on the screen.

```
import java.io.InputStream;
import java.util.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;


// This is the main midlet class. It sets the basic screen,
// and handles transitions to other screens.
public class BookReaderMIDlet
    extends MIDlet
{
    private final Image logo;
    private final BookScreen bookScreen = new BookScreen(this);

    private Book book;
```

```
public BookReaderMIDlet()
{
    logo = makeImage("/logo.png");
    ErrorScreen.init(logo, Display.getDisplay(this));
}


// loads a given image by name
static Image makeImage(String filename)
{
    Image image = null;

    try
    {
        image = Image.createImage(filename);
    }
    catch (Exception e)
    {
        // use a null image instead
    }

    return image;
}


public void startApp()
{
    Displayable current = Display.getDisplay(this).getCurrent();

    if (current == null)
    {
        // the first time we are called

        String text = getAppProperty("MIDlet-Name") + "\n" +
                      getAppProperty("MIDlet-Vendor");
        Alert splashScreen = new Alert(null,
                                       text,
                                       logo,
                                       AlertType.INFO);
        splashScreen.setTimeout(3000);

        // create a 'Book' using an appropriate XML parser
        try
        {
            // Read the XML file from the JAR file. (The XML data could also
            // have easily been loaded using an HttpConnection instead.)
            InputStream in = getClass().getResourceAsStream("/book.xml");
            // You can choose here which implementation/library to use
            // e.g. NanoXML
            //book = new NanoBook();
            // e.g. kXML
            book = new kXMLBook();
```

```
            book.parse(in);
            in.close();

            // set BookScreen contents and display it
            bookScreen.setBook(book);
            Display.getDisplay(this).setCurrent(splashScreen, bookScreen);
        }
        catch (Exception e)
        {
            ErrorScreen.showError("Error parsing the book", bookScreen);
        }
    }
    else
    {
        Display.getDisplay(this).setCurrent(current);
    }
}


public void pauseApp()
{
}


public void destroyApp(boolean unconditional)
{
    notifyDestroyed();
}


// Screen callbacks


void displayBook()
{
    Display.getDisplay(this).setCurrent(bookScreen);
}


void displayChapter(int index)
{
    Vector chapters = book.getChapters();
    // normalize the chapter's index
    index = Math.max(Math.min(index, (chapters.size() - 1)), 0);

    Chapter chap = (Chapter) chapters.elementAt(index);
    ChapterScreen screen = new ChapterScreen(this, chap);

    Display.getDisplay(this).setCurrent(screen);
}
```

```
    void displayBookInfo()
    {
        Hashtable properties = book.getProperties();

        if (properties.size() > 0)
        {
            PropertiesScreen props = new PropertiesScreen(this, properties);

            Display.getDisplay(this).setCurrent(props);
        }
        else
        {
            // first time we've been called
            Alert splashScreen = new Alert(null,
                                           "No information available for this book",
                                           logo,
                                           AlertType.INFO);
            splashScreen.setTimeout(3000);

            Display.getDisplay(this).setCurrent(splashScreen, bookScreen);
        }
    }
}
```

3.4 BookScreen.java

This is the main user interface of the MIDlet. It is a `List` object containing the book's chapters. The screen has the commands Info and Exit. Selecting an item on the list will open the chapter's content.

```
import java.util.*;
import javax.microedition.lcdui.*;


// This class displays the contents of a book as a list of chapters
class BookScreen
    extends List
    implements CommandListener
{
    private final BookReaderMIDlet midlet;
    private final Command exitCommand = new Command("Exit", Command.EXIT, 1);
    private final Command infoCommand = new Command("Info", Command.SCREEN, 2);


    // sets the basic properties of the screen
    BookScreen(BookReaderMIDlet midlet)
    {
        super("Book", List.IMPLICIT);

        this.midlet = midlet;

        setCommandListener(this);
        addCommand(exitCommand);
```

```
        addCommand(infoCommand);
    }


    // when a book object is passed to the method, the list of chapters' titles
    // are added to the list
    void setBook(Book book)
    {
        setTitle(book.getTitle());

        Vector chapters = book.getChapters();
        for (int i = 0; i < chapters.size(); i++)
        {
            Chapter c = (Chapter) chapters.elementAt(i);
            append(c.getTitle(), null);
        }
    }


    public void commandAction(Command command, Displayable displayable)
    {
        if (command == List.SELECT_COMMAND)
        {
            midlet.displayChapter(getSelectedIndex());
        }
        else if (command == infoCommand)
        {
            midlet.displayBookInfo();
        }
        else  if (command == exitCommand)
        {
            midlet.destroyApp(true);
        }
    }

}
```

