edocs™

eaSDK: Auditing Data Streams

eaSDK is a member of the eaSuite™ product line

V3.4
Document ID: SDGN-05-3.4-02
Data Published: 1.30.2003

Table of Contents

Preface

About Customer Self-Service and eaSuite™

eaSuite

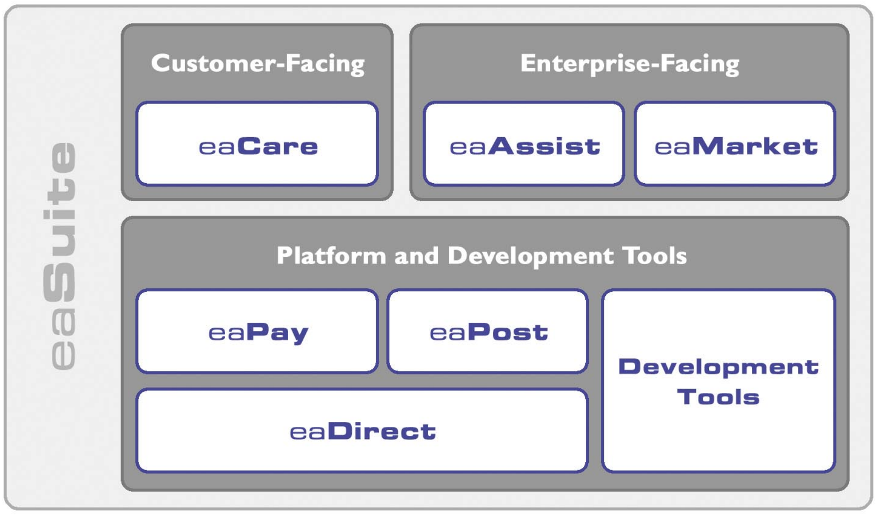
edocs has developed the industry's most comprehensive software and services for deploying Customer Self-Service solutions. eaSuite™ combines electronic presentment and payment (EPP), order management, knowledge management, personalization and application integration technologies to create an integrated, natural starting point for all customer service issues. eaSuite's unique architecture leverages and preserves existing infrastructure and data, and offers unparalleled scalability for the most demanding applications. With deployments across the healthcare, financial services, energy, retail, and communications industries, and the public sector, eaSuite powers some of the world's largest and most demanding customer self-service applications. eaSuite is a standards-based, feature rich, and highly scalable platform, that delivers the lowest total cost of ownership of any self-service solution available.

eaSuite is designed to support how organizations approach designing and deploying Customer Self-Service applications:

Customer-Facing Solutions present customers with the sophisticated functionality to meet customers' self-service needs. eaSuite offers a full set of capabilities to enable the range of business and consumer customer service activities, along with the flexibility to completely customize the solution to meet vertical industry and specific company requirements.

Enterprise-Facing Solutions empower employees within an organization and external partners to leverage the edocs platform to facilitate self-service and to support assisted service. Customer service representatives (CSRs), sales agents, account managers, marketing managers, broker-dealers and channel partners all play a role in delivering customer service, creating content, accessing information and performing activities for the benefit of customers.

Platform and Development Tools are designed to meet the rigorous infrastructure demands of the most technologically advanced organizations. These components of the eaSuite power edocs solutions with the functionality and development tools necessary to make account data available, and to create the customer- and enterprise-facing applications that enable customer self-service.



eaAssist

eaAssist™ reduces interaction costs and increases customer satisfaction by enabling enterprise agents – customer service representatives (CSRs), sales agents, broker-dealers and others – to efficiently access critical account data and service-related information to effectively service customers. Through its browser interface designed especially for the enterprise agent, eaAssist enables agents to take advantage of customer-facing online capabilities to provide better service by more efficiently resolving customer account inquiries at the point of customer contact.

eaMarket

eaMarket™ is the personalization, campaign and content management solution that enables organizations to increase revenue and improve customer satisfaction by weaving personalized marketing and customer service messages throughout the Customer Self-Service experience. The transactional account data that provides the foundation for a Customer Self-Service solution – such as transaction activity, service or usage charges, current task and prior service history – bring valuable insight into customers and can help optimize personalized marketing and customer service campaigns. eaMarket leverages that data to present relevant marketing and customer service messages to customers.

