

Intelligent Offer Management (IOM),
Version 2.3.1

IOM Developer and Datafeed Reference

Notice

Intelligent Offer Management (IOM) Version 2.3.1

IOM Developer and Datafeed Reference

Copyright © 2003-2008 Nuance Communications, Inc. All rights reserved.

1198 E. Arques Avenue, Sunnyvale, CA 94085, U.S.A.

Printed in the United States of America.

Last updated 3/5/08.

Nuance Communications, Inc. provides this document without representation or warranty of any kind. The information in this document is subject to change without notice and does not represent a commitment by Nuance Communications, Inc. The software and/or databases described in this document are furnished under a license agreement and may be used or copied only in accordance with the terms of such license agreement. Without limiting the rights under copyright reserved herein, and except as permitted by such license agreement, no part of this document may be reproduced or transmitted in any form or by any means, including, without limitation, electronic, mechanical, photocopying, recording, or otherwise, or transferred to information storage and retrieval systems, without the prior written permission of Nuance Communications, Inc.

Nuance and the Nuance logo are trademarks or registered trademarks of Nuance Communications, Inc. or its affiliates in the United States and/or other countries. All other trademarks are the property of their respective owners.

Contents

CHAPTER 1	OVERVIEW TO IOM HTTP API AND DATAFEEDS	5
	IOM Documentation.....	5
CHAPTER 2	HTTP API REFERENCE	7
	Typical Calling Sequence	8
	HTTP Response Format and Status Codes	9
	create.do	11
	Syntax.....	11
	X-Schema Definition (XSD) for Custom Demographic Info	12
	Example of Optional Custom Demographic Info	13
	XML Success Response to create.do	13
	JSON Success Response to create.do	13
	destroy.do	14
	Syntax.....	14
	getPromotion.do	14
	Syntax.....	15
	XML Success Responses to getPromotion.do	15
	JSON Success Response to getPromotion.do	16
	getPromotionData.do.....	17
	Syntax.....	17
	refreshCache.do	18
	Syntax.....	18
	refreshFileCache.do	18
	Syntax.....	18
	reportPromotion.do	19
	Syntax.....	19
	XML Success Response to reportPromotion.do.....	19
	JSON Success Response to reportPromotion.do	20
	reportPurchase.do	20
	Syntax.....	20
	XML Success Response to reportPurchase.do	21
	JSON Success Response to reportPurchase.do.....	21
	update.do.....	21
	Destination Files	22

CHAPTER 3	DEVELOPING AN IOM DATAFEED.....	23
	Generalized Datafeed Process.....	23
	Data Model and Tables.....	24
	Data Feed Architecture.....	26
	Feed Parser Customization	27
	Properties Files.....	29
	Archive Process.....	29
	Alarms and Monitoring.....	29
	Data Feed Implementation Examples.....	30
	AT&T GoPhone PAYG	30
	Segment File Format for Cingular *NOW	36
APPENDIX A	API CONFIGURABLE PROPERTIES	39
	iomapi.properties	39
	JBoss properties-service.xml.....	39
	Cache Refresh Timeout.....	40
	Generated Alarms	40

Chapter 1 Overview to IOM HTTP API and Datafeeds

This reference describes the Intelligent Offer Management (IOM) HTTP Application Programming Interface (API) and how to set up datafeeds to transfer carrier back-end data to the IOM data warehouse. This book is part of the entire IOM documentation.

IOM Documentation

Intelligent Offer Management (IOM) is documented in the following books.

Book Title	Audience	Description
<i>IOM Concepts Guide</i>	All readers	<ul style="list-style-type: none">■ Overview concepts for IOM:<ul style="list-style-type: none">□ Theory of general operation□ Components of the IOM system□ Definitions of key terminology■ Tutorials of campaigns from start to finish:<ul style="list-style-type: none">□ Campaign design□ Implementing campaign objects in the IOM Tool□ Voice application design with a sample application■ Release notes for latest version of IOM
<i>IOM Tool User Guide</i>	Campaign administrators	<p>Step-by-step use of the IOM Tool for all tasks:</p> <ul style="list-style-type: none">■ Managing campaign objects■ Importing and exporting applications, campaigns, or offers■ IOM user set-up in C3
<i>IOM Developer and Datafeed Reference</i>	Voice application developers	<p>All details relating to programming IOM-based voice applications and IOM datafeed:</p> <ul style="list-style-type: none">■ Theory of programming, development methodology■ XML-over-HTTP API■ Developing and testing an IOM datafeed, alarms, and service monitors.

