**软著源程序代码:**

**提供该软件代码的开头和完整结尾，整体不能太零碎，要求最少提供60页（前30页和后30页）, 可以按照标准每页50行操作；若全部代码不足的，需提供全部源代码；**

**注：源代码中 ，不能出现大段的文字及空格。**

**下页为示例：**

package Android.zhizhi;

import java.io.EOFException;

import java.io.IOException;

import java.io.OutputStream;

public class AsciiOutputStream extends OutputStream

{

private boolean breakOnNonAscii;

private int ascii = 0; private int non\_ascii = 0;

private int linelen = 0;

private boolean longLine = false;

private boolean badEOL = false;

private boolean checkEOL = false;

private int lastb = 0;

private int ret = 0;

public AsciiOutputStream(boolean breakOnNonAscii, boolean encodeEolStrict) {

this.breakOnNonAscii = breakOnNonAscii;

this.checkEOL = ((encodeEolStrict) && (breakOnNonAscii));

}

public void write(int b) throws IOException {

check(b);

}

public void write(byte[] b) throws IOException {

write(b, 0, b.length);//调用write公共方法

}

public void write(byte[] b, int off, int len) throws IOException {

len += off;

for (int i = off; i < len; i++)

check(b[i]);

}

private final void check(int b) throws IOException {

b &= 255;

if ((this.checkEOL) && (((this.lastb == 13) && (b != 10)) || ((this.lastb != 13) && (b == 10))))

{//判断消息类型

this.badEOL = true;

}if ((b == 13) || (b == 10)) {

this.linelen = 0;

} else {

this.linelen += 1;

if (this.linelen > 998)

this.longLine = true;

}

if (MimeUtility.nonascii(b)) {

this.non\_ascii += 1;//视频播放量累加

if (this.breakOnNonAscii) {

this.ret = 3;

throw new EOFException();//抛出异常

}

} else {

this.ascii += 1;

}this.lastb = b;

}

public int getAscii()

{

if (this.ret != 0) {

return this.ret;//返回编码

}

if (this.badEOL)

return 3;

if (this.non\_ascii == 0)

{

if (this.longLine) {

return 2;//返回值

}

return 1; //返回值

}

if (this.ascii > this.non\_ascii)//编码方式判断

return 2;

return 3;

}

}

package Android.zhizhi;

public class ContentDisposition

{

private String disposition;

private ParameterList list;

public ContentDisposition()

{

}

public ContentDisposition(String disposition, ParameterList list)

{

this.disposition = disposition;

this.list = list;

}

public ContentDisposition(String s)

throws ParseException

{

HeaderTokenizer h = new HeaderTokenizer(s, "()<>@,;:\\\"\t []/?=");

HeaderTokenizer.Token tk = h.next();

if (tk.getType() != -1) {

throw new ParseException("Expected disposition, got " + tk.getValue());//记录异常信息

}

this.disposition = tk.getValue();

String rem = h.getRemainder();

if (rem != null)

this.list = new ParameterList(rem);初始化列表

}

public String getDisposition()

{

return this.disposition;

}

public String getParameter(String name)

{

if (this.list == null) {//校验视频列表是否为空

return null;

}

return this.list.get(name);

}

public ParameterList getParameterList()

{

return this.list;//返回列表数据

}

public void setDisposition(String disposition)

{

this.disposition = disposition;

}

public void setParameter(String name, String value)

{

if (this.list == null) {//校验视频列表是否为空

this.list = new ParameterList();

}

this.list.set(name, value);

}

public void setParameterList(ParameterList list)

{

this.list = list;

}

public String toString()

{

if (this.disposition == null) {

return null;

}

if (this.list == null) {

return this.disposition;

}

StringBuffer sb = new StringBuffer(this.disposition);

sb.append(this.list.toString(sb.length() + 21));

return sb.toString();

}

}

package Android.zhizhi;

public class ContentType

{

private String primaryType;

private String subType;

private ParameterList list;

public ContentType()

{

}

public ContentType(String primaryType, String subType, ParameterList list)

{

this.primaryType = primaryType;

this.subType = subType;

this.list = list;

}

public ContentType(String s)

throws ParseException

{

HeaderTokenizer h = new HeaderTokenizer(s, "()<>@,;:\\\"\t []/?=");

HeaderTokenizer.Token tk = h.next();

if (tk.getType() != -1) {

throw new ParseException("Expected MIME type, got " + tk.getValue());

}

this.primaryType = tk.getValue();

tk = h.next();

if ((char)tk.getType() != '/') {

throw new ParseException("Expected '/', got " + tk.getValue());

}

tk = h.next();

if (tk.getType() != -1) {

throw new ParseException("Expected MIME subtype, got " + tk.getValue());

}

this.subType = tk.getValue();

String rem = h.getRemainder();

if (rem != null)

this.list = new ParameterList(rem);

}

public String getPrimaryType()

{

return this.primaryType;

}

public String getSubType()

{

return this.subType;

}

public String getBaseType()

{

return this.primaryType + '/' + this.subType;

}

public String getParameter(String name)

{

if (this.list == null) {

return null;

}

return this.list.get(name);

}

public ParameterList getParameterList()

{

return this.list;

}

public void setPrimaryType(String primaryType)

{

this.primaryType = primaryType;

}

public void setSubType(String subType)

{

this.subType = subType;

}

public void setParameter(String name, String value)

{

if (this.list == null) {

this.list = new ParameterList();

}

this.list.set(name, value);

}

public void setParameterList(ParameterList list)

{

this.list = list;

}

public String toString()

{

if ((this.primaryType == null) || (this.subType == null)) {

return null;

}

StringBuffer sb = new StringBuffer();

sb.append(this.primaryType).append('/').append(this.subType);

if (this.list != null)

{

sb.append(this.list.toString(sb.length() + 14));

}

return sb.toString();

}

public boolean match(String s)

{

try

{

return match(new ContentType(s)); } catch (ParseException pex) {

}

return false;

}

}

package Android.zhizhi;

public class HeaderTokenizer

{

private String string;

private boolean skipComments;

private String delimiters;

private int currentPos;

private int maxPos;

private int nextPos;

private int peekPos;

public static final String RFC822 = "()<>@,;:\\\"\t .[]";

public static final String MIME = "()<>@,;:\\\"\t []/?=";

private static final Token EOFToken = new Token(-4, null);

public HeaderTokenizer(String header, String delimiters, boolean skipComments)

{

this.string = (header == null ? "" : header);

this.skipComments = skipComments;

this.delimiters = delimiters;

this.currentPos = (this.nextPos = this.peekPos = 0);

this.maxPos = this.string.length();

}

public HeaderTokenizer(String header, String delimiters)

{

this(header, delimiters, true);

}

public HeaderTokenizer(String header)

{

this(header, "()<>@,;:\\\"\t .[]");

}

public Token next()

throws ParseException

{

return next('\000', false);

}

Token next(char endOfAtom)

throws ParseException

{

return next(endOfAtom, false);

}

Token next(char endOfAtom, boolean keepEscapes)

throws ParseException

{

this.currentPos = this.nextPos;

Token tk = getNext(endOfAtom, keepEscapes);

this.nextPos = (this.peekPos = this.currentPos);

return tk;

}

public Token peek()

throws ParseException

{

this.currentPos = this.peekPos;

Token tk = getNext('\000', false);

this.peekPos = this.currentPos;

return tk;

}

public String getRemainder()

{

return this.string.substring(this.nextPos);

}

private Token getNext(char endOfAtom, boolean keepEscapes)

throws ParseException

{

if (this.currentPos >= this.maxPos) {

return EOFToken;

}

if (skipWhiteSpace() == -4) {

return EOFToken;

}

boolean filter = false;

char c = this.string.charAt(this.currentPos);

while (c == '(')

{

int start = ++this.currentPos; int nesting = 1;

for (; (nesting > 0) && (this.currentPos < this.maxPos);

this.currentPos += 1) {

c = this.string.charAt(this.currentPos);

if (c == '\\') {

this.currentPos += 1;

filter = true;

} else if (c == '\r') {

filter = true;

} else if (c == '(') {

nesting++;

} else if (c == ')') {

nesting--;

}

}

if (nesting != 0) {

throw new ParseException("Unbalanced comments");

}

if (!this.skipComments)

{

String s;

String s;

if (filter)

s = filterToken(this.string, start, this.currentPos - 1, keepEscapes);

else {

s = this.string.substring(start, this.currentPos - 1);

}

return new Token(-3, s);

}

if (skipWhiteSpace() == -4)

return EOFToken;

c = this.string.charAt(this.currentPos);

}

if (c == '"') {

this.currentPos += 1;

return collectString('"', keepEscapes);

}

if ((c < ' ') || (c >= '') || (this.delimiters.indexOf(c) >= 0)) {

if ((endOfAtom > 0) && (c != endOfAtom))

{

return collectString(endOfAtom, keepEscapes);

}

this.currentPos += 1;

char[] ch = new char[1];

ch[0] = c;

return new Token(c, new String(ch));

}

for (int start = this.currentPos; this.currentPos < this.maxPos; this.currentPos += 1) {

c = this.string.charAt(this.currentPos);

if ((c < ' ') || (c >= '') || (c == '(') || (c == ' ') || (c == '"') || (this.delimiters.indexOf(c) >= 0))