3.5 ChapterScreen.java

The ChapterScreen displays the contents of a section and its title. It has navigation commands to move to the next and previous chapter, as well as back to the screen menu.

```
import javax.microedition.lcdui.*;

// The chapter screen displays one chapter and allows one to navigate
// to other chapters.
class ChapterScreen
    extends Form
    implements CommandListener
{
    private final BookReaderMIDlet midlet;
    private final Command exitCommand = new Command("Exit", Command.EXIT, 1);
```

```
    private final Command backCommand = new Command("Back", Command.BACK, 1);
    private final Command nextCommand = new Command("Next", Command.SCREEN, 1);
    private final Command prevCommand = new Command("Previous", Command.SCREEN, 2);
    private final int index;


    ChapterScreen(BookReaderMIDlet midlet, Chapter chapter)
    {
        super(chapter.getTitle());

        this.midlet = midlet;

        append(new StringItem("", chapter.getContent()));
        index = chapter.getIndex();

        setCommandListener(this);
        addCommand(exitCommand);
        addCommand(backCommand);
        addCommand(nextCommand);
        addCommand(prevCommand);
    }


    public void commandAction(Command command, Displayable displayable)
    {
        if (command == exitCommand)
        {
            midlet.destroyApp(true);
        }
        else if (command == backCommand)
        {
            midlet.displayBook();
        }
        else if (command == prevCommand)
        {
            midlet.displayChapter(index - 1);
        }
        else if (command == nextCommand)
        {
            midlet.displayChapter(index + 1);
        }
    }
}
```

3.6 PropertiesScreen.java

This `Form` displays a list of the properties of the book included as metadata tags in the XML document.

```
import java.util.*;
import javax.microedition.lcdui.*;

class PropertiesScreen
```

```
     extends Form
     implements CommandListener
{
    private final BookReaderMIDlet midlet;
    private final Command exitCommand = new Command("Exit", Command.EXIT, 1);
    private final Command backCommand = new Command("Back", Command.BACK, 1);


    public PropertiesScreen(BookReaderMIDlet midlet, Hashtable properties)
    {
        super("Info");
        this.midlet = midlet;

        // displays the list of the book's properties
        Enumeration keys = properties.keys();
        while (keys.hasMoreElements())
        {
            String key = (String) keys.nextElement();
            String value = (String) properties.get(key);
            append(new StringItem(key, value));
        }

        setCommandListener(this);
        addCommand(exitCommand);
        addCommand(backCommand);
    }


    public void commandAction(Command command, Displayable displayable)
    {
        if (command == exitCommand)
        {
            midlet.destroyApp(true);
        }
        else if (command == backCommand)
        {
            midlet.displayBook();
        }
    }

}
```

3.7 ErrorScreen.java

The ErrorScreen is used in MIDlet UI components to handle error situations, where a temporary error dialog needs to be displayed to the end user.

```
import javax.microedition.lcdui.*;

class ErrorScreen
     extends Alert
{
    private static Image image;
    private static Display display;
    private static ErrorScreen instance = null;


    private ErrorScreen()
    {
        super("Error");
        setType(AlertType.ERROR);
        setTimeout(5000);
        setImage(image);
    }


    static void init(Image img, Display disp)
    {
        image = img;
        display = disp;
    }


    static void showError(String message, Displayable next)
    {
        if (instance == null)
        {
            instance = new ErrorScreen();
        }
        instance.setString(message);
        display.setCurrent(instance, next);
    }
}
```

3.8 Book.java

Book is a simple abstract data class that contains information about the book such as title, properties, and a collection of chapters. Descendent classes should implement the `parse` method to read the XML document using a particular library.

```
import java.io.*;
import java.util.*;


// An abstract data class encapsulating the concept of a Book.
// Concrete implementations define their own XML parsing.
abstract class Book
{
    protected String title = "No title";
    protected final Hashtable properties = new Hashtable();
    protected final Vector chapters = new Vector();


    Book()
    {
    }


    String getTitle()
    {
        return title;
    }


    // return a hashtable of properties which are represented
    // as <metadata> elements in the XML file
    Hashtable getProperties()
    {
        return properties;
    }


    // return a vector of chapter objects
    Vector getChapters()
    {
        return chapters;
    }


    // Loads the book from an appropriate input stream.
    // Concrete implementations may use different XML parsers.
    abstract void parse(InputStream in)
        throws Exception;
}
```