Book Title	Audience	Description
<i>IOM Legacy JSP ASP Reference</i>	Developers supporting IOM 1.6 legacy applications	Details related to the pre-IOM-2.3 legacy Java Server Pages Application Programming Interface.

Chapter 2 HTTP API Reference

The general syntax for invoking the Intelligent Offer Management (IOM) HTTP Application Programming Interface (API) is as follows:

```
http://host:port/services/iomapi[-2.3.0]/command.do?  
name1=value1&name2=value2&name3=value3...
```

where:

- *http://host:port* is the optional root of the request URL, depending on whether you are running the Intelligent Offer Management (IOM) on your local computer or not.

Note: Your application should use a variable for the root of the request URL, rather than hard-coding it.

- */command.do* is one of the following HTTP commands summarized below and detailed later in this chapter.
- *name1=value1&name2=value2&name3=value3...* are the name/value pairs appropriate for the HTTP command.

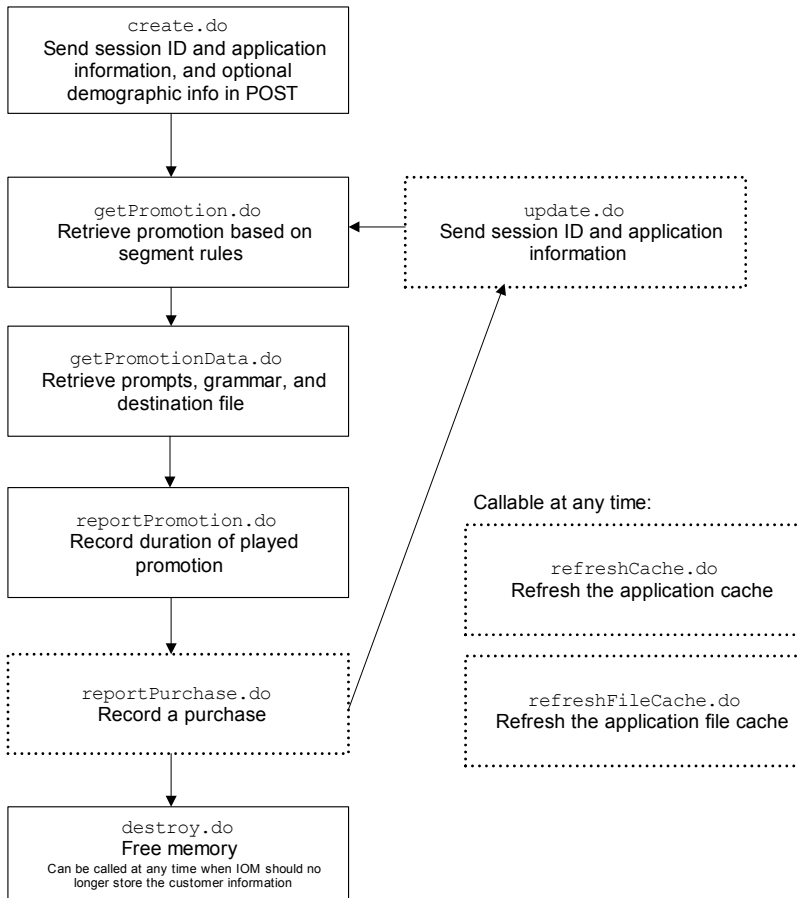
Table 1 Summary of HTTP API Commands

Command	Description
<i>/create.do</i>	Initialize IOM by sending session ID, application name, and customer info
<i>/destroy.do</i>	Free memory by removing the session cache
<i>/getPromotion.do</i>	Retrieve promotion by slot name and locale
<i>/getPromotionData.do</i>	Retrieve a promotion's associated files, such as prompts and grammar
<i>/refreshCache.do</i>	Immediately refresh the IOM cache
<i>/refreshFileCache.do</i>	Immediately refresh the IOM file cache
<i>/reportPromotion.do</i>	Record the duration of played promotion and whether the user accepted
<i>/reportPurchase.do</i>	Record a purchase
<i>/update.do</i>	Update the application and customer information

Typical Calling Sequence

The figure below illustrates the typical sequence for calling the IOM HTTP API commands. Optional commands are shown in dashed boxes.

Figure 1 IOM HTTP API Calling Sequence



Notes:



1. If `create.do` fails, do *not* call any other APIs.
2. Call `update.do` whenever your application has newly updated customer information.
3. Call `reportPurchase.do` whenever a product is purchased as a result of a promotion.
4. At the end of your application, call `destroy.do` to write report data to the database and to free-up memory.