{

if ((endOfAtom <= 0) || (c == endOfAtom)) {

break;

}

this.currentPos = start;

return collectString(endOfAtom, keepEscapes);

}

}

return new Token(-1, this.string.substring(start, this.currentPos));

}

private Token collectString(char eos, boolean keepEscapes)

throws ParseException

{

boolean filter = false;

for (int start = this.currentPos; this.currentPos < this.maxPos; this.currentPos += 1) {

char c = this.string.charAt(this.currentPos);

if (c == '\\') {

this.currentPos += 1;

filter = true;

} else if (c == '\r') {

filter = true;

} else if (c == eos) {

this.currentPos += 1;

String s;

String s;

if (filter)

s = filterToken(this.string, start, this.currentPos - 1, keepEscapes);

else {

s = this.string.substring(start, this.currentPos - 1);

}

if (c != '"') {

s = trimWhiteSpace(s);

this.currentPos -= 1;

}

return new Token(-2, s);

}

}

if (eos == '"')

throw new ParseException("Unbalanced quoted string");

String s;

if (filter)

s = filterToken(this.string, start, this.currentPos, keepEscapes);

else

s = this.string.substring(start, this.currentPos);

String s = trimWhiteSpace(s);

return new Token(-2, s);

}

private int skipWhiteSpace()

{

for (; this.currentPos < this.maxPos; this.currentPos += 1)

{

char c;

if (((c = this.string.charAt(this.currentPos)) != ' ') && (c != '\t') && (c != '\r') && (c != '\n'))

{

return this.currentPos; }

}return -4;

}

private static String trimWhiteSpace(String s)

{

char c;

for (int i = s.length() - 1; (i >= 0) && (

((c = s.charAt(i)) == ' ') || (c == '\t') || (c == '\r') || (c == '\n')); i--);

if (i <= 0) {

return "";

}

return s.substring(0, i + 1);

}

private static String filterToken(String s, int start, int end, boolean keepEscapes)

{

StringBuffer sb = new StringBuffer();

boolean gotEscape = false;

boolean gotCR = false;

for (int i = start; i < end; i++) {

char c = s.charAt(i);

if ((c == '\n') && (gotCR))

{

gotCR = false;

}

else

{

gotCR = false;

if (!gotEscape)

{

if (c == '\\')

gotEscape = true;

else if (c == '\r')

gotCR = true;

else {

sb.append(c);

}

}

else

{

if (keepEscapes)

sb.append('\\');

sb.append(c);

gotEscape = false;

}

}

}

return sb.toString();

}

public static class Token

{

private int type;

private String value;

public static final int ATOM = -1;

public static final int QUOTEDSTRING = -2;

public static final int COMMENT = -3;

public static final int EOF = -4;

public Token(int type, String value)

{

this.type = type;

this.value = value;

}

public int getType()

{

return this.type;

}

public String getValue()

{

return this.value;

}

}

}

package Android.zhizhi;

import com.sun.mail.util.PropUtil;

import java.io.UnsupportedEncodingException;

import java.net.InetAddress;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.List;

import java.util.Locale;

import java.util.StringTokenizer;

import Android.zhizhi.Address;

import Android.zhizhi.Session;

public class InternetAddress extends Address

implements Cloneable

{

protected String address;

protected String personal;

protected String encodedPersonal;

private static final long serialVersionUID = -7507595530758302903L;

private static final boolean ignoreBogusGroupName = PropUtil.getBooleanSystemProperty("zhizhi.mime.address.ignorebogusgroupname", true);

private static final String rfc822phrase = "()<>@,;:\\\"\t .[]".replace(' ', '\000').replace('\t', '\000');

private static final String specialsNoDotNoAt = "()<>,;:\\\"[]";

private static final String specialsNoDot = "()<>,;:\\\"[]@";

public InternetAddress()

{

}

public InternetAddress(String address)

throws AddressException

{

InternetAddress[] a = parse(address, true);

if (a.length != 1) {

throw new AddressException("Illegal address", address);

}

this.address = a[0].address;

this.personal = a[0].personal;

this.encodedPersonal = a[0].encodedPersonal;

}

public InternetAddress(String address, boolean strict)

throws AddressException

{

this(address);

if (strict)

if (isGroup())

getGroup(true);

else

checkAddress(this.address, true, true);

}

public InternetAddress(String address, String personal)

throws UnsupportedEncodingException

{

this(address, personal, null);

}

public InternetAddress(String address, String personal, String charset)

throws UnsupportedEncodingException

{

this.address = address;

setPersonal(personal, charset);

}

public Object clone()

{

InternetAddress a = null;

try {

a = (InternetAddress)super.clone(); } catch (CloneNotSupportedException e) {

}

return a;

}

public String getType()

{

return "rfc822";

}

public void setAddress(String address)

{

this.address = address;

}

public void setPersonal(String name, String charset)

throws UnsupportedEncodingException

{

this.personal = name;

if (name != null)

this.encodedPersonal = MimeUtility.encodeWord(name, charset, null);

else

this.encodedPersonal = null;

}

public void setPersonal(String name)

throws UnsupportedEncodingException

{

this.personal = name;

if (name != null)

this.encodedPersonal = MimeUtility.encodeWord(name);

else

this.encodedPersonal = null;

}

public String getAddress()

{

return this.address;

}

public String getPersonal()

{

if (this.personal != null) {

return this.personal;

}

if (this.encodedPersonal != null) {

try {

this.personal = MimeUtility.decodeText(this.encodedPersonal);

return this.personal;

}

catch (Exception ex)

{

return this.encodedPersonal;

}

}

return null;

}

{

private int type;

private String value;

public static final int ATOM = -1;

public static final int QUOTEDSTRING = -2;

public static final int COMMENT = -3;

public static final int EOF = -4;

public Token(int type, String value)

{

this.type = type;

this.value = value;

}

public int getType()

{

return this.type;

}

public String getValue()

{

return this.value;

}

}

}

package Android.zhizhi;

import com.sun.mail.util.PropUtil;

import java.io.UnsupportedEncodingException;

import java.net.InetAddress;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.List;

import java.util.Locale;

import java.util.StringTokenizer;

import Android.zhizhi.Address;

import Android.zhizhi.Session;

public class InternetAddress extends Address

implements Cloneable

{

protected String address;

protected String personal;

protected String encodedPersonal;

private static final long serialVersionUID = -7507595530758302903L;

private static final boolean ignoreBogusGroupName = PropUtil.getBooleanSystemProperty("zhizhi.mime.address.ignorebogusgroupname", true);

private static final String rfc822phrase = "()<>@,;:\\\"\t .[]".replace(' ', '\000').replace('\t', '\000');

private static final String specialsNoDotNoAt = "()<>,;:\\\"[]";

private static final String specialsNoDot = "()<>,;:\\\"[]@";

public InternetAddress()

{

}

public InternetAddress(String address)

throws AddressException

{

InternetAddress[] a = parse(address, true);

if (a.length != 1) {

throw new AddressException("Illegal address", address);

}

this.address = a[0].address;

this.personal = a[0].personal;

this.encodedPersonal = a[0].encodedPersonal;

}

public InternetAddress(String address, boolean strict)

throws AddressException

{

this(address);

if (strict)

if (isGroup())

getGroup(true);

else

checkAddress(this.address, true, true);

}

public InternetAddress(String address, String personal)

throws UnsupportedEncodingException

{

this(address, personal, null);

}

public InternetAddress(String address, String personal, String charset)

throws UnsupportedEncodingException

{

this.address = address;

setPersonal(personal, charset);

}

public Object clone()

{

InternetAddress a = null;

try {

a = (InternetAddress)super.clone(); } catch (CloneNotSupportedException e) {

}

return a;

}

public String getType()

{

return "rfc822";

}

public void setAddress(String address)

{

this.address = address;

}

public void setPersonal(String name, String charset)

throws UnsupportedEncodingException

{

this.personal = name;

if (name != null)

this.encodedPersonal = MimeUtility.encodeWord(name, charset, null);

else

this.encodedPersonal = null;

}

public String toString()

{

if ((this.encodedPersonal == null) && (this.personal != null))

try {

this.encodedPersonal = MimeUtility.encodeWord(this.personal);

} catch (UnsupportedEncodingException ex) {

}

if (this.encodedPersonal != null)

return quotePhrase(this.encodedPersonal) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

public String toUnicodeString()

{

String p = getPersonal();

if (p != null)

return quotePhrase(p) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

public String toString()

{

if ((this.encodedPersonal == null) && (this.personal != null))

try {

this.encodedPersonal = MimeUtility.encodeWord(this.personal);

} catch (UnsupportedEncodingException ex) {

}

if (this.encodedPersonal != null)

return quotePhrase(this.encodedPersonal) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

public String toUnicodeString()

{

String p = getPersonal();

if (p != null)

return quotePhrase(p) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

public String toString()

{

if ((this.encodedPersonal == null) && (this.personal != null))

try {

this.encodedPersonal = MimeUtility.encodeWord(this.personal);

} catch (UnsupportedEncodingException ex) {

}

if (this.encodedPersonal != null)

return quotePhrase(this.encodedPersonal) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

public String toUnicodeString()

{

String p = getPersonal();

if (p != null)

return quotePhrase(p) + " <" + this.address + ">";

if ((isGroup()) || (isSimple())) {

return this.address;