eaDirect

eaDirect™ is the core infrastructure of enterprise Customer Self-Service solutions for organizations large and small with special emphasis on meeting the needs of organizations with large numbers of customers, high data volumes and extensive integration with systems and business processes across the enterprise. Organizations use eaDirect with its data access layer, composition engine, and security, enrollment and logging framework to power complex Customer Self-Service applications.

eaPay

eaPay™ is the electronic payment solution that decreases payment processing costs, accelerates receivables and improves operational efficiency. eaPay is a complete payment scheduling and warehousing system with real-time and batch connections to payment gateways for Automated Clearing House (ACH) and credit card payments, and payments via various payment processing service providers.

eaPost

eaPost® is the account content distribution system that handles all the complexities of enrollment, authentication and secure distribution of summary account information to any endpoint, while also bringing customers back the organization's Website to manage and control their self-service experience.

Development Tools

eaSuite Development Tools™ are visual development applications that provide intuitive graphical user interface (GUI) environments for designing and developing Customer Self-Service solutions. The Development Tools encompass data management, workflow authoring, rules management and accounts receivable integration, as well as a full Software Developers Kit for custom application development.

About This Guide

The edocs Software Developers Kit, eaSDK, allows developers to write custom code against edocs applications. This SDK is intended for edocs system integrator partners, senior developers with an edocs client company, and edocs Professional Services representatives.

The edocs Software Developers Kit (eaSDK) allows developers to write custom code against edocs applications. This SDK is intended for edocs system integrator partners, senior developers with an edocs client company, and edocs Professional Services representatives.

eaSDK Auditing Data Streams is intended for web application developers implementing the Verify API, `com.edocs.app.verify,` to audit data volumes before accepting them for presentment..

The eaSDK assumes you have:

- Completed a Statement Mastering Plan.
- Installed and configured eaDirect and the sample application eaSample.
- Installed the eaDirect tools, DefTool and Composer, on a Windows NT or 2000 system.

It assumes in-depth understanding of and practical experience with:

- eaDirect system architecture, installation, deployment, application design, and administration
- Java 2 Enterprise Edition (J2EE), including Enterprise JavaBeans (EJBs), servlets, and JSPs
- Packaging and deploying J2EE applications for WebLogic 6.1 or WebSphere 4.0
- Directory services including the Java Naming Directory Interface (JNDI) and the Lightweight Directory Access Protocol (LDAP)
- HTML and XML, web server administration, and web browsers

Related Documentation

Javadoc for the SDK, Online Help for DefTool and Composer, and a PDF version of this guide are also available.

Online	How to Access
Javadoc	SDK Javadoc is available on the eaDirect SDK CD-ROM.

Help	Select Help > Help Topics in DefTool or Composer.
A PDF of this guide	A PDF of this guide is available on the eaDirect product CD-ROM.

This guide is part of the eaDirect documentation set. For more information about implementing your eaDirect application, see one of the following guides:

Print Document	**Description**
eaDirect Installation and Configuration Guide	How to install eaDirect and configure it in a distributed environment.
eaDirect User's Guide	How to design and create an eaDirect application for presenting statements online, and how to use the eaDirect DefTool and Composer design and development tools.
eaDirect Production Guide	How to set up and run a live eaDirect application in a J2EE environment.

If You Need Help

Technical support is available to customers who have valid maintenance and support contracts with edocs. Technical support engineers can help you install, configure, and maintain your edocs application.

To reach the U.S. Service Center, located in Natick, MA (Monday through Friday 8:00am to 8:00pm EST):

- Telephone: 508.652.8400
- Toll Free: 877.336.3362
- E-support: support.edocs.com (This requires a one-time online registration)
- E-mail: support@edocs.com

When you report a problem, please be prepared to provide us the following information:

- What is your name and role in your organization?
- What is your company's name?
- What is your phone number and best times to call you?
- What is your e-mail address?
- In which edocs product did a problem occur?
- What is your Operating System version?
- What were you doing when the problem occurred?
- How did the system respond to the error?
- If the system generated a screen message, please send us that screen message.
- If the system wrote information to a log file, please send us that log file.

If the system crashed or hung, please tell us.

Introduction to Auditing Data Streams 1

This module describes the Verify API, made available to help administrators verify whether a data stream has been correctly processed in eaDirect. Code examples are provided to illustrate the use of the various methods. (Note: this document uses the terms data stream, data stream, and volume interchangeably.)