HTTP Response Format and Status Codes

Responses are in either XML or JSON (JavaScript Object Notation). In a response, a value of 200 indicates a successful request. Failures are indicated by the other status codes shown in Table 2.

XML Failure Response	JSON Failure Response
<pre><?xml version="1.0" ?> <result> <item key="statusCode" value="200 or errorCode"/> <item key="statusMessage" value="errorMessage"/> </result></pre>	<pre><?xml version="1.0"?> <result> <![CDATA[iomResponse = { "statusMessage":"errorMessage", "statusCode":"200 or errorCode", };]]> </result></pre>

Table 2 HTTP API Status Codes and Messages

Code	Message	Possible Cause
200	Success	
600 to 699	IOM-specific Errors	
600	Failure	
601	PromotionPlayer not found	You may have called <code>update.do</code> or other APIs before calling <code>create.do</code> .
602	PromotionPlayer exists	You have called <code>create.do</code> more than once.
603	Slot not found	You called <code>getPromotion.do</code> with a slot name that does not exist.
604	Application not found	You called <code>create.do</code> with a application name that does not exist.
606	No promotion	All of the evaluation rules for <code>getPromotion.do</code> do not result in a match.
607	Maximum number of promotions has been reached.	You called <code>getPromotion.do</code> more than the limit specified for maximum number of promotions that can be played, as specified for the application in the field <code>Maximum # of promotions allowed per call</code> . See

Table 2 HTTP API Status Codes and Messages

Code	Message	Possible Cause
608	Application mismatch	You called an API with an application name that does not match the name you specified on <code>create.do</code> .
609	No promotion data	
610	Locale not supported	You specified a locale that this application does not support.

create.do


Initializes IOM session by sending the session ID, application name, and optional subscriber demographic information.

Syntax

```
http://host:port/iomapi/create.do?sessionId=
sessionId&appName=appname&format=format&did=did
< optional Demographic Info
in XML in HTTP POST body />
```

where:

Table 3 create.do Name/Value Pairs

Parameter		Description
sessionId=sessionId	Optional	The unique VXML session ID. If you do not pass a sessionId, IOM creates one for you. You must thereafter use this sessionId in subsequent calls. The response from create.do always contains the sessionId.
appName=appname	Required	The VXML application name.
format=format	Optional	Desired response format: ■ json ■ xml (default)
did=did	Optional	The DID of the hosted application.
<account> <number>min<number> <DemographicInfo> <demographicInfoItem> <key> segmentVariableName </key> <value> someValue </value> </demographicInfoItem> </DemographicInfo> </account>	Optional	<div></div> <p>Note: Sending your own demographic information for the subscriber is optional. If you send it, it must be in XML format in the body of a new HTTP POST with content-type: application/xml. See the XSD definition and example below.</p>

X-Schema Definition (XSD) for Custom Demographic Info

Use this XSD to build your IOM-based voice applications only if you intend to supply your own custom demographic information.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:iom="http://api.iom.product.bevocal.com"
  targetNamespace="http://api.iom.product.bevocal.com">
  <annotation>
    <documentation>This describes the schema for the Demographic
      Data implementation passed to the IOM HTTP interface service.
    </documentation>
  </annotation>

  <element name="account" type="iom:AccountInfo"/>
  <complexType name="AccountInfo">
    <sequence>
      <element name="number" type="string" minOccurs="1" maxOccurs="1"/>
      <element name="demographicInfo" type="iom:DemographicInfo"
        minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="DemographicInfo">
    <sequence>
      <element minOccurs="0" maxOccurs="unbounded" name="demographicInfoItem"
        type='iom:DemographicInfoItem'/>
    </sequence>
  </complexType>

  <complexType name="DemographicInfoItem">
    <sequence>
      <element minOccurs="1" name="key" type="string"/>
      <element minOccurs="1" name="value" type="anySimpleType"/>
    </sequence>
  </complexType>
</schema>
```

Table 4 create.do Custom DemographicInfo Elements

Parameter	Description
number	The subscriber's phone number or account number.
key	The name of the IOM segment variable. Must be the name of a segment variable created with the IOM Tool; see the <i>IOM Tool User Guide</i> .
value	Any value you want to pass. This can be a multi-valued, comma-delimited string, with no spaces.

Example of Optional Custom Demographic Info

This XML is contained in the HTTP POST body on `create.do`.

