eFax Developer™

Java User Guide for Inbound Processing

Version 2.0

Revision Date: 11/05/2014

Contents

Notice

In all communications concerning this documentation, please refer to the revision date displayed on the front cover of this user's guide.

Copyright

Welcome

Welcome to *eFax Developer's Java User Guide for Inbound Processing*. This guide was designed to assist you in integrating *eFax Developer*™ into your current inbound processing.

Users who wish to "cut to the chase" can skip directly to the Programming section as a quick-start.

Contents at a glance

- **Product Introduction** will provide an introduction to *eFax Developer*™ *Inbound*
- **Product Integration** will provide information on using *eFax Developer*™ *SDKs*
- **Programming** will provide essential information for the client-side programmer
- **Code Samples** will provide code samples for accessing the *InboundRequest* object

Product Introduction

eFax Developer™ Inbound Overview

eFax Developer™ is an online interface used in conjunction with the *eFax Developer*™ client-side API. The client-side process is divided into inbound and outbound processing. This documentation will cover client-side inbound processing in its entirety.

Based on XML technology, *eFax Developer*™ gives clients the ability to have inbound faxes forwarded via an HTTP(S) POST request to a desired URL address. The POST will contain an XML-formatted argument representing data specific to the inbound fax received.

Clients will establish a procedure that will accept and process the request then return a response indicating that the POST request was successfully received.

Components

Software Developer's Kit (SDK)

The *eFax Developer*™ SDK contains everything the client programmer will need to create a seamless easy to use interface to *eFax Developer*™.

Components found within the SDK are separated into inbound and outbound functionality. This document will focus on inbound processing only.

com.efaxdeveloper.util-2.0.x

The *com.efaxdeveloper.util-2.0.x.jar* file is the client-side API library. This Java™ Archive (JAR) file contains the core classes required to interface with *eFax Developer*™.

commons-codec-1.6

The *commons-codec-1.6.jar* file contains the Apache Commons Codec, a required component to successfully interface with *eFax Developer*™.

InboundRequest (object)

InboundRequest is the main object class used by the client's inbound process.

Barcode (object)

The *Barcode* object is used by clients who are set to receive barcodes as part of their inbound fax. When a barcode is encountered, the *InboundRequest* object will return an ArrayList of one-to-many *Barcode* objects. The client can easily retrieve data for each barcode returned through their *Barcode* object instance.

Barcode Recognition with DataMatrix Barcode Symbology (optional)

Based on client account settings, fax images can be analyzed for barcodes. If this option is selected, the XML-formatted data will contain additional elements that hold the results of this analysis. Due to its reliability and robustness, the 2-dimentional DataMatrix symbology was chosen as our default.

Capable of encoding all 128 ASCII characters along with a number of different character sets, DataMatrix can accommodate up to 500 MB per square inch with a data capacity of 3,116 digits or up to 2,335 ASCII characters. DataMatrix barcodes have a high degree of redundancy and resists printing defects which make them the perfect match for fax image processing.

User Defined Field Handling (optional)

Based on a client's account settings, *eFax Developer*™ may generate a number of user defined field name/value pairings. If this feature is used, the *InboundRequest* object will return a Map object containing each name/value pair retrieved. Clients that desire further information regarding this special feature should contact eFax Developer's Customer Support Group.

Page Splitting (optional)

Based on client account settings, a multi-page fax image can be split out and returned as multiple single-page fax images. Clients who choose this option will receive XML-formatted data containing elements representing each page as a separate Base64 encoded container. Clients that desire further information regarding this special feature should contact eFax Developer's Customer Support Group.

XSD (XML Schema Definition)

To ensure data integrity, the *InboundRequest* object can be set to validate the inbound request against a client-side XSD schema. A sample schema is provided as part of the SDK.

eFax Developer™ Online Interface

Clients can change various inbound settings through *eFax Developer*™ online by clicking the "Settings" icon located on any page of the interface. Go to https://secure.efaxdeveloper.com to log into the *eFax Developer*™ online interface.