About Auditing Data for Presentment

Before online statements are released for public access, it may be necessary to validate the input data stream. Various verification criteria may be defined depending on the information available about the data stream.

The methods in the Verify API only provide the means to manipulate an input stream as a whole, including the ability to mark the volume as accepted or rejected for online presentment.

Alternatively, the Content Access API operates at the statement level. If Verify is enabled, the Content Access API can only access statements in a volume that has been **accepted**.

Typical uses include:

- Add an audit level so administrators can ensure the print files are error free before making statements available to end-users.
- Confirm that the data delivered to eaDirect is compliant with the data definition specified.
- Define various verification criteria depending on information available about the data stream, such as number of accounts present in the data stream or account numbers known to be present in the data stream.

- Compare the number of extracted accounts to the number supposed to be present.
- Check to see if the account numbers known to be present in the data stream have actually been extracted (dummy accounts can be inserted in a data stream to facilitate such tests).
- Examine a random set of accounts more closely to determine whether line item values total up correctly.
- Test whether totals tally across all accounts to match some data stream total.

Verify API methods allow you to:

- Query which applications (DDNs) are deployed on an eaDirect server
- Obtain a list of accounts extracted from a single data stream
- Examine account summary information (via the Content Access API)
- Examine detailed account information (via the Content Access API)
- Accept or reject a processed data stream for online presentment

Application Programming Interfaces (API) for Auditing Data Streams

2

Package com.edocs.app.verify Description

About the Verify API

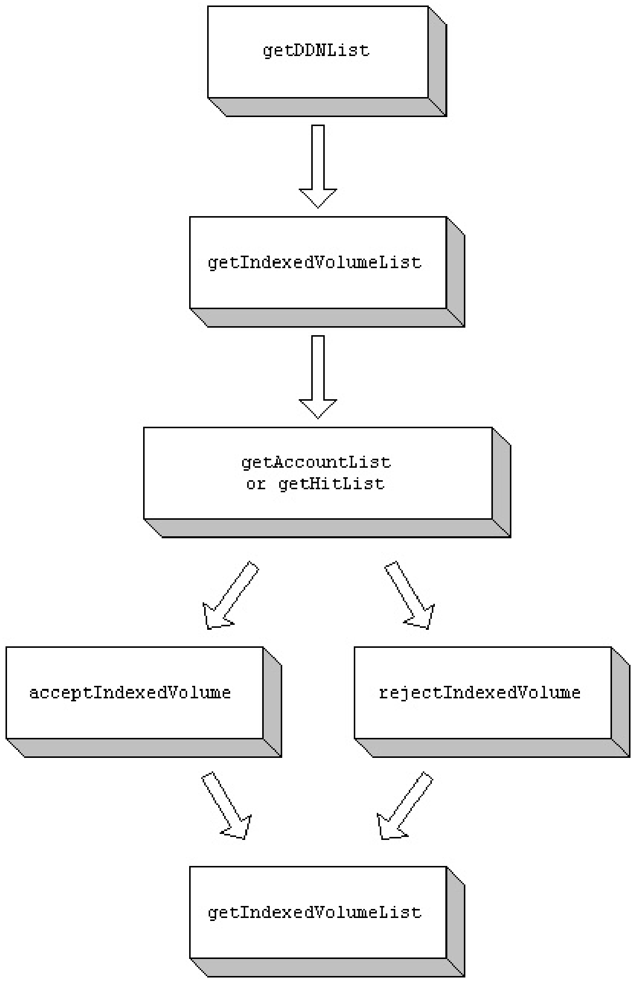
Provides the `Verify` class and methods for auditing indexed volumes of data before releasing them for presentment. Verify method `getIndexedVolumeList` retrieves a list of indexed volumes available for audit, while `getAccountList` retrieves all the account numbers in a volume. `GetDDNList` retrieves all DDNs. Two signatures of `GetHitList` retrieve all Description items either for a given volume, or for a given account. `acceptIndexedVolume` or `rejectIndexedVolume` respectively accept or reject a volume for presentment to customers. Finally, `updateDescriptionInfo` supports updates to the optional information field (Y_#) on a statement page.