```
<account>
  <number>8773386255</number>
  <demographicInfo>
    <demographicInfoItem>
      <key>ACCOUNT_BALANCE</key>
      <value>100</value>
    </demographicInfoItem>
  </demographicInfo>
</account>
```

XML Success Response to `create.do`

```
<?xml version="1.0"?>
<result>
  <item key="statusCode" value="200"/>
  <item key="statusMessage" value=""/>
  <item key="sessionId"
    value="84550dee-016a-41bd-a2e5-47d62d51d437"/>
</result>
```

JSON Success Response to `create.do`

```
<?xml version="1.0"?>
<result>
  <![CDATA[
    {
      "statusMessage": "",
      "sessionId": "16949b34-d90f-4560-9faa-b2243b2b0c44",
      "statusCode": "200"
    }
  ]]>
</result>
```

destroy.do

Frees memory by removing the session cache. `destroy.do` can be called at any time when you no longer want IOM to store the customer or application information.

Syntax

```
http://host:port/iomapi/destroy.do?sessionId=sessionid&format=format
```

Table 5 destroy.do Name/Value Pairs

Parameter		Description
<code>sessionId=<i>sessionid</i></code>	Required	The unique VXML session ID returned from the <code>create.do</code> API.
<code>format=<i>format</i></code>	Optional	Desired response format: <ul style="list-style-type: none">■ <code>json</code>■ <code>xml</code> (default)

getPromotion.do

Returns the name of a promotion, the names of its associated prompts, grammar, and destination file, the URLs for the `getPromotionData.do` command to retrieve them, and the characteristics of the promotion, such as its type (interactive or informational), timeout settings, and so forth.

For example, in the response to `getPromotion.do`, the initial prompt for a promotion is represented like so (in JSON format):

```
.  
. .  
.  
"initialName": "promo_sweeps_info.wav",  
"initialUrl": "getPromotionData.do?promotionId=3181&resourceType=initialPrompt&locale=en-US",  
. .  
.
```

After the response from `getPromotion.do`, you can thus pass the value of `initialUrl` on the next HTTP GET.

Syntax

```
http://host:port/iomapi/getPromotion.do?
    sessionId=sessionId&appName=appname&
    slot=slotName&locale=loc&format=format
```

where:

Table 6 getPromotion.do Name/Value Pairs

Name/Value Pair		Description
sessionId=sessionId	Required	The unique VXML session ID
appName=appname	Required	The VXML application name. Must match the promotion's application name.
slot=slotName	Required	The name of the slot.
locale=loc	Required	loc is one of the application's supported locales: en-US, es-US, or fr-CA
format=format	Optional	Desired response format: <ul style="list-style-type: none">■ json■ xml (default)

XML Success Responses to getPromotion.do

```
<?xml version="1.0" ?>
<result>
  <item key="statusCode" value="200" />
  <item key="statusMessage" value="" />
  <item key="promotionName" value="Accessories Promotion" />
  <item key="promotionId" value="12950">
  <item key="productCode" value="ACCESSORIES">
  <item key="initialName" value="iom_accessories.wav">
  <item key="initialUrl" value="getPromotionData.do?promotionI
d=12950&resourceType=initialPrompt&locale=en-US">
  <item key="acceptName" value="okay_hang_on.wav">
  <item key="acceptUrl"
value="getPromotionData.do?promotionId=12950&locale=en-US&resourceType=
accept">
  <item key="errorName" value="didnt_understand_0.wav">
  <item key="errorUrl"
value="getPromotionData.do?promotionId=12950&locale=en-US&resourceType=
error">
  <item key="declineName" value="okay_no_problem.wav">
  <item key="declineUrl"
value="getPromotionData.do?promotionId=12950&locale=en-US&resourceType=
decline">
  <item key="destinationName" value="iom_accessories_dest.xml">
```

```

    <item key="destinationUrl"
value="getPromotionData.do?promotionId=12950&locale=en-US&resourceType=
dest_mappings">
    <item key="grammarName" value="yesno.grammar">
    <item key="grammarUrl"
value="getPromotionData.do?promotionId=12950&locale=en-US&resourceType=
grammar">
    <item key="helpName" value="">
    <item key="helpUrl" value="">
    <item key="timeoutValue" value="4s">
    <item key="interactive" value="true">
    <item key="bargainable" value="false">
</result>

```

JSON Success Response to getPromotion.do