UserName and Password validation

eFax Developer™ will always send a username and password as part of the XML POST. As an additional security measure, clients may change their XSD appropriately to validate the username and password. Clients that perform username and password validation will be responsible for keeping their XSD in sync with any username and password changes made online via the *eFax Developer*™ online interface.

Product Integration

Integration Requirements

Applications that use the *com.efaxdeveloper.util-2.0.x* API must meet the following requirements.

- JDK 1.5 or greater is required
- Apache Commons Codec 1.6 or later (commons-codec-1.6.jar is included with our SDK package)

Integration Planning

To successfully integrate with *eFax Developer*™, it is important to have the appropriate tools as well as a project plan prior to attempting integration. The following is provided to assist you in the integration planning process.

- Contact *eFax Developer*™ Support to set up your account. Optional barcode recognition and/or user defined field handling should be discussed if desired.
- Log into the *eFax Developer*™ online interface to set your inbound settings as desired
- Unzip the SDK into a folder of your choosing
- Add the *com.efaxdeveloper.util-2.0.x.jar* and *commons_codec-1.6.jar* to your CLASSPATH
- Carefully review this documentation, sample code and Javadoc found within the SDK folder
- Develop the client-side process to receive the XML POST
- Test your application's integration with *eFax Developer*™
- Make your application live

Programming

Client-Side Programming Overview

The following is provided as a high-level overview of expected client-side processing.

- Create an application that accepts a URL-encoded HTTP(S) POST containing a single name/value pair where the parameter name equals "xml" and the parameter value contains the XML-formatted data. Pass the XML-formatted data to an instance of the *InboundRequest* object. Retrieve the fax data using the object's various getter methods. Generate an HTML response of "Post Successful" back to *eFax Developer*™.

Using eFax Developer's InboundRequest object

The following is provided as a more in-depth overview of the expected client-side processing.

- Retrieve the "xml" parameter from the HTTP(S) POST as you would normally do in your application environment.

- Establish a print output as you would normally do to send a response to a browser.

- Instantiate the *InboundRequest* object using the appropriate constructor method. During client testing, constructor methods that accept file input can be used to read XML from a specified file. Sample XML is included as part of this SDK for client reference and testing.

- Use the *InboundRequest* accessor/getter methods to retrieve the fax specific data. Please review the upcoming section regarding Accessor Method Mapping.

- Clients with user defined fields can retrieve them from the getUserFields() accessor method. The getUserFields() method will return a Map which can be spun through to retrieve the user defined field name/value pairs received.

- Clients set to interpret barcodes can retrieve them from the getBarcodes() accessor method. The getBarcodes() method will return an ArrayList of one-to-many *Barcode* objects. The ArrayList can be processed to retrieve and store information specific to each barcode image.

- Use the "getDocumentAsFile (String documentPath)" method when a physical copy of the fax document is desired. The "getDocumentAsFile (String documentPath)" method will write the physical file to the specified path. The file will be written using the fax name and file type returned in the XML POST. All "getDocumentAsXXXX()" calls are optionally used by clients who wish to store a copy of the physical fax on their server.

- Clients with page splitting turned on can use the "getPagesAsFiles (String pagePath)" method when a physical copy of each fax page is desired. The "getPagesAsFiles (String pagePath)" method will write each page of a physical fax to the specified path as its own file. Files will be written using the fax name returned in the XML POST and a suffix representing the page depicted. Calls to "getPagesAsFiles" or any "getPageAsXXXX()" methods are optionally used by clients who wish to store a copy of the physical fax on their server.

- Send an HTML response to the browser containing the string "Post Successful" anywhere in the HTML body.

Date Format Specification

Dates returned from the *InboundRequest* object can be formatted as desired by the client process. A number of easy to use preset values have been established. In addition to the following preset values, clients have the ability to specify their own settings based on java.text.SimpleDateFormat syntax. The following listing represents each preset date format code used by the *InboundRequest* object. Each example assumes an input date of "01/01/2005 12:30:00 PM."