3.9 Chapter.java

The Chapter class represents a book's chapter. It is a very simple data container.

```
class Chapter
{
    private final String title;
    private final String content;
    private final int index;


    Chapter(String title, String content, int index)
    {
        this.title = title;
        this.content = content;
        this.index = index;
    }


    String getTitle()
    {
        return title;
    }


    String getContent()
    {
        return content;
    }


    int getIndex()
    {
        return index;
    }
}
```

3.10 NanoBook.java

NanoBook is a Book class that uses NanoXML **[NanoXML]** for XML parsing. NanoXML is an open source parser that is very small (about 14 KB after obfuscation), with no DTD validation support. Version 2.0 of NanoXML is offered in three branches, NanoXML/Lite being the one optimized for small devices. However, NanoXML/Lite is not CLDC compatible. At the time of this writing, there is a patched version that removes the incompatibilities. NanoXML offers a DOM-like structure that makes it very easy to traverse the tree to build the data objects. In this particular example, the parse method will read the title, metadata, and chapter tags, and will build the chapter object using the chapter's children tags.

```
import java.io.*;
import java.util.*;
import nanoxml.kXMLElement;


// Class NanoBook extends Book and implements the load method
```

```
// using the NanoXML parser
class NanoBook
    extends Book
{
    void parse(InputStream in)
        throws Exception
    {
        // nano xml uses the kXMLElement to do the parsing and traversing
        kXMLElement xml = new kXMLElement();
        xml.parseFromReader(new InputStreamReader(in));

        // traverse the children
        Enumeration enum = xml.enumerateChildren();
        while (enum.hasMoreElements())
        {
            kXMLElement current = (kXMLElement) enum.nextElement();
            String tagname = current.getTagName();

            // set the title
            if ("title".equals(tagname))
            {
                title = current.getContents();
            }
            // set any metadata key/value pair
            else if ("metadata".equals(tagname))
            {
                String key = current.getProperty("key");

                properties.put(key, current.getContents());
            }
            // If a chapter is found continue the parsing down the tree
            else if ("chapter".equals(tagname))
            {
                Enumeration chapterEnum = current.enumerateChildren();
                String title = "[No title]";
                String content = "[No content]";
                int location = current.getProperty("index", -1);
                // parse the chapter
                while (chapterEnum.hasMoreElements())
                {
                    kXMLElement currentChapter = (kXMLElement)
chapterEnum.nextElement();

                    tagname = currentChapter.getTagName();
                    if ("title".equals(tagname))
                    {
                        title = currentChapter.getContents();
                    }
                    else if ("content".equals(tagname))
                    {
                        content = currentChapter.getContents();
                    }
```

```
                }

                // create the chapter object and insert it at the right location
                Chapter chap = new Chapter(title, content, location);
                chapters.insertElementAt(chap,
                                         (location >= 0) ? location : (chapters.size()
- 1));
            }
        }
    }
}
```

3.11 kXMLBook.java

This class extends the Book class doing the parsing with the kXML library. kXML has been designed specifically for J2ME and no patched version is necessary. kXML version 2 uses the new XML pull parser interfaces, therefore it is possible to replace kXML by another pull parser implementation. At the time of this writing, kXML is the only pull parser version available for J2ME. kXML does not support DTD validation.