Process Flow for Verify Methods

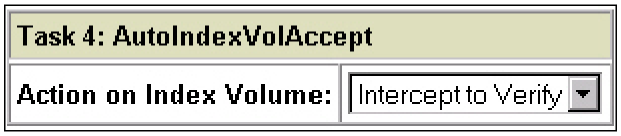
The following table summarizes the Verify methods; the Javadoc file provided with this module contains the reference information to provide detailed programming usage.

Method Name	Description
`acceptIndexedVolume`	Mark a processed data stream as valid.
`getAccountList`	Returns a list of account numbers associated with a processed print (data) file.
`getDDNList`	Returns a list of the DDNs defined.
`getHitList`	Returns account summary information.
`getIndexedVolumeList`	Returns a list of the processed data streams, identified by their indexed volume number (IVN).
`rejectIndexedVolume`	Marks as rejected an indexed volume
`updateSummaryInfo`	Enables updating of optional data fields

While the above list is in alphabetical order, Verify methods are typically called more or less in a fixed order, because the results of one call are often times necessary as input for another. The following diagram shows a possible process flow from one set of calls to the next. Methods listed in a single box or on the same level may be called in any order.



By default, when one runs an indexer job on a data stream in eaDirect, one has the option of requesting automatic verification (by selecting **`AutoIndexVolAccept`**) or intercepting to apply customized or manual verification. Interception requires that one select the **`Intercept to Verify`** option in the indexer job specification as shown below.



Task 4: AutoIndexVolAccept

Action on Index Volume: Intercept to Verify

This task is initially set to `Auto Accept`, which automatically verifies each volume processed by the Indexer job. If you set this value to `Intercept to Verify`, it will not return summary items for that print file until it is marked as accepted. Using the Verify API, you can define the parameters to accept or reject a volume based on the analysis results. If they are marked as accepted, `getSummary` will return the summary items in that print file and end users may have access to them.

Auditing Data Streams with the Verify API

3

The following sections discuss the individual methods of the Verify API. It begins with `getDDNList`, which allows you to determine which applications have been defined on an eaDirect server.

These methods require the import of the `com.edocs.app.verify.*` into the JSP to access them (see the example below).

Retrieve a List of All Applications

About getDDNList

To obtain the list of applications (DDNs) defined on an eaDirect server, the Verify API provides the `getDDNList` method that takes no arguments. Often one may already know the application one is interested in.

`getDDNList` is provided to facilitate the building of tools or user interfaces that enable dynamic selection of the application name. This method returns a string array (`String []`).

Example

The following example returns a list of DDNs deployed on the server you execute it on. You could use the return to populate a drop down list box or an HTML table to enable the user to pick which application he/she wishes to verify print files from.

```
<HTML>
```

```
<HEAD>
<TITLE>Applications List Example</TITLE>
</HEAD>
<%@
        page import ="
java.util.Properties,
java.text.*,
com.edocs.app.verify.*,
com.edocs.app.util.*" %>
<%
try
{
    String[] DDNList = Verify.getDDNList();
    out.print("<BR>" + "DDN List on this Server: " +
"<BR>")
    for (int i = 0; i < DDNList.length ; i++)
    out.print(DDNList[i] + "<BR>")
}
catch (Exception e)
{
}
%>
</TABLE>
</HTML>
```

Retrieve a List of Indexed Volumes

About getIndexedVolumeList

Each time a data stream is processed, it returns an **indexed volume** identified by a unique number, the index volume number (IVN). The method `getIndexedVolumeList` returns information about all the data streams processed. The information returned contains particulars such as date of processing and number of accounts extracted.

Parameters

`getIndexedVolumeList` accepts the following input parameters:

Name	Description
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is `short`.
`ddn`	The application name (DDN). Data type is `string`.

Results

`getIndexedVolumeList` returns a table. Row zero contains the column headings. Row one and onwards contain information about each of the indexed volumes. The number of rows retrieved is controlled using the “count” argument. Information about the last “count” number of indexed volumes processed is returned. If count is specified to be zero, information about all indexed volumes is returned. The column values in row one and beyond capture the value specified by the column heading (column value of the row zero element).

Caution Note the zero-based counting for rows and columns. Row [0] contains the column headings; data starts at row [1].

Row Name	Description

Row[0]	Column headings
Row[1], Row [2]... Row[count]	Information about an indexed volume.