}

return "<" + this.address + ">";

}

private static String quotePhrase(String phrase)

{

int len = phrase.length();

boolean needQuoting = false;

for (int i = 0; i < len; i++) {

char c = phrase.charAt(i);

if ((c == '"') || (c == '\\'))

{

StringBuffer sb = new StringBuffer(len + 3);

sb.append('"');

for (int j = 0; j < len; j++) {

char cc = phrase.charAt(j);

if ((cc == '"') || (cc == '\\'))

{

sb.append('\\');

}sb.append(cc);

}

sb.append('"');

return sb.toString();

}if (((c < ' ') && (c != '\r') && (c != '\n') && (c != '\t')) || (c >= '') || (rfc822phrase.indexOf(c) >= 0))

{

needQuoting = true;

}

}

if (needQuoting) {

StringBuffer sb = new StringBuffer(len + 2);

sb.append('"').append(phrase).append('"');

return sb.toString();

}

return phrase;

}

private static String unquote(String s) {

if ((s.startsWith("\"")) && (s.endsWith("\""))) {

s = s.substring(1, s.length() - 1);

if (s.indexOf('\\') >= 0) {

StringBuffer sb = new StringBuffer(s.length());

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

if ((c == '\\') && (i < s.length() - 1))

c = s.charAt(++i);

sb.append(c);

}

s = sb.toString();

}

}

return s;

}

public boolean equals(Object a)

{

if (!(a instanceof InternetAddress)) {

return false;

}

String s = ((InternetAddress)a).getAddress();

if (s == this.address)

return true;

public static String toString(Address[] addresses)

{

return toString(addresses, 0);

}

public static String toString(Address[] addresses, int used)

{

if ((addresses == null) || (addresses.length == 0)) {

return null;

}

StringBuffer sb = new StringBuffer();

for (int i = 0; i < addresses.length; i++) {

if (i != 0) {

sb.append(", ");

used += 2;

}

String s = addresses[i].toString();

int len = lengthOfFirstSegment(s);

if (used + len > 76) {

sb.append("\r\n\t");

used = 8;

}

sb.append(s);

used = lengthOfLastSegment(s, used);

}

return sb.toString();

}

private static int lengthOfFirstSegment(String s)

{

int pos;

if ((pos = s.indexOf("\r\n")) != -1) {

return pos;

}

return s.length();

}

private static int lengthOfLastSegment(String s, int used)

{

int pos;

if ((pos = s.lastIndexOf("\r\n")) != -1) {

return s.length() - pos - 2;

}

return s.length() + used;

}

public static InternetAddress getLocalAddress(Session session)

{

try

{

return \_getLocalAddress(session); } catch (SecurityException sex) {

} catch (AddressException ex) {

} catch (UnknownHostException ex) {

}

return null;

}

static InternetAddress \_getLocalAddress(Session session)

throws SecurityException, AddressException, UnknownHostException

{

String user = null; String host = null; String address = null;

if (session == null) {

user = System.getProperty("user.name");

host = getLocalHostName();

} else {

address = session.getProperty("zhizhi.from");

if (address == null) {

user = session.getProperty("zhizhi.user");

if ((user == null) || (user.length() == 0))

user = session.getProperty("user.name");

if ((user == null) || (user.length() == 0))

user = System.getProperty("user.name");

host = session.getProperty("zhizhi.host");

if ((host == null) || (host.length() == 0)) {

host = getLocalHostName();

}

}

}

if ((address == null) && (user != null) && (user.length() != 0) && (host != null) && (host.length() != 0))

{

address = MimeUtility.quote(user.trim(), "()<>,;:\\\"[]@\t ") + "@" + host;

}

if (address == null) {

return null;

}

return new InternetAddress(address);

}

private static String getLocalHostName()

throws UnknownHostException

{

String host = null;

InetAddress me = InetAddress.getLocalHost();

if (me != null) {

host = me.getHostName();

if ((host != null) && (host.length() > 0) && (isInetAddressLiteral(host)))

host = '[' + host + ']';

}

return host;

}

private static boolean isInetAddressLiteral(String addr)

{

boolean sawHex = false; boolean sawColon = false;

for (int i = 0; i < addr.length(); i++) {

char c = addr.charAt(i);

if ((c < '0') || (c > '9'))

{

if (c != '.')

{

if (((c >= 'a') && (c <= 'z')) || ((c >= 'A') && (c <= 'Z')))

sawHex = true;

else if (c == ':')

sawColon = true;

else

return false;

}

}

}

return (!sawHex) || (sawColon);

}

public static InternetAddress[] parse(String addresslist)

throws AddressException

{

return parse(addresslist, true);

}

public static InternetAddress[] parse(String addresslist, boolean strict)

throws AddressException

{

return parse(addresslist, strict, false);

}

public static InternetAddress[] parseHeader(String addresslist, boolean strict)

throws AddressException

{

return parse(addresslist, strict, true);

}

private static InternetAddress[] parse(String s, boolean strict, boolean parseHdr)

throws AddressException

{

int start\_personal = -1; int end\_personal = -1;

int length = s.length();

boolean ignoreErrors = (parseHdr) && (!strict);

boolean in\_group = false;

boolean route\_addr = false;

boolean rfc822 = false;

List v = new ArrayList();

int end;

int start = end = -1; for (int index = 0; index < length; index++) {

char c = s.charAt(index);

}

if (nesting > 0) {

if (!ignoreErrors) {

throw new AddressException("Missing ')'", s, index);

}

index = pindex + 1;

}

else {

index--;

if (start\_personal == -1)

start\_personal = pindex + 1;

if (end\_personal == -1)

if (start == -1) {

route\_addr = false;

rfc822 = false;

start = end = -1;

continue;

}

if (!in\_group)

{

if (end == -1)

end = index;

String addr = s.substring(start, end).trim();

InternetAddress ma = new InternetAddress();

ma.setAddress(addr);

if (start\_personal >= 0) {

ma.encodedPersonal = unquote(s.substring(start\_personal, end\_personal).trim());

}

v.add(ma);

route\_addr = false;

rfc822 = false;

start = end = -1;

start\_personal = end\_personal = -1;

}

}

int rindex = index;

boolean inquote = false;

for (index++; index < length; index++) {

c = s.charAt(index);

if (inquote) {

if (!ignoreErrors) {

throw new AddressException("Missing '\"'", s, index);

}

for (index = rindex + 1; index < length; index++) {

c = s.charAt(index);

if (c == '\\')

index++;

else if (c == '>')

{

break;

}

}

}

if (index >= length) {

if (!ignoreErrors) {

throw new AddressException("Missing '>'", s, index);

}

index = rindex + 1;

if (start == -1)

start = rindex;

}

else

{

if (!in\_group) {

start\_personal = start;

if (start\_personal >= 0)

end\_personal = rindex;

start = rindex + 1;

}

route\_addr = true;

end = index;

if ((parseHdr) && (!strict) && (index + 1 < length) && (s.charAt(index + 1) == '@')) {

continue;

}

InternetAddress ma = new InternetAddress();

end = index + 1;

ma.setAddress(s.substring(start, end).trim());

v.add(ma);

route\_addr = false;

rfc822 = false;

start = end = -1;

start\_personal = end\_personal = -1;

continue;

}

if (!ignoreErrors) {

throw new AddressException("Illegal semicolon, not in group", s, index);

}

if ((parseHdr) && (!strict) && (pers != null) && (pers.indexOf('@') >= 0) && (addr.indexOf('@') < 0) && (addr.indexOf('!') < 0))

{

String tmp = addr;

addr = pers;

pers = tmp;

}

if ((rfc822) || (strict) || (parseHdr)) {

if (!ignoreErrors)

checkAddress(addr, route\_addr, false);

InternetAddress ma = new InternetAddress();

ma.setAddress(addr);

if (pers != null)

ma.encodedPersonal = pers;

v.add(ma);

}

else {

StringTokenizer st = new StringTokenizer(addr);

while (st.hasMoreTokens()) {

String a = st.nextToken();

checkAddress(a, false, false);

InternetAddress ma = new InternetAddress();

ma.setAddress(a);

v.add(ma);

}

}

route\_addr = false;

rfc822 = false;

start = end = -1;

start\_personal = end\_personal = -1;

if ((parseHdr) && (!strict))

{

if (index + 1 < length) {

String addressSpecials = ")>[]:@\\,.";

char nc = s.charAt(index + 1);

if (addressSpecials.indexOf(nc) >= 0) {

if (nc != '@')

{

continue;

}

for (int i = index + 2; i < length; i++) {

nc = s.charAt(i);

if (nc == ';')

break;

if (addressSpecials.indexOf(nc) >= 0)

break;

}

if (nc == ';')

{

continue;

}

}

}

if (start == -1) {

start = index;

}

}

if (start >= 0)

{

if (end == -1) {

end = length;

}

String addr = s.substring(start, end).trim();

String pers = null;

if ((rfc822) && (start\_personal >= 0)) {

pers = unquote(s.substring(start\_personal, end\_personal).trim());

if (pers.trim().length() == 0) {

pers = null;

}

}

if ((parseHdr) && (!strict) && (pers != null) && (pers.indexOf('@') >= 0) && (addr.indexOf('@') < 0) && (addr.indexOf('!') < 0))