DF_01 will return "01/01/2005 13:30:00" (this is the default)
DF_02 will return "01/01/2005 13:30"
DF_03 will return "01/01/2005 12:30:00 PM"
DF_04 will return "01/01/2005 12:30 PM"
DF_05 will return "01/01/2005"

DF_06 will return "2005/01/01 13:30:00"
DF_07 will return "2005/01/01 13:30"
DF_08 will return "2005/01/01 12:30:00 PM"
DF_09 will return "2005/01/01 12:30 PM"
DF_10 will return "2005/01/01"

DF_11 will return "January 1, 2005 13:30:00"
DF_12 will return "January 1, 2005 13:30"
DF_13 will return "January 1, 2005 12:30:00 PM"
DF_14 will return "January 1, 2005 12:30 PM"
DF_15 will return "January 1, 2005"

DF_16 will return "Jan 1, 2005 13:30:00"
DF_17 will return "Jan 1, 2005 13:30"
DF_18 will return "Jan 1, 2005 12:30:00 PM"
DF_19 will return "Jan 1, 2005 12:30 PM"
DF_20 will return "Jan 1, 2005"

Method Mapping

Accessor Method Mapping (*InboundRequest Object)*

Method	Returns	Length	Description/Values
getAccountID()	String	10	*eFax Developer*™ account identifier.
getANI()	String	25	The automatic number identification (caller id) contains the calling party's fax number.
getBarcodes()	ArrayList	N/A	The ArrayList will contain one-to-many *Barcode* objects.
getBarcodesByPage(int page)	ArrayList	N/A	The ArrayList will contain one-to-many Barcode objects from the specified page.
getBarcodesRead()	String	5	The number of barcodes successfully interpreted within the inbound fax document.
getCSID()	String	50	The station identifier, when supplied by the receiving fax machine upon successful transmission.
getDateReceivedAsDate()	Date	N/A	The original fax received date returned as a Date object. Pacific Time Zone
getDateReceivedAsString()	String	19	The original fax received date in "MM/dd/yyyy HH:mm:ss" (24 hour) default format. Pacific Time Zone
getDateReceivedAsString(String format)	String	Varies	The original fax received date in the specified date format. Pacific Time Zone
getDocumentAsBase64()	String	Varies	A Base64 encoded string representation of the files contents. Clients with page splitting will call the getPageAsBase64() method instead.

Method	Returns	Length	Description/Values
getDocumentAsBytes()	byte[]	Varies	A byte[] array representation of the files contents. Clients with page splitting will call the getPageAsBytes() method instead.
getDocumentAsFile(String documentPath)	void	N/A	Upon successful completion, a file will be created within the designated path using the file name and file type received in the XML POST. Clients with page splitting will call the getPagesAsFiles() method instead.
getFaxName()	String	50	The actual fax name given to this fax by *eFax Developer*™ as changed by the client.
getFileType()	String	3	"pdf" "tif" The document type as stored based on the "Inbound File Format" setting.
getMCFID()	String	8	*eFax Developer*™ fax ID.
getNumberDialed()	String	10	The fax number that was dialed.
getPageAsBase64(int page)	String	Varies	A Base64 encoded string representation of the page number specified.
getPageAsBytes(int page)	byte[]	Varies	A byte[] array representation of the page number specified.
getPageAsFile(String pagePath, int page)	void	N/A	Upon successful completion, a file will be created within the designated path for the page number specified.
getPageCount()	String	5	The number of pages received.
getPagesAsFiles(String pagePath)	void	N/A	Upon successful completion, a file will be created within the designated path for each page of the physical fax.
getPassword()	String	50	User Password

Method	Returns	Length	Description/Values
getRequestDateAsDate()	Date	N/A	The XML generation date returned as a Date object. Pacific Time Zone
getRequestDateAsString()	String	19	The XML generation date in "MM/dd/yyyy HH:mm:ss" (24 hour) default format. Pacific Time Zone
getRequestDateAsString(String format)	String	Varies	The XML generation date in the specified date format. Pacific Time Zone
getRequestType()	String	15	"New Inbound" "Manual Repost" The type of request that generated the XML POST.
getStatus()	String	5	Numeric field indicating the fax status. "0" indicates a successful transmission while all other values indicate an error code which can be cross-referenced with am *eFax Developer*™ supplied table.
getUserFields()	Map	N/A	The Map object contains user field name/value pairings.
getUserFieldsRead()	String	5	The number of user field name/value pairs returned.
getUserName()	String	50	User Name
hasBarcodes()	Boolean	N/A	The Boolean value received indicates whether or not barcodes where returned.
hasUserFields()	Boolean	N/A	The Boolean value received indicates whether or not user fields where returned.
isPageSplitting()	Boolean	N/A	The Boolean value received indicates whether or not page splits were returned.