Column #	**Column Name**	**Description**
0	Z_IVN	Index volume number
1	Z_FILE_NAME	Original file name
2	Z_FILE_PATH	Current file path
3	Z_DATE_CREATED	Date file was used as input. (long as string)
4	Z_DATE_ACCEPTED	Date file was accepted. Empty if file not accepted.
5	Z_DATE_REJECTED	Date file was rejected. Empty if file not rejected.
6	Z_DATE_EXPIRED	Date the file expired. Empty if it has not expired.
7	Z_DOC_COUNT	Number of documents in file

Date values returned as a string are really a single long value indicating the number of milliseconds since the *epoch* (January 1, 1970, 00:00:00 GMT).

When the indexer job runs with the selection `Intercept to Verify`, the *Z_DATE_ACCEPTED* and *Z_DATE_REJECTED* fields are left empty; it is neither accepted nor rejected. By contrast, the **`AutoIndexVolAccept`** task is used to automatically mark the IVN as accepted. Using the other Verify API methods, you can analyze the indexed volume before marking it accepted or rejected.

Example

The code sample below demonstrates one use of **`getIndexedVolumeList`**:

```
<%
     String ddn = "Training"; // any deployed application
     short count = 5;
     String[][] ivnList = getIndexVolumeList(ddn, count);
     String[] columnHeadings = invList[0];
     int numCols = columnHeadings.length;
     int numRows = ivnList.length;
<table>
<tr>
<%  // output column headings in BOLD
      for (int  i = 0; i< numCols; i++) {  %>
          <td> <b> <%= columnHeadings[i] %> </b> </td>
<%    } %>
</tr>
<%    for (int  i = 1; i< numRows;  i++) {
%>
<tr>
 <%      for (int  j = 0; j< numCols; j++) {    %>
             <td>  <%= ivnList[i][j] %>  </td>
 <%      }
     }  %>
</tr>
<%   }
</table>
```

The other Verify API methods deal for the most part with a single IVN. They allow a more detailed examination of the processing results and finally enable one to accept or reject such processing.

Example

The following example retrieves all files processed for the DDN **`training`**.

```
<HTML>
<HEAD>
```

```
<TITLE>IVN List Example</TITLE>
</HEAD>
<%@ page import ="
java.util.Properties,
java.text.*,
com.edocs.app.verify.*,
com.edocs.app.util.*" %>
<% short count = 0
String DDN = "training"
/*Declare multidimensional array to hold
return of getIndexVolumeList*/
String[][] ivlArr
//Declare a string called ivn and initialize to null
String ivn = null
/*Make a call to getIndexVolumeList and set
return equal to ivlArr*/
ivlArr = Verify.getIndexedVolumeList(DDN, count)
int rows = ivlArr.length
out.print("<B>" + "ivns for " + DDN + "
application"+ "</B>" + "<br>")
for (int i = 1; i < rows; i++)
{
ivn = ivlArr[i][0]
out.print( "ivn= " + ivn + "<br>")
}
%>
</HTML>
```

Retrieve a List of Account Numbers

About getAccountList

You have selected an indexed volume identified by its IVN (indexed volume number). What's next? You might be interested in finding out the account (or subaccount) numbers for which information was extracted. Three signatures of **`getAccountList`** return a list of account numbers:

```
getAccountList (string ivn, long offset, short count)

getAccountList (string ivn, name context, long offset,
short count)

getAccountList(string ivn, long offset, short count, String
pattern)
```

The length of the list returned can be tuned by selecting values for the input arguments **`offset, count`** and **`pattern`**.

When retrieving composite or compound account numbers, for example when working with subaccounts, use the **`javax.naming`** interface to specify the **`context`** parameter as type **`Name`**. For more information about the **Name** interface, see the Javadoc *eaDirect 3.4 SDK API Specification.*

Parameters

Name	Description
`ivn`	The index volume number (IVN). It is unique to a processed data stream. Data type is **`string`**.
`context`	The account number for which to retrieve subaccount data. Data type is **`name`**. See the Javadoc *eaDirect 3.4 SDK API Specification.*
`offset`	Determines the starting point to process account numbers in the data stream. Data type is **`long`**.
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is **`short`**.

`pattern`	A string used to filter the account numbers with a SQL command in a LIKE clause. Data type is **`string`**.