{

String tmp = addr;

addr = pers;

pers = tmp;

}

if ((rfc822) || (strict) || (parseHdr)) {

if (!ignoreErrors)

checkAddress(addr, route\_addr, false);

InternetAddress ma = new InternetAddress();

ma.setAddress(addr);

if (pers != null)

ma.encodedPersonal = pers;

v.add(ma);

}

else {

StringTokenizer st = new StringTokenizer(addr);

while (st.hasMoreTokens()) {

String a = st.nextToken();

checkAddress(a, false, false);

InternetAddress ma = new InternetAddress();

ma.setAddress(a);

v.add(ma);

}

}

}

InternetAddress[] a = new InternetAddress[v.size()];

v.toArray(a);

return a;

}

public void validate()

throws AddressException

{

if (isGroup())

getGroup(true);

else

checkAddress(getAddress(), true, true);

}

private static void checkAddress(String addr, boolean routeAddr, boolean validate)

throws AddressException

{

int start = 0;

int len = addr.length();

if (len == 0) {

throw new AddressException("Empty address", addr);

}

if ((routeAddr) && (addr.charAt(0) == '@'))

{

int i;

for (start = 0; (i = indexOfAny(addr, ",:", start)) >= 0;

start = i + 1) {

if (addr.charAt(start) != '@')

throw new AddressException("Illegal route-addr", addr);

if (addr.charAt(i) == ':')

{

start = i + 1;

break;

}

}

}

char c = 65535;

char lastc = 65535;

boolean inquote = false;

for (int i = start; i < len; i++) {

lastc = c;

c = addr.charAt(i);

if ((c != '\\') && (lastc != '\\'))

{

if (c == '"') {

if (inquote)

{

if ((validate) && (i + 1 < len) && (addr.charAt(i + 1) != '@')) {

throw new AddressException("Quote not at end of local address", addr);

}

inquote = false;

} else {

if ((validate) && (i != 0)) {

throw new AddressException("Quote not at start of local address", addr);

}

inquote = true;

}

}

else if (!inquote)

{

if (c == '@') {

if (i != 0) break;

throw new AddressException("Missing local name", addr);

}

if ((c <= ' ') || (c >= '')) {

throw new AddressException("Local address contains control or whitespace", addr);

}

if ("()<>,;:\\\"[]@".indexOf(c) >= 0)

throw new AddressException("Local address contains illegal character", addr);

}

}

}

if (inquote) {

throw new AddressException("Unterminated quote", addr);

}

if (c != '@') {

if (validate)

throw new AddressException("Missing final '@domain'", addr);

return;

}

start = i + 1;

if (start >= len) {

throw new AddressException("Missing domain", addr);

}

if (addr.charAt(start) == '.')

throw new AddressException("Domain starts with dot", addr);

for (i = start; i < len; i++) {

c = addr.charAt(i);

if (c == '[')

return;

if ((c <= ' ') || (c >= '')) {

throw new AddressException("Domain contains control or whitespace", addr);

}

if ((!Character.isLetterOrDigit(c)) && (c != '-') && (c != '.')) {

throw new AddressException("Domain contains illegal character", addr);

}

if ((c == '.') && (lastc == '.')) {

throw new AddressException("Domain contains dot-dot", addr);

}

lastc = c;

}

if (lastc == '.')

throw new AddressException("Domain ends with dot", addr);

}

private boolean isSimple()

{

return (this.address == null) || (indexOfAny(this.address, "()<>,;:\\\"[]") < 0);

}

public boolean isGroup()

{

return (this.address != null) && (this.address.endsWith(";")) && (this.address.indexOf(':') > 0);

}

public InternetAddress[] getGroup(boolean strict)

throws AddressException

{

String addr = getAddress();

if (!addr.endsWith(";"))

return null;

int ix = addr.indexOf(':');

if (ix < 0) {

return null;

}

String list = addr.substring(ix + 1, addr.length() - 1);

return parseHeader(list, strict);

}

private static int indexOfAny(String s, String any)

{

return indexOfAny(s, any, 0);

}

private static int indexOfAny(String s, String any, int start) {

try {

int len = s.length();

for (int i = start; i < len; i++) {

if (any.indexOf(s.charAt(i)) >= 0)

return i;

}

return -1; } catch (StringIndexOutOfBoundsException e) {

}

return -1;

}

}

package Android.zhizhi;

import java.io.PrintStream;

import java.text.FieldPosition;

import java.text.NumberFormat;

import java.text.ParseException;

import java.text.ParsePosition;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import java.util.GregorianCalendar;

import java.util.Locale;

import java.util.TimeZone;

public class ZhiziDateFormat extends SimpleDateFormat

{

private static final long serialVersionUID = -8148227605210628779L;

static boolean debug = false;

private static final Calendar cal = new GregorianCalendar(TimeZone.getTimeZone("GMT"));

public ZhizhiDateFormat()

{

super("EEE, d MMM yyyy HH:mm:ss 'XXXXX' (z)", Locale.US);

}

public StringBuffer format(Date date, StringBuffer dateStrBuf, FieldPosition fieldPosition)

{

int start = dateStrBuf.length();

super.format(date, dateStrBuf, fieldPosition);

int pos = 0;

for (pos = start + 25; dateStrBuf.charAt(pos) != 'X'; pos++);

this.calendar.clear();

this.calendar.setTime(date);

int offset = this.calendar.get(15) + this.calendar.get(16);

if (offset < 0) {

dateStrBuf.setCharAt(pos++, '-');

offset = -offset;

} else {

dateStrBuf.setCharAt(pos++, '+');

}

int rawOffsetInMins = offset / 60 / 1000;

int offsetInHrs = rawOffsetInMins / 60;

int offsetInMins = rawOffsetInMins % 60;

dateStrBuf.setCharAt(pos++, Character.forDigit(offsetInHrs / 10, 10));

dateStrBuf.setCharAt(pos++, Character.forDigit(offsetInHrs % 10, 10));

dateStrBuf.setCharAt(pos++, Character.forDigit(offsetInMins / 10, 10));

dateStrBuf.setCharAt(pos++, Character.forDigit(offsetInMins % 10, 10));

return dateStrBuf;

}

public Date parse(String text, ParsePosition pos)

{

return parseDate(text.toCharArray(), pos, isLenient());

}

private static Date parseDate(char[] orig, ParsePosition pos, boolean lenient)

{

try

{

int day = -1;

int month = -1;

int year = -1;

int hours = 0;

int minutes = 0;

int seconds = 0;

int offset = 0;

ZhizhiDateParser p = new ZhizhiDateParser(orig, pos.getIndex());

p.skipUntilNumber();

day = p.parseNumber();

if (!p.skipIfChar('-')) {

p.skipWhiteSpace();

}

month = p.parseMonth();

if (!p.skipIfChar('-')) {

p.skipWhiteSpace();

}

year = p.parseNumber();

if (year < 50)

year += 2000;

else if (year < 100) {

year += 1900;

}

p.skipWhiteSpace();

hours = p.parseNumber();

p.skipChar(':');

minutes = p.parseNumber();

if (p.skipIfChar(':')) {

seconds = p.parseNumber();

}

try

{

p.skipWhiteSpace();

offset = p.parseTimeZone();

} catch (ParseException pe) {

if (debug) {

System.out.println("No timezone? : '" + new String(orig) + "'");

}

}

pos.setIndex(p.getIndex());

return ourUTC(year, month, day, hours, minutes, seconds, offset, lenient);

}

catch (Exception e)

{

if (debug) {

System.out.println("Bad date: '" + new String(orig) + "'");

e.printStackTrace();

}

pos.setIndex(1);

}return null;

}

private static synchronized Date ourUTC(int year, int mon, int mday, int hour, int min, int sec, int tzoffset, boolean lenient)

{

cal.clear();

cal.setLenient(lenient);

cal.set(1, year);

cal.set(2, mon);

cal.set(5, mday);

cal.set(11, hour);

cal.set(12, min);

cal.add(12, tzoffset);

cal.set(13, sec);

return cal.getTime();

}

public void setCalendar(Calendar newCalendar)

{

throw new RuntimeException("Method setCalendar() shouldn't be called");

}

public void setNumberFormat(NumberFormat newNumberFormat)

{

throw new RuntimeException("Method setNumberFormat() shouldn't be called");

}

}

package Android.zhizhi;

import java.text.ParseException;

class ZhizhiDateParser

{

int index = 0;

char[] orig = null;

public ZhizhiDateParser(char[] orig, int index) {

this.orig = orig;

this.index = index;

}

public void skipUntilNumber()

throws ParseException

{

try

{

this.index += 1;

}

}

catch (ArrayIndexOutOfBoundsException e) {

}

throw new ParseException("No Number Found", this.index);

}

public void skipWhiteSpace()

{

int len = this.orig.length;

while (this.index < len)

}

public int peekChar()

throws ParseException

{

if (this.index < this.orig.length) {

return this.orig[this.index];

}

throw new ParseException("No more characters", this.index);

}

public void skipChar(char c)

throws ParseException

{

if (this.index < this.orig.length) {

if (this.orig[this.index] == c)

this.index += 1;

else

throw new ParseException("Wrong char", this.index);

}

else

throw new ParseException("No more characters", this.index);