Using the pull parser technique means that the software drives the parsing. In this case it basically iterates over the tree and finds the title, metadata, and chapter tags. The iteration is done using the `next()`method, which moves the parser to the next element on the XML tree, be it a start or end tag, or text content. Notice that the `next()` method doesn't care about the parent/child relationship and it is the application's responsibility to be aware of the parsing context.

```
import org.kxml2.io.*;
import java.io.*;
import java.util.*;


// Class kXMLBook extends Book and implements the load method
// using the kXML 2 parser
class kXMLBook
    extends Book
{
    void parse(InputStream in)
        throws Exception
    {
        KXmlParser parser = new KXmlParser();
        parser.setInput(new InputStreamReader(in));

        int eventType = KXmlParser.START_TAG;

        while (eventType != KXmlParser.END_DOCUMENT)
        {
            eventType = parser.next();
            if (eventType == KXmlParser.START_TAG)
            {
                if (parser.getName().equals("title"))
                {
                    parser.next();
                    this.title = parser.getText();
```

```
            }
            else if (parser.getName().equals("metadata"))
            {
                String key = parser.getAttributeValue(null, "key");
                parser.next();
                String value = parser.getText();
                properties.put(key, value);
            }
            else if (parser.getName().equals("chapter"))
            {
                parseChapter(parser);
            }
        }

    }

}


// parse a chapter
void parseChapter(KXmlParser parser)
    throws Exception
{
    parser.require(KXmlParser.START_TAG, "", "chapter");

    String title = "[No title]";
    String content = "[No content]";

    // get the chapter number
    int location = Integer.parseInt(parser.getAttributeValue(null, "index"));
    int eventType = parser.next();

    // parse until a chapter tag is hit
    while (!"chapter".equals(parser.getName()))
    {
        if (eventType == KXmlParser.START_TAG)
        {
            if (parser.getName().equals("title"))
            {
                parser.next();
                title = parser.getText();
            }
            else if (parser.getName().equals("content"))
            {
                parser.next();
                content = parser.getText();
            }
        }
        eventType = parser.next();
    }

    Chapter chap = new Chapter(title, content, location);
```

```
        chapters.insertElementAt(chap, (location >= 0) ? location : chapters.size()
- 1);
        parser.require(parser.END_TAG, "", "chapter");
    }
}
```

3.12 BookReaderMIDlet.jad

This is the application descriptor for the BookReader MIDlet.

```
MIDlet-Name: BookReader
MIDlet-Version: 1.0
MIDlet-Vendor: Forum Nokia
MIDlet-Jar-URL: BookReader.jar
MIDlet-Jar-Size: 23315
MIDlet-Icon: /logo.png
MIDlet-1: BookReader, ,BookReaderMIDlet
```

After removing the unused classes and obfuscation of the MIDlet, NanoXML parser classes use 14391 bytes and the kXML parser classes use 14004 bytes (uncompressed size).

4 References

JSR-172 *J2ME Web Services Specification*
http://www.jcp.org/en/jsr/detail?id=172

KXML http://www.kxml.org

MIDPPROG *Brief Introduction to MIDP Programming*
Forum Nokia, 2002
http://www.forum.nokia.com

NanoXML http://nanoxml.n3.net/

http://www.ericgiguere.com/nanoxml/kNanoXML.zip

UTF-8 *IETF RFC 2279: UTF-8, a transformation format of ISO 10646*
1998
http://www.ietf.org/rfc/rfc2279.txt

UTF-16 *IETF RFC 2781: UTF-16, an encoding of ISO 10646*
2000
http://www.ietf.org/rfc/rfc2781.txt

XML *Extensible Markup Language (XML) 1.0 (Second Edition)*
W3C Recommendation, 6 October 2000
http://www.w3.org/TR/REC-xml

XML PULL *Common API for XML Pull Parsing*
http://www.xmlpull.org/

Build Test Sell

Developing and marketing mobile applications with Nokia

Go to Forum.Nokia.com

Forum.Nokia.com provides the tools and resources you need for content and application development as well as the channels for sales to operators, enterprises, and consumers.

Forum.Nokia.com

Download tools and emulators

Forum.Nokia.com/tools has links to tools from Nokia and other industry leaders including Borland, Adobe, AppForge, Macromedia, Metrowerks, and Sun.

Forum.Nokia.com/tools

Get documents and specifications

The documents area contains useful white papers, FAQs, tutorials, and APIs for Symbian OS and Series 60 Platform, J2ME, messaging (including MMS), and other technologies. Forum.Nokia.com/devices lists detailed technical specifications for Nokia devices.

Forum.Nokia.com/documents
Forum.Nokia.com/devices

Test your application and get support

Forum Nokia offers free and fee-based support that provides you with direct access to Nokia engineers and equipment and connects you with other developers around the world. The Nokia OK testing program enables your application to enjoy premium placement in Nokia's sales channels.

Forum.Nokia.com/support
Forum.Nokia.com/ok

Market through Nokia channels

Go to Forum.Nokia.com/business to learn about all of the marketing channels open to you, including Nokia Tradepoint, an online B2B marketplace.

Forum.Nokia.com/business

Reach buyers around the globe

Place your applications in Nokia Tradepoint and they're available to dozens of buying organizations around the world, ranging from leading global operators and enterprises to regional operators and XSPs. Your company and applications will also be considered for the regional Nokia Software Markets as well as other global and regional opportunities, including personal introductions to operators, on-device and in-box placement, and participation in invitation-only events around the world.

Forum.Nokia.com/business