Using the offset parameter

Consider the data stream as a long list of statement records. **`getAccountList`** allows you to specify beyond which point in that data stream to look and retrieve account numbers. The offset acts as a bookmark, letting you home in on where you last left off. You may seek to examine the processing results chunk by chunk, sliding your view window perhaps by some fixed amount each time. This start position for viewing is the "offset." Often all account information may be accurately retrieved up to some point either because the data stream from that point on got corrupted or because the parsing rules failed to handle some situation at that point. By remembering the offset where the problem lay, the next time you could home in to see if the problem was resolved by selecting the same offset.

Example

```
String pattern = "2300%";
String ivn = request.getParameter("ivn");
// Z_IVN from a previous call to getIndexedVolumeList
short count = 10;
long offset = 0;
String[] accountNumbers  = null;
    accountNumbers = Verify.getAccountList(ivn,
                                           offset,
                                           count,
                                           pattern);
   // do some further processing of these accounts
```

The account number in conjunction with the IVN number can be used to retrieve statement summary and detail information. By examining them closely, one may determine whether the account information extraction from the data stream is accurate.

Example

The following example returns all accounts for the IVN with a Z_IVN of “2” which maps back to the training application as depicted in the prior JSP example.

```
<HTML>

<HEAD>

<TITLE>Account List Example</TITLE>

</HEAD>

<%@ page import =" java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>

<%

long l = 0; short s = 0

/*Start at the beginning of the file and return all accounts for
ivn passed in using getAccountList*/

String ivn = "2";

String[] acctList =

Verify.getAccountList(ivn, l, s);

out.print( "<BR>" + "<B>" + "Account List" + "</B>" + "<BR>")

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

        out.print( acctList[i] + "<BR>")

%>

</HTML>
```

Retrieve Account Summary Information

About getHitList

Two signatures of the **`getHitList`** method retrieve account summary information. It is similar to the Content Access API method **`getSummary`**, but it also returns summary information for statements that have not been accepted. One signature retrieves summary information for all accounts within a single indexed volume. The other signature operates across all indexed volumes for a single account number. The sections that follow discuss each in detail.

getHitList Signature For All Accounts In A Single IVN

Typically, use **`getHitList(String ivn, long offset, short count)`** to present an administrator with account summaries for accounts extracted from the processing of a single data stream. The purpose is to facilitate a closer inspection of one or more accounts to ensure accuracy.

Parameters

This signature of **`getHitList`** accepts the following input parameters:

Name	**Description**
`offset`	Determines the starting point to process account numbers in the data stream. Data type is **`long`**.
`count`	Used to indicate the maximum number of rows to be returned. Zero returns all. Data type is **`short`**.

The offset argument determines the point beyond which account numbers in the data stream are returned. Typically, one marches down the list of account numbers, retrieving a set of them, examines them carefully, and then moves on to the next set. The count argument determines the set size. The number of account summaries retrieved is the smaller of the two values: number of account numbers available beyond “offset,” and the set size.

Results

The signature `getHitList(String ivn, long offset, short count)` returns a table. The first row of the table contains the column headings and the remainder of the rows, one per account summary retrieved, contains column values. The tables below summarize the result set structure and column contents.

Row Name	Description
Row[0]	Column headings
Row[1]...Row[count]	Summary information for accounts
n	Number of columns
k	Number of optional fields as defined in indexer job.

Column #	Heading	Description
0	Z_PRIMARY_KEY	Account number
1	Z_DOC_ID	Document identifier
2	Z_DOC_DATE	Document date
3	Z_IVN	Indexed volume number
4 .. (n-k –1)	Indexed field names	Value of indexed fields
(n-k)...(n-1)	Optional field names (Y_1 to Y_k)	Values of optional fields

This method is also useful for customer service representative applications in navigating to an account in question.

getHitList Signature For One Account Across All IVNs

The signature **`getHitList (String account, String ddn, short count, long from, long to)`** provides summary information across all processed data streams for a given application (DDN) for a given account number. The “from” and “to” values indicate the range of processing dates from which to retrieve IVNs.

Typically, this method is used to test changes in parsing rules for the data stream. In particular, one tests that an account summary originally extracted correctly remains so and that problematic accounts cease to be so.