}

public boolean skipIfChar(char c)

throws ParseException

{

if (this.index < this.orig.length) {

if (this.orig[this.index] == c) {

this.index += 1;

return true;

}

return false;

}

throw new ParseException("No more characters", this.index);

}

public int parseNumber()

throws ParseException

{

int length = this.orig.length;

boolean gotNum = false;

int result = 0;

while (this.index < length) {

if (gotNum) {

return result;

}

throw new ParseException("No Number found", this.index);

}

this.index += 1;

}

if (gotNum) {

return result;

}

throw new ParseException("No Number found", this.index);

}

public int parseMonth()

throws ParseException

{

public int parseTimeZone()

throws ParseException

{

if (this.index >= this.orig.length) {

throw new ParseException("No more characters", this.index);

}

char test = this.orig[this.index];

if ((test == '+') || (test == '-')) {

return parseNumericTimeZone();

}

return parseAlphaTimeZone();

}

public int parseNumericTimeZone()

throws ParseException

{

boolean switchSign = false;

char first = this.orig[(this.index++)];

if (first == '+')

switchSign = true;

else if (first != '-') {

throw new ParseException("Bad Numeric TimeZone", this.index);

}

int oindex = this.index;

int tz = parseNumber();

if (tz >= 2400)

throw new ParseException("Numeric TimeZone out of range", oindex);

int offset = tz / 100 \* 60 + tz % 100;

if (switchSign) {

return -offset;

}

return offset;

}

public int parseAlphaTimeZone()

throws ParseException

{

int result = 0;

boolean foundCommon = false;

char curr;

} catch (ArrayIndexOutOfBoundsException e) {

throw new ParseException("Bad Alpha TimeZone", this.index);

}

if (foundCommon) {

curr = this.orig[(this.index++)];

if ((curr == 'S') || (curr == 's')) {

curr = this.orig[(this.index++)];

if ((curr != 'T') && (curr != 't'))

throw new ParseException("Bad Alpha TimeZone", this.index);

}

else if ((curr == 'D') || (curr == 'd')) {

curr = this.orig[(this.index++)];

if ((curr == 'T') || (curr != 't'))

{

result -= 60;

}

else throw new ParseException("Bad Alpha TimeZone", this.index);

}

}

return result;

}

int getIndex() {

return this.index;

}

}

package Android.zhizhi;

import com.sun.mail.util.ASCIIUtility;

import com.sun.mail.util.FolderClosedIOException;

import com.sun.mail.util.LineOutputStream;

import com.sun.mail.util.MessageRemovedIOException;

import com.sun.mail.util.MimeUtil;

import com.sun.mail.util.PropUtil;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.ByteArrayInputStream;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.UnsupportedEncodingException;

import java.util.Enumeration;

import java.util.Vector;

import Android.activation.DataHandler;

import Android.activation.DataSource;

import Android.activation.FileDataSource;

import Android.zhizhi.BodyPart;

import Android.zhizhi.FolderClosedException;

import Android.zhizhi.Message;

import Android.zhizhi.MessageRemovedException;

import Android.zhizhi.MessagingException;

import Android.zhizhi.Multipart;

public class MimeBodyPart extends BodyPart

implements MimePart

{

private static final boolean setDefaultTextCharset = PropUtil.getBooleanSystemProperty("zhizhi.mime.setdefaulttextcharset", true);

private static final boolean setContentTypeFileName = PropUtil.getBooleanSystemProperty("zhizhi.mime.setcontenttypefilename", true);

private static final boolean encodeFileName = PropUtil.getBooleanSystemProperty("zhizhi.mime.encodefilename", false);

private static final boolean decodeFileName = PropUtil.getBooleanSystemProperty("zhizhi.mime.decodefilename", false);

private static final boolean ignoreMultipartEncoding = PropUtil.getBooleanSystemProperty("zhizhi.mime.ignoremultipartencoding", true);

static final boolean cacheMultipart = PropUtil.getBooleanSystemProperty("zhizhi.mime.cachemultipart", true);

protected DataHandler dh;

protected byte[] content;

protected InputStream contentStream;

protected InternetHeaders headers;

private Object cachedContent;

public MimeBodyPart()

{

this.headers = new InternetHeaders();

}

public MimeBodyPart(InputStream is)

throws MessagingException

{

if ((!(is instanceof ByteArrayInputStream)) && (!(is instanceof BufferedInputStream)) && (!(is instanceof SharedInputStream)))

{

is = new BufferedInputStream(is);

}

this.headers = new InternetHeaders(is);

if ((is instanceof SharedInputStream)) {

SharedInputStream sis = (SharedInputStream)is;

this.contentStream = sis.newStream(sis.getPosition(), -1L);

} else {

try {

this.content = ASCIIUtility.getBytes(is);

} catch (IOException ioex) {

throw new MessagingException("Error reading input stream", ioex);

}

}

}

public MimeBodyPart(InternetHeaders headers, byte[] content)

throws MessagingException

{

this.headers = headers;

this.content = content;

}

public int getSize()

throws MessagingException

{

if (this.content != null)

return this.content.length;

if (this.contentStream != null)

try {

int size = this.contentStream.available();

if (size > 0)

return size;

}

catch (IOException ex)

{

}

return -1;

}

public int getLineCount()

throws MessagingException

{

return -1;

}

public String getContentType()

throws MessagingException

{

String s = getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(this, s);

if (s == null)

s = "text/plain";

return s;

}

public boolean isMimeType(String mimeType)

throws MessagingException

{

return isMimeType(this, mimeType);

}

public String getDisposition()

throws MessagingException

{

return getDisposition(this);

}

public void setDisposition(String disposition)

throws MessagingException

{

setDisposition(this, disposition);

}

public String getEncoding()

throws MessagingException

{

return getEncoding(this);

}

public String getContentID()

throws MessagingException

{

return getHeader("Content-Id", null);

}

public void setContentID(String cid)

throws MessagingException

{

if (cid == null)

removeHeader("Content-ID");

else

setHeader("Content-ID", cid);

}

public String getContentMD5()

throws MessagingException

{

return getHeader("Content-MD5", null);

}

public void setContentMD5(String md5)

throws MessagingException

{

setHeader("Content-MD5", md5);

}

public String[] getContentLanguage()

throws MessagingException

{

return getContentLanguage(this);

}

public void setContentLanguage(String[] languages)

throws MessagingException

{

setContentLanguage(this, languages);

}

public String getDescription()

throws MessagingException

{

return getDescription(this);

}

public void setDescription(String description)

throws MessagingException

{

setDescription(description, null);

}

public void setDescription(String description, String charset)

throws MessagingException

{

setDescription(this, description, charset);

}

public String getFileName()

throws MessagingException

{

return getFileName(this);

}

public void setFileName(String filename)

throws MessagingException

{

setFileName(this, filename);

}

public InputStream getInputStream()

throws IOException, MessagingException

{

return getDataHandler().getInputStream();

}

protected InputStream getContentStream()

throws MessagingException

{

if (this.contentStream != null)

return ((SharedInputStream)this.contentStream).newStream(0L, -1L);

if (this.content != null) {

return new ByteArrayInputStream(this.content);

}

throw new MessagingException("No MimeBodyPart content");

}

public InputStream getRawInputStream()

throws MessagingException

{

return getContentStream();

}

public DataHandler getDataHandler()

throws MessagingException

{

if (this.dh == null)

this.dh = new MimePartDataHandler(new MimePartDataSource(this));

return this.dh;

}

public Object getContent()

throws IOException, MessagingException

{

if (this.cachedContent != null)

return this.cachedContent;

Object c;

try {

c = getDataHandler().getContent();

} catch (FolderClosedIOException fex) {

throw new FolderClosedException(fex.getFolder(), fex.getMessage());

} catch (MessageRemovedIOException mex) {

throw new MessageRemovedException(mex.getMessage());

}

if ((cacheMultipart) && (((c instanceof Multipart)) || ((c instanceof Message))) && ((this.content != null) || (this.contentStream != null)))

{

this.cachedContent = c;

if ((c instanceof MimeMultipart))

((MimeMultipart)c).parse();

}

return c;

}

public void setDataHandler(DataHandler dh)

throws MessagingException

{

this.dh = dh;

this.cachedContent = null;

invalidateContentHeaders(this);

}

public void setContent(Object o, String type)

throws MessagingException

{

if ((o instanceof Multipart))

setContent((Multipart)o);

else

setDataHandler(new DataHandler(o, type));

}

public void setText(String text)

throws MessagingException

{

setText(text, null);

}

public void setText(String text, String charset)

throws MessagingException

{

setText(this, text, charset, "plain");

}

public void setText(String text, String charset, String subtype)

throws MessagingException

{

setText(this, text, charset, subtype);

}

public void setContent(Multipart mp)

throws MessagingException

{

setDataHandler(new DataHandler(mp, mp.getContentType()));

mp.setParent(this);

}

public void attachFile(File file)

throws IOException, MessagingException

{

FileDataSource fds = new FileDataSource(file);

setDataHandler(new DataHandler(fds));

setFileName(fds.getName());

}

public void removeHeader(String name)

throws MessagingException

{

this.headers.removeHeader(name);

}

public Enumeration getAllHeaders()

throws MessagingException

{

return this.headers.getAllHeaders();

}

public Enumeration getMatchingHeaders(String[] names)

throws MessagingException

{

return this.headers.getMatchingHeaders(names);