```

<?xml version="1.0"?>
<result>
<![CDATA[{
"statusCode":"200",
"statusMessage":"",
"promotionName":"Accessories Promotion",
"promotionId":"12950",
"productCode":"ACCESSORIES",
"initialName":"iom_accessories.wav",
"initialUrl":"getPromotionData.do?promotionId=12950&locale=en-US&resour
ceType=initial",
"acceptName":"okay_hang_on.wav",
"acceptUrl":"getPromotionData.do?promotionId=12950&locale=en-US&resourc
eType=accept",
"errorName":"didnt_understand_0.wav",
"errorUrl":"getPromotionData.do?promotionId=12950&locale=en-US&resource
Type=error",
"declineName":"okay_no_problem.wav",
"declineUrl":"getPromotionData.do?promotionId=12950&locale=en-US&resour
ceType=decline",
"destinationName":"iom_accessories_dest.xml",
"destinationUrl":"getPromotionData.do?promotionId=12950&locale=en-US&re
sourceType=dest_mappings",
"grammarName":"yesno.grammar",
"grammarUrl":"getPromotionData.do?promotionId=12950&locale=en-US&resour
ceType=grammar",
"helpName":"",
"helpUrl":"",
"timeoutValue":"4s",
"interactive":"true",
"bargainable":"false"
}]]></result>

```


getPromotionData.do

Returns a prompt, grammar, or destination file. For usage, see the discussion of the response from “getPromotion.do” on page 14.

For file caching, it is recommended that you use HTTP GET, not POST.

Syntax

```
http://host:port/iomapi/getPromotionData.do?  
  sessionId=sessionid&appName=appname&  
  promotionId=promotionId&  
  resourceType=desiredResource&  
  locale=loc&format=format
```

where:

Table 7 getPromotionData.do Name/Value Pairs

Name/Value Pair		Description
sessionId=sessionid	Required	The unique VXML session ID
appName=appname	Required	The VXML application name. Must match the promotion’s application name.
promotionId=promotionId	Required	Numerical identifier of the promotion. This is the value of the promotionId name/value pair returned in the response from getPromotion.do; see “getPromotion.do” on page 14.
resourceType=desiredResource	Required	The desired file. Allowable values are as follows: <ul style="list-style-type: none">■ initialPrompt■ errorPrompt■ acceptPrompt■ helpPrompt■ grammar■ dest_mappings
locale=loc	Required	loc is one of the application’s supported locales: en-US, es-US, or fr-CA
format=format	Optional	Desired response format: <ul style="list-style-type: none">■ json■ xml (default)

refreshCache.do

Immediately refresh the memory cache. By default, results are cached for 10 minutes.

Syntax

http://host:port/iomapi/refreshCache.do?appName=appname

where the optional `appName` value is the VXML application name. If `appName` is not specified, the cache is refreshed for all active applications.

refreshFileCache.do

Immediately refresh the file cache. By default, results are cached for 10 minutes.

Syntax

*http://host:port/iomapi/refreshFileCache.do?
appName=appname*

where the optional `appName` value is the VXML application name. If `appName` is not specified, the file cache is refreshed for all active applications.

reportPromotion.do

Records the results of a played promotion.

Syntax

```
http://host:port/iomapi/reportPromotion.do?  
  sessionId=sessionid&appName=appname&  
  promotionId=promotionId&  
  duration=milliseconds&isAccepted=true | false&format=for  
  mat
```

where:

Table 8 reportPromotion.do Name/Value Pairs

Name/Value Pair		Description
<code>sessionId=<i>sessionid</i></code>	Required	The unique VXML session ID
<code>appName=<i>appname</i></code>	Required	The VXML application name. Must match the promotion's application name.
<code>promotionId=<i>promotionId</i></code>	Required	Numerical identifier of the promotion. This is the value of the <code>promotionId</code> name/value pair returned in the response from <code>getPromotion.do</code> ; see “ <code>getPromotion.do</code> ” on page 14.
<code>duration=<i>milliseconds</i></code>	Required	Length of time in milliseconds that the promotion was played.
<code>isAccepted=<i>true false</i></code>	Required	Caller's response to the promotion
<code>format=<i>format</i></code>	Optional	Desired response format: <ul style="list-style-type: none">■ <code>json</code>■ <code>xml</code> (default)

XML Success Response to reportPromotion.do

```
<?xml version="1.0" ?>  
<result>  
  <item key="statusCode" value="200" />  
  <item key="statusMessage" value=""/>  
</result>
```

JSON Success Response to reportPromotion.do

```
<?xml version="1.0"?>
<result>
<![CDATA[
iomResponse =
{
    "statusMessage": "",
    "statusCode": "200",
};
]]>
</result>
```

reportPurchase.do

Records the caller's purchase of a product or service.

Syntax