Tip

When retrieving composite or compound account numbers, for example when working with subaccounts, you can use the **`javax.naming`** interface to specify the account parameter as type **`Name`** instead of type **`String`**. For more information about the **Name** interface, see the Javadoc *eaDirect 3.4 SDK API Specification.*

Parameters

This signature of **`getHitList`** accepts the following input parameters:

Name	Description
`account`	The list of accounts this user is allowed to view. Data type may be either String or Name.
`ddn`	The application name (DDN). Data type is string.
`count`	Used to indicate the maximum number of rows of summary information to be returned. Zero returns all. Data type is short.
`from`	The “from” date determines from how far back in time to retrieve information. “from” is used in conjunction with the “to” date to control the amount of account summary information to retrieve. Data type is long.
`to`	The upper bound date used in retrieving information. Data type is long.

Results

This signature of **`getHitList`** returns the following table information:

Row Name	Description
Row[0]	Column headings
Row[1] .. Row[count]	Summary information for accounts
n	Number of columns
k	Number of optional fields

Column #	Column Heading	Description
0	Z_DOC_ID	Document identifier
1	Z_DOC_DATE	Document date
2	Z_IVN	Indexed volume number
3 .. (n-k –1)	Indexed field names	Value of indexed fields
n-k .. n-1	Optional field names (Y_1 to Y_n)	Value of optional fields

This method is very similar to the Content Access API method **`getSummary`**. The only difference is that it provides summary information for an account regardless of whether the IVN was accepted or rejected, while the **`app.user`** method restricts itself to accepted IVNs only. The **`app.verify`** method allows comparison of the information extracted, which is handy in determining the effect of changes in data stream parsing rules.

The above methods work together to help make a decision whether to accept or reject the processing results of an indexed volume. The next step is to actually accept or reject an indexed volume.

Accept or Reject an Indexed Volume

About acceptIndexedVolume and rejectIndexedVolume

After careful inspection, one makes a decision to accept or reject an indexed volume. Accordingly, one invokes the **`acceptIndexedVolume`** or **`rejectIndexedVolume`** method. Both methods take as their sole input the IVN number. Only accepted indexed volumes become available to the end-user. The code sample below illustrates one usage:

```
String ivn = request.getParameter(“ivn”);
     Verify.acceptIndexedVolume(ivn);
     System.out.println(“Accepted indexed volume: “ + ivn);
```

To ascertain whether you have accepted or rejected an indexed volume, invoke the **`getIndexedVolumeList`** method again and examine the *Z_DATE_ACCEPTED* and/or *Z_DATE_REJECTED* fields for the IVN of interest. For an accepted IVN the *Z_DATE_*ACCEPTED field will reflect the date on which you accepted it (the date itself is represented as a java long integer) and the *Z_DATE_REJECTED* field would be empty.

For example, the following fields would contain values for a DDN that has been accepted (the date values shown in these examples are only a “visual” representation of the actual values, which are number values):

Z_DATE_ACCEPTED	Z_DATE_REJECTED	Z_DATE_CREATED
2001-03-22		2001-03-21

If the method **`rejectIndexedVolume`** were subsequently called for this IVN then the fields would contain the following values:

Z_DATE_ACCEPTED	Z_DATE_REJECTED	Z_DATE_CREATED
	2001-03-22	2001-03-21

The reverse is true as well. Calling **`acceptIndexedVolume`** causes eaDirect to clear the *Z_DATE_REJECTED* value and add a *Z_DATE_ACCEPTED* value.

Tip The above calls might used with an HTML form submission if a user interface implements the Verify API methods.

Example

In the following example one IVN will be rejected, stamping the *Z_DATE_REJECTED* field with the current date and rendering the IVN or data stream unavailable for customer viewing:

```
<HTML>

<HEAD>

<TITLE>Reject IVN Example</TITLE>

</HEAD>

<%@ page import ="java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>

<%

String ivn = "2";    Verify.rejectIndexedVolume(ivn);
out.print("<B>" + "ivn: " + "</B>"+ ivn + " was marked as
Rejected!" + "<br>")

%>

</HTML>
```