}

public Enumeration getNonMatchingHeaders(String[] names)

throws MessagingException

{

return this.headers.getNonMatchingHeaders(names);

}

public void addHeaderLine(String line)

throws MessagingException

{

this.headers.addHeaderLine(line);

}

public Enumeration getAllHeaderLines()

throws MessagingException

{

return this.headers.getAllHeaderLines();

}

public Enumeration getMatchingHeaderLines(String[] names)

throws MessagingException

{

return this.headers.getMatchingHeaderLines(names);

}

public Enumeration getNonMatchingHeaderLines(String[] names)

throws MessagingException

{

return this.headers.getNonMatchingHeaderLines(names);

}

protected void updateHeaders()

throws MessagingException

{

updateHeaders(this);

if (this.cachedContent != null) {

this.dh = new DataHandler(this.cachedContent, getContentType());

this.cachedContent = null;

this.content = null;

if (this.contentStream != null)

try {

this.contentStream.close();

} catch (IOException ioex) {

}

this.contentStream = null;

}

}

static boolean isMimeType(MimePart part, String mimeType)

throws MessagingException

{

try

{

ContentType ct = new ContentType(part.getContentType());

return ct.match(mimeType); } catch (ParseException ex) {

}

static void setText(MimePart part, String text, String charset, String subtype)

throws MessagingException

{

if (charset == null) {

if (MimeUtility.checkAscii(text) != 1)

charset = MimeUtility.getDefaultMIMECharset();

else {

charset = "us-ascii";

}

}

part.setContent(text, "text/" + subtype + "; charset=" + MimeUtility.quote(charset, "()<>@,;:\\\"\t []/?="));

}

static String getDisposition(MimePart part) throws MessagingException

{

String s = part.getHeader("Content-Disposition", null);

if (s == null) {

return null;

}

ContentDisposition cd = new ContentDisposition(s);

return cd.getDisposition();

}

static void setDisposition(MimePart part, String disposition) throws MessagingException

{

if (disposition == null) {

part.removeHeader("Content-Disposition");

} else {

String s = part.getHeader("Content-Disposition", null);

if (s != null)

{

ContentDisposition cd = new ContentDisposition(s);

cd.setDisposition(disposition);

disposition = cd.toString();

}

part.setHeader("Content-Disposition", disposition);

}

}

static String getDescription(MimePart part) throws MessagingException

{

String rawvalue = part.getHeader("Content-Description", null);

if (rawvalue == null)

return null;

try

{

return MimeUtility.decodeText(MimeUtility.unfold(rawvalue)); } catch (UnsupportedEncodingException ex) {

}

return rawvalue;

}

static void setDescription(MimePart part, String description, String charset)

throws MessagingException

{

if (description == null) {

part.removeHeader("Content-Description");

return;

}

try

{

part.setHeader("Content-Description", MimeUtility.fold(21, MimeUtility.encodeText(description, charset, null)));

}

catch (UnsupportedEncodingException uex) {

throw new MessagingException("Encoding error", uex);

}

}

static String getFileName(MimePart part) throws MessagingException {

String filename = null;

String s = part.getHeader("Content-Disposition", null);

if (s != null)

{

ContentDisposition cd = new ContentDisposition(s);

filename = cd.getParameter("filename");

}

if (filename == null)

{

s = part.getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(part, s);

if (s != null)

try {

ContentType ct = new ContentType(s);

filename = ct.getParameter("name");

} catch (ParseException pex) {

}

}

if ((decodeFileName) && (filename != null)) {

try {

filename = MimeUtility.decodeText(filename);

} catch (UnsupportedEncodingException ex) {

throw new MessagingException("Can't decode filename", ex);

}

}

return filename;

}

static void setFileName(MimePart part, String name) throws MessagingException

{

if ((encodeFileName) && (name != null)) {

try {

name = MimeUtility.encodeText(name);

} catch (UnsupportedEncodingException ex) {

throw new MessagingException("Can't encode filename", ex);

}

}

String s = part.getHeader("Content-Disposition", null);

ContentDisposition cd = new ContentDisposition(s == null ? "attachment" : s);

cd.setParameter("filename", name);

part.setHeader("Content-Disposition", cd.toString());

if (setContentTypeFileName) {

s = part.getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(part, s);

if (s != null)

try {

ContentType cType = new ContentType(s);

cType.setParameter("name", name);

part.setHeader("Content-Type", cType.toString());

}

catch (ParseException pex) {

}

}

}

static String[] getContentLanguage(MimePart part) throws MessagingException {

String s = part.getHeader("Content-Language", null);

if (s == null) {

return null;

}

HeaderTokenizer h = new HeaderTokenizer(s, "()<>@,;:\\\"\t []/?=");

Vector v = new Vector();

while (true)

{

HeaderTokenizer.Token tk = h.next();

int tkType = tk.getType();

if (tkType == -4)

break;

if (tkType == -1) {

v.addElement(tk.getValue());

}

}

if (v.size() == 0) {

return null;

}

String[] language = new String[v.size()];

v.copyInto(language);

return language;

}

public void attachFile(String file)

throws IOException, MessagingException

{

File f = new File(file);

attachFile(f);

}

public void saveFile(File file)

throws IOException, MessagingException

{

OutputStream out = null;

InputStream in = null;

try {

out = new BufferedOutputStream(new FileOutputStream(file));

in = getInputStream();

byte[] buf = new byte[8192];

int len;

while ((len = in.read(buf)) > 0)

out.write(buf, 0, len);

}

finally {

try {

if (in != null)

in.close();

} catch (IOException ex) {

}

try { if (out != null)

out.close();

}

catch (IOException ex)

{

}

}

}

public void saveFile(String file)

throws IOException, MessagingException

{

File f = new File(file);

saveFile(f);

}

public void writeTo(OutputStream os)

throws IOException, MessagingException

{

writeTo(this, os, null);

}

public String[] getHeader(String name)

throws MessagingException

{

return this.headers.getHeader(name);

}

public String getHeader(String name, String delimiter)

throws MessagingException

{

return this.headers.getHeader(name, delimiter);

}

public void setHeader(String name, String value)

throws MessagingException

{

this.headers.setHeader(name, value);

}

public void addHeader(String name, String value)

throws MessagingException

{

this.headers.addHeader(name, value);

}

public void removeHeader(String name)

throws MessagingException

{

this.headers.removeHeader(name);

}

public Enumeration getAllHeaders()

throws MessagingException

{

return this.headers.getAllHeaders();

}

public Enumeration getMatchingHeaders(String[] names)

throws MessagingException

{

return this.headers.getMatchingHeaders(names);

}

public Enumeration getNonMatchingHeaders(String[] names)

throws MessagingException

{

return this.headers.getNonMatchingHeaders(names);

}

public void addHeaderLine(String line)

throws MessagingException

{

this.headers.addHeaderLine(line);

}

public Enumeration getAllHeaderLines()

throws MessagingException

{

return this.headers.getAllHeaderLines();

}

public Enumeration getMatchingHeaderLines(String[] names)

throws MessagingException

{

return this.headers.getMatchingHeaderLines(names);

}

public Enumeration getNonMatchingHeaderLines(String[] names)

throws MessagingException

{

return this.headers.getNonMatchingHeaderLines(names);

}

protected void updateHeaders()

throws MessagingException

{

updateHeaders(this);

if (this.cachedContent != null) {

this.dh = new DataHandler(this.cachedContent, getContentType());

this.cachedContent = null;

this.content = null;

if (this.contentStream != null)

try {

this.contentStream.close();

} catch (IOException ioex) {

}

this.contentStream = null;

}

}

static boolean isMimeType(MimePart part, String mimeType)

throws MessagingException

{

try

{

ContentType ct = new ContentType(part.getContentType());

return ct.match(mimeType); } catch (ParseException ex) {

}

static void setText(MimePart part, String text, String charset, String subtype)

throws MessagingException

{

if (charset == null) {

if (MimeUtility.checkAscii(text) != 1)

charset = MimeUtility.getDefaultMIMECharset();

else {

charset = "us-ascii";

}

}

part.setContent(text, "text/" + subtype + "; charset=" + MimeUtility.quote(charset, "()<>@,;:\\\"\t []/?="));

}

static String getDisposition(MimePart part) throws MessagingException

{

String s = part.getHeader("Content-Disposition", null);

if (s == null) {

return null;

}

ContentDisposition cd = new ContentDisposition(s);

return cd.getDisposition();

}

static void setDisposition(MimePart part, String disposition) throws MessagingException

{

if (disposition == null) {

part.removeHeader("Content-Disposition");

} else {

String s = part.getHeader("Content-Disposition", null);

if (s != null)

{

ContentDisposition cd = new ContentDisposition(s);

cd.setDisposition(disposition);

disposition = cd.toString();

}

part.setHeader("Content-Disposition", disposition);

}

}

static String getDescription(MimePart part) throws MessagingException

{

String rawvalue = part.getHeader("Content-Description", null);

if (rawvalue == null)

return null;

try

{

return MimeUtility.decodeText(MimeUtility.unfold(rawvalue)); } catch (UnsupportedEncodingException ex) {

}

return rawvalue;

}

static void setDescription(MimePart part, String description, String charset)

throws MessagingException

{

if (description == null) {

part.removeHeader("Content-Description");

return;