```
http://host:port/iomapi/reportPurchase.do?
    sessionId=sessionId&appName=appname&
    promotionId=promotionId&price=price&
    productCode=code&isPurchased=true|false&format=format
```

where:

Table 9 reportPromotion.do Name/Value Pairs

Name/Value Pair		Description
sessionId=sessionId	Required	The unique VXML session ID
appName=appname	Required	The VXML application name. Must match the promotion's application name.
promotionId=promotionId	Required	Numerical identifier of the promotion. This is the value of the promotionId name/value pair returned in the response from getPromotion.do; see "getPromotion.do" on page 14.
price=price	Optional	The cost of the product
productCode=code	Optional	Identification code of the product or service.

Table 9 reportPromotion.do Name/Value Pairs

Name/Value Pair		Description
isPurchased= true false	Required	Purchase status. Default is false.
format= <i>format</i>	Optional	Desired response format: <ul style="list-style-type: none"> ■ json ■ xml (default)

XML Success Response to reportPurchase.do

```
<?xml version="1.0" ?>
<result>
  <item key="statusCode" value="200" />
  <item key="statusMessage" value=""/>
</result>
```

JSON Success Response to reportPurchase.do

```
<?xml version="1.0"?>
<result>
<![CDATA[
iomResponse =
{
  "statusMessage": "",
  "statusCode": "200",
};
]]>
</result>
```

update.do

Updates the application and customer information.



Note: The usage, parameters, and responses of `update.do` are identical to those of `create.do`. If you sent your own custom demographic information on the original `create.do` for this session, you must include it on `update.do`. See “`create.do`” on page 11.

Destination Files

A destination file is a file uploaded to IOM for a specific promotion with the IOM Tool (see Table 1, “Promotion Fields,” on page 28 in the *IOM Tool User Guide*).

For each promotion, the destination map is composed of key/value pairs of grammar response values and the corresponding forms or subdialogs to which IOM should transfer after the responses have been interpreted.



Note: Theoretically, the value in a destination map can be anything your voice application needs to operate on. For example, it could be a product code or a promotion code. In many common uses, however, the value is most often a form or subdialog name.

The following is a simple destination map with only a single item:

```
<?xml version="1.0"?>
<config>
  <item key="yes" value="Purchase_Feature_Dialog" />
</config>
```

A destination map can have multiple items:

```
<?xml version="1.0"?>
<config>
  <item key="1" value="Purchase_Prod1_Dialog" />
  <item key="2" value="Purchase_Prod2_Dialog" />
  <item key="All" value="Purchase_Everything_Dialog" />
  <item key="no" value="Thankyou_and_Goodbye_Form" />
</config>
```

For an example of using destination files in a voice application, see the tutorials in the *IOM Concepts Guide*.

Chapter 3 Developing an IOM Datafeed

Intelligent Offer Management (IOM) uses external datafeeds to the populate usage data to define segments for promotions.

The datafeed update framework is in Java, with wrapper shell scripts. The core Java archive file is in the IOM installation at:

installDir/lib/datafeed-1.0.0.jar

In addition to the Java Runtime Engine, the following Java-based software is also required.

Table 1 Required Java Software for Datafeed

Component	Description
bcpg-jdk14-133.jar	Bouncy Castle JCE Implementation
bcprov-jdk14-133.jar	Bouncy Castle PGP Implementation
bvc-util-1.0.2.jar	Alarms
commons-logging-1.0.4.jar	Logging
commons-net-1.4.1.jar	Apache Jakarta common networking
edtftpj-1.5.2.jar	FTP client
jakarta-oro-2.0.8.jar	Apache Jakarta regular expressions
ojdbc14.zip	JDBC driver for Oracle



Note: Make sure you have set your CLASSPATH to include the absolute paths to all of the above software.

Generalized Datafeed Process

At its most general, the IOM datafeed process is as follows.

- **Programming**
 - The data (content) and format of a data file are determined in advance. There is no one single format; the contents depend on the purpose of the datafeed.

- A program is written with the IOM datafeed Java classes that will parse the datafile and put the data into the IOM datawarehouse.
- **Processing**
 - At predetermined points, a file is transferred to the IOM FTP service for processing.
 - Once the file is received, it is processed through a series of scripts.

Data Model and Tables

IOM has two database tables for usage data:

- IOM_USAGE_HISTORY stores raw usage data.
- IOM_USAGE_SUMMARY stores summarized data.

For example, Ericsson usage data is uploaded on a weekly basis and the raw data is stored in IOM_USAGE_HISTORY. Four weeks of data is then summed and stored in IOM_USAGE_SUMMARY.

The IOM_USAGE_HISTORY table stores the raw history data obtained from carriers and is not accessed directly by IOM. There is a separate, regularly scheduled process to update this table with data uploaded by the service providers via the FTP service. See “Shell Scripts” on page 35.

Table 2 IOM_USAGE_HISTORY

Column	Type and Length	Explanation
MIN_ID	VARCHAR2 (20) NOT NULL	MIN of caller
CARRIER	VARCHAR2 (30)	Name of Carrier
TYPE	VARCHAR2 (50) NOT NULL	Type of call
DATA_FIELD1	VARCHAR2 (50)	Feed specific data
DATA_FIELD2	VARCHAR2 (50)	Feed specific data
START_DATE	DATE NOT NULL	Start date of the feed segment
END_DATE	DATE NOT NULL	End date of the feed segment

Table 2 IOM_USAGE_HISTORY

Column	Type and Length	Explanation
SOURCE	VARCHAR2 (30)	Source of the feed
LAST_ MODIFIED_ DATE	DATE DEFAULT SYSDATE	Last modified date of the record

The IOM_USAGE_SUMMARY table contains data aggregated from the IOM_USAGE_HISTORY table and used by the IOM Engine.

Table 3 IOM_USAGE_SUMMARY

Column	Type and Length	Explanation
MIN_ID	VARCHAR2 (20) NOT NULL	MIN of caller
TYPE	VARCHAR2 (50)	Type of call
DURATION_ TIME	NUMBER (10)	Duration of call
DURATION_ UNIT	VARCHAR2 (10)	Unit of Duration
VARIABLE_ NAME	VARCHAR2 (20) NOT NULL	Usage variable as defined by
VARIABLE_ VALUE	VARCHAR2 (50) NOT NULL	Value of variable
LAST_ MODIFIED_ DATE	DATE DEFAULT SYSDATE	Last modified date of the record

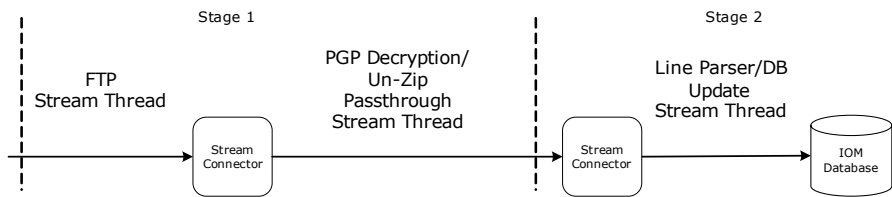
Data Feed Architecture

A typical IOM datafeed does the following:

- Uploads the data from an external source (service provider) via FTP.
- Processes the data.
- Populates the IOM usage tables in the database.

The IOM datafeed takes place in two stages:

1. The first stage receives remote data as an FTP stream. It either decrypts the data with PGP and unpacks it with ZIP, or just passes the data to the next stage.
2. The second stage parses the output from the first stage and uploads it to the IOM database.

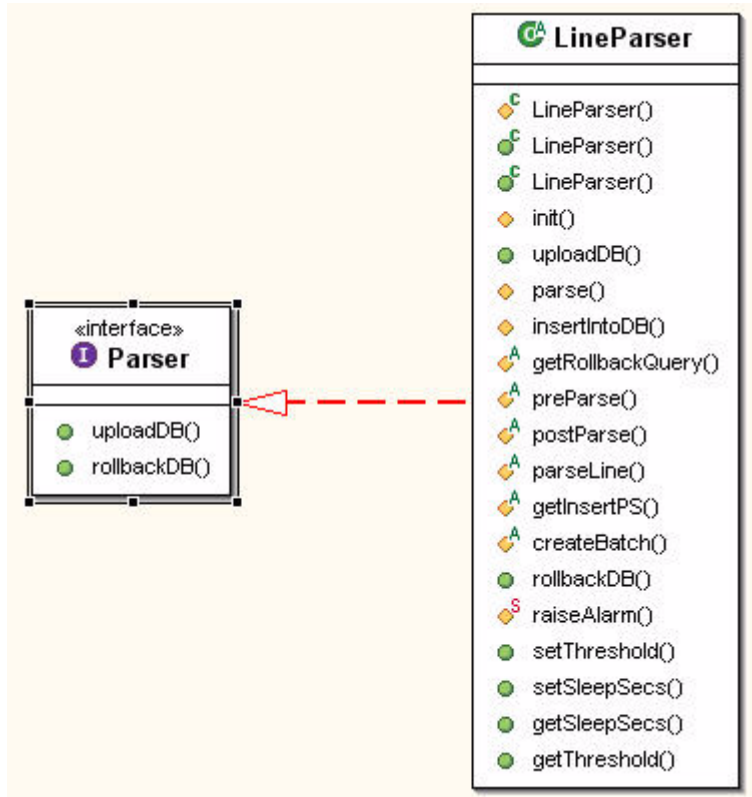