Example

The following example will accept the IVN passed in, stamping the *Z_DATE_ACCEPTED* field with the current date and thus enabling end users to retrieve information from the billing cycle that IVN represents:

```
<HTML>

<HEAD>

<TITLE>Accept IVN Example</TITLE> </HEAD>

<%@ page import ="java.util.Properties, java.text.*,
com.edocs.app.verify.*, com.edocs.app.util.*" %>

<%
```

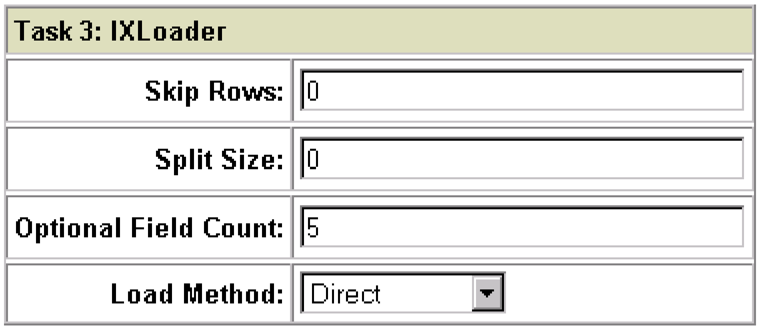
```
String ivn = "2";    Verify.acceptIndexedVolume(ivn);
out.print("<B>" + "ivn: " + "</B>"+ ivn + " was marked as accepted
and is ready for end users!" + "<br>")

%>

</HTML>
```

Update Summary Information

About updateSummaryInfo

An account may have information that is not extracted from the data stream. Such pieces of information, saved in optional fields in the eaDirect database, can be accessed using the methods **`user.getSummary`** and **`verify.getHitList.`**

These optional fields are defined in the Command Center as part of the **`IXLoader`** task of a job that allows you to enter a value for the Optional Field Count parameter. For example, if you want five optional fields you would enter:



Task 3: IXLoader

Skip Rows:	0
Split Size:	0
Optional Field Count:	5
Load Method:	Direct

An optional field may be **`DueDate`**. The application business rules may stipulate that the statement is payable twenty days from the date it is posted. That is a fixed length of time, and perhaps not one necessarily captured in the data stream. The due date in this case would be a function of the date the indexed volume is accepted.

Optional fields of this nature, and for that matter all of the optional fields, can be updated using the Verify API **`updateSummaryInfo`** method. (The Content Access API method **`updateSummaryInfo`** is similar.)

Parameters

Name	**Description**
`ivn`	The index volume number (IVN). It is unique to a processed data stream.
`docId`	The document identifier of the IVN. Obtain **`docId`** by calling **`getHitList`**.
`name`	The name of an optional field column, such as Y_1 or Y_2. The number selected during the **`IXLoader`** task determines the number of optional fields available. If you specify five optional fields, the names would range from Y_1 to Y_5.
`value`	The string value to assign to an optional field.

`Verify.updateSummaryInfo(String ivn, String docId, String name, String value)` requires the index volume number and the **`docId`**. Obtain these with one of the **`getHitList`** signatures. It also requires the name of the optional field and the value that must be assigned.

Example: Update Optional Field “Due Date”

The code sample below illustrates how it can be used in the context of an IVN that has the optional field “Y_1” that is associated with the semantics of a **`DueDate`**.

```
String ivn = “IVN_of_interest”;

Date today = Date();

long 20DaysForward = 20*24*60*60*1000; // in  milliseconds

String dueDate = “” + (new Date(today.getTime() +

20DaysForward)).getTime();

String docId = null; Verify.acceptIndexVolume(ivn);
```

```
// lets accept it  today!
  String[][] acctSummrys = Verify.getHitList(ivn, 0L, 0S);
// count == 0 => all accounts
  for (int i = 1; i< acctSummrys.length; i++) {
    docId =  acctSummrys [i][0];
    Verify.updateSummaryInfo(ivn,
                             docId,
                             "Y_1", // optional field
                             dueDate);
    }
```

Example: Update Optional Field to Accept Accounts

Another use for **`updateSummaryInfo`** is to update an optional field **`AcceptAccount`**. Its default value could be **`true`**, and if closer inspection of the account summary or detail indicates otherwise, it could be marked **`false`**. This can give you fine-grained control over processing results: acceptance or rejection at the account level. Application business logic may be specified that programmatically totals the number of account level rejects, determines whether it is within some acceptable threshold, and if so accepts the whole IVN.

Index