Accessor Method Mapping (*Barcode Object)*

Method	Returns	Length	Description/Values
getKey()	String	Varies	The interpreted barcode value
getReadSequence()	String	3	A per-page sequence number given to each barcode as it is read on a given page.
getReadDirection()	String	50	"2-Dimentional" "Left/Right" "Top/Bottom" "Right/Left" "Bottom/Top" The read direction used to interpret the barcode stored within this container.
getSymbology()	String	30	The symbology or protocol used to generate the barcode stored within this container.
getPageNumber()	String	5	The physical page location of the barcode stored within this container.
getXStartPointA()	String	8	Coordinate grid x-axis for starting edge point A.
getYStartPointA()	String	8	Coordinate grid y-axis for starting edge point A.
getXStartPointB()	String	8	Coordinate grid x-axis for starting edge point B.
getYStartPointB()	String	8	Coordinate grid y-axis for starting edge point B.
getXEndPointA()	String	8	Coordinate grid x-axis for ending edge point A.
getYEndPointA()	String	8	Coordinate grid y-axis for ending edge point A.
getXEndPointB()	String	8	Coordinate grid x-axis for ending edge point B.
getYEndPointB()	String	8	Coordinate grid y-axis for ending edge point B.

Code Samples

Parsing Inbound Request (General Flow)

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  // Begin building the HTML response
  out.println("<html>");
  out.println("<head><title>InboundRequestServlet</title></head>");
  out.println("<body>");

  // Retrieve the inbound "xml" parameter value from the request
  String xml = request.getParameter("xml");

  try {
    // Establish a new InboundRequest instance
    InboundRequest req = new InboundRequest(xml);

    // Once the InboundRequest instance has been established, various
    // Metadata elements can be retrieved by your processing. A number of
    // options are available to you based on your account settings. At any
    // point, your process can reject the request by throwing an Exception.

    // When barcode fields have been passed (optional account setting)
    if (req.hasBarcodes()) {
      // Process barcodes (see Parsing for Barcode Metadata)
    }

    // When user defined fields have been passed (optional account setting)
    if (req.hasUserFields()) {
      // Process user defined fields (see Parsing for User Defined Fields)
    }

    // If the inbound request passes your validation, respond back to eFax
    // Developer™ with a "Post Successful"
    out.println("<p>Post Successful</p>");

  }
  catch (Exception e) {
    out.println("<p>Post Failed!</p>");
  }
  catch (Error err) {
    out.println("<p>Post Failed!</p>");
  }
  finally {
    out.println("</body></html>");
    out.flush();
    out.close();
  }
}
```

Parsing for Inbound Metadata

```
  // Retrieve the inbound "xml" parameter value from the request
  String xml = request.getParameter("xml");

  try {

    // Establish a new InboundRequest instannce
    InboundRequest req = new InboundRequest(xml);

    // Parse the InboundRequest
    System.out.println("UserName:      " + req.getUserName());
    System.out.println("Password:      " + req.getPassword());
    System.out.println("RequestDate:   " + req.getRequestDateAsString());
    System.out.println("RequestType:   " + req.getRequestType());
    System.out.println("AccountID:     " + req.getAccountID());
    System.out.println("NumberDialed:  " + req.getNumberDialed());
    System.out.println("DateReceived:  " + req.getDateReceivedAsString());
    System.out.println("FaxName:       " + req.getFaxName());
    System.out.println("FileType:      " + req.getFileType());
    System.out.println("PageCount:     " + req.getPageCount());
    System.out.println("CSID:          " + req.getCSID());
    System.out.println("ANI:           " + req.getANI());
    System.out.println("Status:        " + req.getStatus());
    System.out.println("MCFID:         " + req.getMCFID());