}

try

{

part.setHeader("Content-Description", MimeUtility.fold(21, MimeUtility.encodeText(description, charset, null)));

}

catch (UnsupportedEncodingException uex) {

throw new MessagingException("Encoding error", uex);

}

}

static String getFileName(MimePart part) throws MessagingException {

String filename = null;

String s = part.getHeader("Content-Disposition", null);

if (s != null)

{

ContentDisposition cd = new ContentDisposition(s);

filename = cd.getParameter("filename");

}

if (filename == null)

{

s = part.getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(part, s);

if (s != null)

try {

ContentType ct = new ContentType(s);

filename = ct.getParameter("name");

} catch (ParseException pex) {

}

}

if ((decodeFileName) && (filename != null)) {

try {

filename = MimeUtility.decodeText(filename);

} catch (UnsupportedEncodingException ex) {

throw new MessagingException("Can't decode filename", ex);

}

}

return filename;

}

static void setFileName(MimePart part, String name) throws MessagingException

{

if ((encodeFileName) && (name != null)) {

try {

name = MimeUtility.encodeText(name);

} catch (UnsupportedEncodingException ex) {

throw new MessagingException("Can't encode filename", ex);

}

}

String s = part.getHeader("Content-Disposition", null);

ContentDisposition cd = new ContentDisposition(s == null ? "attachment" : s);

cd.setParameter("filename", name);

part.setHeader("Content-Disposition", cd.toString());

if (setContentTypeFileName) {

s = part.getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(part, s);

if (s != null)

try {

ContentType cType = new ContentType(s);

cType.setParameter("name", name);

part.setHeader("Content-Type", cType.toString());

}

catch (ParseException pex) {

}

}

}

static String[] getContentLanguage(MimePart part) throws MessagingException {

String s = part.getHeader("Content-Language", null);

if (s == null) {

return null;

}

HeaderTokenizer h = new HeaderTokenizer(s, "()<>@,;:\\\"\t []/?=");

Vector v = new Vector();

while (true)

{

HeaderTokenizer.Token tk = h.next();

int tkType = tk.getType();

if (tkType == -4)

break;

if (tkType == -1) {

v.addElement(tk.getValue());

}

}

if (v.size() == 0) {

return null;

}

String[] language = new String[v.size()];

v.copyInto(language);

return language;

}

static void setContentLanguage(MimePart part, String[] languages) throws MessagingException

{

StringBuffer sb = new StringBuffer(languages[0]);

int len = "Content-Language".length() + 2 + languages[0].length();

for (int i = 1; i < languages.length; i++) {

sb.append(',');

len++;

if (len > 76) {

sb.append("\r\n\t");

len = 8;

}

sb.append(languages[i]);

len += languages[i].length();

}

part.setHeader("Content-Language", sb.toString());

}

static String getEncoding(MimePart part) throws MessagingException {

String s = part.getHeader("Content-Transfer-Encoding", null);

if (s == null) {

return null;

}

s = s.trim();

HeaderTokenizer h = new HeaderTokenizer(s, "()<>@,;:\\\"\t []/?=");

HeaderTokenizer.Token tk;

int tkType;

do {

tk = h.next();

tkType = tk.getType();

if (tkType == -4) break;

}

while (tkType != -1);

return tk.getValue();

return s;

}

static void setEncoding(MimePart part, String encoding) throws MessagingException

{

part.setHeader("Content-Transfer-Encoding", encoding);

}

static String restrictEncoding(MimePart part, String encoding)

throws MessagingException

{

if ((!ignoreMultipartEncoding) || (encoding == null)) {

return encoding;

}

return encoding;

}

String type = part.getContentType();

if (type == null) {

return encoding;

}

try

{

ContentType cType = new ContentType(type);

if (cType.match("multipart/\*"))

return null;

if ((cType.match("message/\*")) && (!PropUtil.getBooleanSystemProperty("zhizhi.mime.allowencodedmessages", false)))

{

return null;

}

} catch (ParseException pex) {

}

return encoding;

}

static void updateHeaders(MimePart part) throws MessagingException {

DataHandler dh = part.getDataHandler();

if (dh == null)

return;

try

{

String type = dh.getContentType();

boolean composite = false;

boolean needCTHeader = part.getHeader("Content-Type") == null;

ContentType cType = new ContentType(type);

if (cType.match("multipart/\*"))

{

composite = true;

Object o;

Object o;

if ((part instanceof MimeBodyPart)) {

MimeBodyPart mbp = (MimeBodyPart)part;

o = mbp.cachedContent != null ? mbp.cachedContent : dh.getContent();

}

else

{

Object o;

if ((part instanceof MimeMessage)) {

MimeMessage msg = (MimeMessage)part;

o = msg.cachedContent != null ? msg.cachedContent : dh.getContent();

}

else {

o = dh.getContent(); }

}if ((o instanceof MimeMultipart))

((MimeMultipart)o).updateHeaders();

else {

throw new MessagingException("MIME part of type \"" + type + "\" contains object of type " + o.getClass().getName() + " instead of MimeMultipart");

}

}

else if (cType.match("message/rfc822")) {

composite = true;

}

if ((dh instanceof MimePartDataHandler)) {

return;

}

if (!composite) {

if (part.getHeader("Content-Transfer-Encoding") == null) {

setEncoding(part, MimeUtility.getEncoding(dh));

}

if ((needCTHeader) && (setDefaultTextCharset) && (cType.match("text/\*")) && (cType.getParameter("charset") == null))

{

String enc = part.getEncoding();

String charset;

String charset;

}

if (needCTHeader)

{

String s = part.getHeader("Content-Disposition", null);

if (s != null)

{

ContentDisposition cd = new ContentDisposition(s);

String filename = cd.getParameter("filename");

if (filename != null) {

cType.setParameter("name", filename);

type = cType.toString();

}

}

part.setHeader("Content-Type", type);

}

} catch (IOException ex) {

throw new MessagingException("IOException updating headers", ex);

}

}

static void invalidateContentHeaders(MimePart part) throws MessagingException

{

part.removeHeader("Content-Type");

part.removeHeader("Content-Transfer-Encoding");

}

static void writeTo(MimePart part, OutputStream os, String[] ignoreList)

throws IOException, MessagingException

{

LineOutputStream los = null;

if ((os instanceof LineOutputStream))

los = (LineOutputStream)os;

else {

los = new LineOutputStream(os);

}

Enumeration hdrLines = part.getNonMatchingHeaderLines(ignoreList);

while (hdrLines.hasMoreElements()) {

los.writeln((String)hdrLines.nextElement());

}

los.writeln();

InputStream is = null;

byte[] buf = null;

try

{

DataHandler dh = part.getDataHandler();

if ((dh instanceof MimePartDataHandler))

{

if ((part instanceof MimeBodyPart)) {

MimeBodyPart mbp = (MimeBodyPart)part;

is = mbp.getContentStream();

} else if ((part instanceof MimeMessage)) {

MimeMessage msg = (MimeMessage)part;

is = msg.getContentStream();

}

}

if (is != null)

{

buf = new byte[8192];

int len;

while ((len = is.read(buf)) > 0)

os.write(buf, 0, len);

} else {

os = MimeUtility.encode(os, restrictEncoding(part, part.getEncoding()));

part.getDataHandler().writeTo(os);

}

} finally {

if (is != null)

is.close();

buf = null;

}

os.flush();

}

static class MimePartDataHandler extends DataHandler

{

public MimePartDataHandler(DataSource ds)

{

super();

}

}

}

protected MimeMessage(Folder folder, int msgnum)

{

super(folder, msgnum);

this.flags = new Flags();

this.saved = true;

initStrict();

}

protected MimeMessage(Folder folder, InputStream is, int msgnum)

throws MessagingException

{

this(folder, msgnum);

initStrict();

parse(is);

}

protected MimeMessage(Folder folder, InternetHeaders headers, byte[] content, int msgnum)

throws MessagingException

{

this(folder, msgnum);

this.headers = headers;

this.content = content;

initStrict();

}

private void initStrict()

{

if (this.session != null)

this.strict = PropUtil.getBooleanSessionProperty(this.session, "zhizhi.mime.address.strict", true);

}

protected void parse(InputStream is)

throws MessagingException

{

if ((!(is instanceof ByteArrayInputStream)) && (!(is instanceof BufferedInputStream)) && (!(is instanceof SharedInputStream)))

{

is = new BufferedInputStream(is);

}

this.headers = createInternetHeaders(is);

if ((is instanceof SharedInputStream)) {

SharedInputStream sis = (SharedInputStream)is;

this.contentStream = sis.newStream(sis.getPosition(), -1L);

} else {

try {

this.content = ASCIIUtility.getBytes(is);

} catch (IOException ioex) {

throw new MessagingException("IOException", ioex);

}

}

this.modified = false;

}

public Address[] getFrom()

throws MessagingException

{

Address[] a = getAddressHeader("From");

if (a == null) {

a = getAddressHeader("Sender");

}

return a;

}

public void setFrom(Address address)

throws MessagingException

{

if (address == null)

removeHeader("From");

else

setHeader("From", address.toString());

}

public void setFrom()

throws MessagingException

{

InternetAddress me = null;

try {

me = InternetAddress.\_getLocalAddress(this.session);

}

catch (Exception ex)

{

throw new MessagingException("No From address", ex);