  }
  catch (Exception e) {}
  catch (Error err) {}
}
```

Parsing for Binary Image

```
  // Retrieve the inbound "xml" parameter value from the request
  String xml = request.getParameter("xml");

  try {

    // Establish a new InboundRequest instannce
    InboundRequest req = new InboundRequest(xml);

    // When the client account does not return page splits (most likely)
    if (!req.isPageSplitting()) {
      // Decode the Base64 encoded document to a specified location. The
      // file will be stored to your specified path using the fax name
      // retrieved from the XML POST FaxName parameter. Note that you are
      // providing a directory path to a location where you would like the
      // physical file to be written. The following method call is not
      // required. If the following method is not called, a hard copy of the
      // physical image will not be generated on your server.
      req.getDocumentAsFile("C:\\testXML\\");
    }
    // Otherwise, the client account is set to return page splits
    else {
      // Write each page to a specified file path
      req.getPagesAsFiles("C:\\testXML\\");
    }

  }
  catch (Exception e) {}
  catch (Error err) {}
}
```

Parsing for Barcode Metadata

```
// Retrieve the inbound "xml" parameter value from the request
String xml = request.getParameter("xml");

try {

  // Establish a new InboundRequest instannce
  InboundRequest req = new InboundRequest(xml);

  // When barcode fields have been passed (optional account setting)
  if (req.hasBarcodes()) {

    // Establish a null Barcode reference
    Barcode b = null;

    // Establish a new ArrayList object to hold all Barcode objects. Use
    // the getBarcodesByPage(int) method to return all Barcode objects
    // from a specified page.
    ArrayList<Barcode> bars = new ArrayList<Barcode>(req.getBarcodes());
    // Establish an iterator Object
    Iterator<Barcode> iterator = bars.iterator();
    // Process each Barcode object
    while (iterator.hasNext()) {
      // Retrieve a Barcode object
      b = new Barcode(iterator.next());
      // Retrieve the Barcode object state
      System.out.println("Key:           " + b.getKey());
      System.out.println("ReadSequence:  " + b.getReadSequence());
      System.out.println("ReadDirection: " + b.getReadDirection());
      System.out.println("Symbology:     " + b.getSymbology());
      System.out.println("PageNumber:    " + b.getPageNumber());
      System.out.println("XStartPointA:  " + b.getXStartPointA());
      System.out.println("XStartPointB:  " + b.getXStartPointB());
      System.out.println("YStartPointA:  " + b.getYStartPointA());
      System.out.println("YStartPointB:  " + b.getYStartPointB());
      System.out.println("XEndPointA:    " + b.getXEndPointA());
      System.out.println("XEndPointB:    " + b.getXEndPointB());
      System.out.println("YEndPointA:    " + b.getYEndPointA());
      System.out.println("YEndPointB:    " + b.getYEndPointB());
    }

  }

}
catch (Exception e) {}
catch (Error err) {}
}
```

Parsing for User Defined Fields

```
  // Retrieve the inbound "xml" parameter value from the request
  String xml = request.getParameter("xml");

  try {

    // Establish a new InboundRequest instannce
    InboundRequest req = new InboundRequest(xml);

    // When user defined fields have been passed (optional account setting)
    if (req.hasUserFields()) {

      // Establish a TreeMap object to hold the user name/value pairs
      Map<String, String> userFields =
                 new TreeMap<String, String>(req.getUserFields());
      // Store the TreeMap contents to an ArrayList object for sorting
      List<String> keys = new ArrayList<String>(userFields.keySet());
      // Establish an Iterator object for the List object
      Iterator<String> iterator = keys.iterator();
      // Establish a String object to hold the user field name
      String fieldName = "";
      // Establish a String object to hold the user field value
      String fieldValue = "";
      // Spin through the list of user name/value pairs
      while (iterator.hasNext()) {
        // Store the field name value
        fieldName = iterator.next();
        // Store the field value
        fieldValue = userFields.get(fieldName);
        // Print the user name/value pairs received in the XML
        System.out.println(fieldName + ": " + fieldValue);
      }

    }

  }
  catch (Exception e) {}
  catch (Error err) {}
}
```