}

if (me != null)

setFrom(me);

else

throw new MessagingException("No From address");

}

public void addFrom(Address[] addresses)

throws MessagingException

{

addAddressHeader("From", addresses);

}

public Address getSender()

throws MessagingException

{

Address[] a = getAddressHeader("Sender");

if ((a == null) || (a.length == 0))

return null;

return a[0];

}

public void setSender(Address address)

throws MessagingException

{

if (address == null)

removeHeader("Sender");

else

setHeader("Sender", address.toString());

}

public Address[] getRecipients(Message.RecipientType type)

throws MessagingException

{

if (type == RecipientType.NEWSGROUPS) {

String s = getHeader("Newsgroups", ",");

return s == null ? null : NewsAddress.parse(s);

}

return getAddressHeader(getHeaderName(type));

}

public Address[] getAllRecipients()

throws MessagingException

{

Address[] all = super.getAllRecipients();

Address[] ng = getRecipients(RecipientType.NEWSGROUPS);

if (ng == null)

return all;

if (all == null) {

return ng;

}

Address[] addresses = new Address[all.length + ng.length];

System.arraycopy(all, 0, addresses, 0, all.length);

System.arraycopy(ng, 0, addresses, all.length, ng.length);

return addresses;

}

public void setRecipients(Message.RecipientType type, Address[] addresses)

throws MessagingException

{

if (type == RecipientType.NEWSGROUPS) {

if ((addresses == null) || (addresses.length == 0))

removeHeader("Newsgroups");

else

setHeader("Newsgroups", NewsAddress.toString(addresses));

}

else setAddressHeader(getHeaderName(type), addresses);

}

public void setRecipients(Message.RecipientType type, String addresses)

throws MessagingException

{

if (type == RecipientType.NEWSGROUPS) {

if ((addresses == null) || (addresses.length() == 0))

removeHeader("Newsgroups");

else

setHeader("Newsgroups", addresses);

}

else setAddressHeader(getHeaderName(type), addresses == null ? null : InternetAddress.parse(addresses));

}

public void addRecipients(Message.RecipientType type, Address[] addresses)

throws MessagingException

{

if (type == RecipientType.NEWSGROUPS) {

String s = NewsAddress.toString(addresses);

if (s != null)

addHeader("Newsgroups", s);

} else {

addAddressHeader(getHeaderName(type), addresses);

}

}

public void addRecipients(Message.RecipientType type, String addresses)

throws MessagingException

{

if (type == RecipientType.NEWSGROUPS) {

if ((addresses != null) && (addresses.length() != 0))

addHeader("Newsgroups", addresses);

}

else addAddressHeader(getHeaderName(type), InternetAddress.parse(addresses));

}

public Address[] getReplyTo()

throws MessagingException

{

Address[] a = getAddressHeader("Reply-To");

if ((a == null) || (a.length == 0))

a = getFrom();

return a;

}

public void setReplyTo(Address[] addresses)

throws MessagingException

{

setAddressHeader("Reply-To", addresses);

}

private Address[] getAddressHeader(String name)

throws MessagingException

{

String s = getHeader(name, ",");

return s == null ? null : InternetAddress.parseHeader(s, this.strict);

}

private void setAddressHeader(String name, Address[] addresses)

throws MessagingException

{

String s = InternetAddress.toString(addresses);

if (s == null)

removeHeader(name);

else

setHeader(name, s);

}

private void addAddressHeader(String name, Address[] addresses) throws MessagingException

{

if ((addresses == null) || (addresses.length == 0))

return;

Address[] a = getAddressHeader(name);

Address[] anew;

Address[] anew;

if ((a == null) || (a.length == 0)) {

anew = addresses;

} else {

anew = new Address[a.length + addresses.length];

System.arraycopy(a, 0, anew, 0, a.length);

System.arraycopy(addresses, 0, anew, a.length, addresses.length);

}

String s = InternetAddress.toString(anew);

if (s == null)

return;

setHeader(name, s);

}

public String getSubject()

throws MessagingException

{

String rawvalue = getHeader("Subject", null);

if (rawvalue == null)

return null;

try

{

return MimeUtility.decodeText(MimeUtility.unfold(rawvalue)); } catch (UnsupportedEncodingException ex) {

}

return rawvalue;

}

public void setSubject(String subject)

throws MessagingException

{

setSubject(subject, null);

}

public void setSubject(String subject, String charset)

throws MessagingException

{

if (subject == null)

removeHeader("Subject");

else

try {

setHeader("Subject", MimeUtility.fold(9, MimeUtility.encodeText(subject, charset, null)));

}

catch (UnsupportedEncodingException uex) {

throw new MessagingException("Encoding error", uex);

}

}

public Date getSentDate()

throws MessagingException

{

String s = getHeader("Date", null);

if (s != null) {

try {

synchronized (zhizhiDateFormat) {

return zhizhiDateFormat.parse(s);

}

} catch (ParseException pex) {

return null;

}

}

return null;

}

public void setSentDate(Date d)

throws MessagingException

{

if (d == null)

removeHeader("Date");

else

synchronized (zhizhiDateFormat) {

setHeader("Date", zhizhiDateFormat.format(d));

}

}

public Date getReceivedDate()

throws MessagingException

{

return null;

}

public int getSize()

throws MessagingException

{

if (this.content != null)

return this.content.length;

if (this.contentStream != null)

try {

int size = this.contentStream.available();

if (size > 0)

return size;

}

catch (IOException ex)

{

}

return -1;

}

public int getLineCount()

throws MessagingException

{

return -1;

}

public String getContentType()

throws MessagingException

{

String s = getHeader("Content-Type", null);

s = MimeUtil.cleanContentType(this, s);

if (s == null)

return "text/plain";

return s;

}

public boolean isMimeType(String mimeType)

throws MessagingException

{

return MimeBodyPart.isMimeType(this, mimeType);

}

public String getDisposition()

throws MessagingException

{

return MimeBodyPart.getDisposition(this);

}

public void setDisposition(String disposition)

throws MessagingException

{

MimeBodyPart.setDisposition(this, disposition);

}

public String getEncoding()

throws MessagingException

{

return MimeBodyPart.getEncoding(this);

}

public String getContentID()

throws MessagingException

{

return getHeader("Content-Id", null);

}

public void setContentID(String cid)

throws MessagingException

{

if (cid == null)

removeHeader("Content-ID");

else

setHeader("Content-ID", cid);

}

public String getContentMD5()

throws MessagingException

{

return getHeader("Content-MD5", null);

}

public void setContentMD5(String md5)

throws MessagingException

{

setHeader("Content-MD5", md5);

}

public String getDescription()

throws MessagingException

{

return MimeBodyPart.getDescription(this);

}

public void setDescription(String description)

throws MessagingException

{

setDescription(description, null);

}

public void setDescription(String description, String charset)

throws MessagingException

{

MimeBodyPart.setDescription(this, description, charset);

}

public String[] getContentLanguage()//获取语言类型

throws MessagingException

{

return MimeBodyPart.getContentLanguage(this);

}

public void setContentLanguage(String[] languages)

throws MessagingException

{

MimeBodyPart.setContentLanguage(this, languages);

}

public String getMessageID()

throws MessagingException

{

return getHeader("Message-ID", null);

}

public String getFileName()

throws MessagingException

{

return MimeBodyPart.getFileName(this);

}

public void setFileName(String filename)

throws MessagingException

{

MimeBodyPart.setFileName(this, filename);

}

private String getHeaderName(Message.RecipientType type)

throws MessagingException

{

String headerName;

if (type == Message.RecipientType.TO) { //判断接收到的消息类型

headerName = "To";

}

else

{

String headerName;//定义名头

if (type == Message.RecipientType.CC) {

headerName = "Cc";

}

else

{

String headerName;

if (type == Message.RecipientType.BCC) {

headerName = "Bcc";

}

else

{

String headerName;//定义名头

if (type == RecipientType.NEWSGROUPS) //判断类型是否为新闻

headerName = "Newsgroups";

else

throw new MessagingException("Invalid Recipient Type");

}

}

}

}

String headerName;

return headerName;

}

public InputStream getInputStream()

throws IOException, MessagingException

{

return getDataHandler().getInputStream();//返回输入流

}

protected InputStream getContentStream()

throws MessagingException//抛出异常

{

if (this.contentStream != null)

return ((SharedInputStream)this.contentStream).newStream(0L, -1L);

if (this.content != null) {

return new SharedByteArrayInputStream(this.content);

}

throw new MessagingException("No MimeMessage content");

}

public Object getContent()

throws IOException, MessagingException

{

if (this.cachedContent != null)//判断缓存内容是否为空

return this.cachedContent;//返回缓存内容

Object c;

public void setText(String text)

throws MessagingException

{

setText(text, null);

}

}

public final char getCodePointValue(int ch)

{

if ((ch >= 0) && (ch < 55296))

{

int offset = (this.m\_index\_[(ch >> 5)] << '\002') + (

ch & 0x1F);//右位移

return this.m\_data\_[offset];

}

int offset = getCodePointOffset(ch);

return offset >= 0 ? this.m\_data\_[offset] : this.m\_initialValue\_;//三目运算判断返回值

}

public final char getLeadValue(char ch)

{

return this.m\_data\_[getLeadOffset(ch)];

}

}