The core layer is the Stream Connector, which allows streams to be connected as producer (input) and consumer (output). Multiple streams can be chained together in stages for multi-level processing. Each input or output stream runs in its own thread to avoid potential deadlocks, such as read operation blocks waiting for more data.

Feed Parser Customization

The IOM datafeed architecture offers many customization points, though currently only a simple line parser has been implemented.

The line-based parser assumes the content is divided into lines which are then tokenized, processed, and inserted into the target database. The following classes are defined.



You must create an implementation class, which implements the `Parser` interface, and the two methods listed in Table 4 must be defined.

Table 4 Parser interface methods to be defined

Method	Description
<code>uploadDB()</code>	An <code>InputStream</code> argument is given as input data in raw bytes.
<code>rollbackDB()</code>	Called to rollback the database if the parse process fails.

The `LineParser` abstract class needs to be extended by defining the abstract methods below.

Table 5 Abstract methods for the `LineParser` class

Method	Description
<code>getRollbackQuery()</code>	Defines the query to execute to rollback the database if the parse process fails.
<code>preParse()</code>	Called before the parser starts.
<code>postParse()</code>	Called after the parser finishes.
<code>parseLine()</code>	Called with a text line argument. This function is called once per line.
<code>getInsertPS()</code>	Returns a <code>PreparedStatement</code> to be executed to insert records into the database.
<code>createBatch()</code>	Populates the <code>PreparedStatement</code> returned by <code>getInsertPS()</code> . The number of arguments and data values must match. All data must be added with the <code>PreparedStatement.addBatch()</code> function. The <code>PreparedStatement.executeBatch()</code> function is used to insert all data into the database.
Non-line based parser	Data is not given on line basis.

Properties Files

The datafeed parsing scripts rely on customer-specific properties files to control behavior, specify environments, and provide connectivity information. Two sets of parameters are used to allow the upload processes to be configured separately.

- `threshold`, `sleepSecs` and `maxRetries` parameters relate to the history table.
- `thresholdSummary`, `sleepSecsSummary`, and `maxRetriesSummary` relate to the summary table.

For examples of the properties files, with its keys and values, see “Ericsson Properties File” on page 32 and “Vesta Properties File” on page 34.

There are two other properties files.

Table 6 Additional Properties files

File	Description
<code>driver.properties</code>	Database connectivity parameters
<code>log4j.properties</code>	Logging parameters

Archive Process

The uploaded data file will be moved to an archive directory after successful completion of the update process. The `ftp.archiveDirectory` property is used to specify the archive directory name where the data file will be moved.

Alarms and Monitoring

The Nuance NOC uses industry standard tools like HP OpenView as well as internally developed tools to monitor and check the status and health of Nuance’s applications and infrastructure. The IOM Data Feed architecture makes use of one of these internally developed tools called Service Monitor to allow the monitoring of the status and outcome of regularly running feeds.

The service monitor provides a mechanism to call a URL and to check the results of the call for known values. The underlying datafeed must provide the information from its sources, like its log files, the database or other means by which the status of the feed can be retrieved and compared with expected values. For more detailed information, see “Data Feed Implementation Examples” on page 30 and “Alarms and Monitoring” on page 29.

Data Feed Implementation Examples

This section details the following:

- The datafeed implementation for AT&T’s GoPhone Pay As You Go (PAYG).
- The segment file format used for Cingular *NOW

AT&T GoPhone PAYG

The AT&T GoPhone PAYG application has the following two datafeeds:

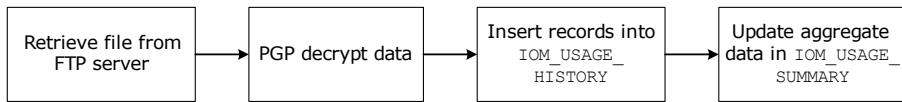
- The Ericsson datafeed provides subscriber usage information. This information is aggregated and summarized over 4 week chunks for use in IOM segments.
- The Vesta feed provides pre-summarized billing information in the form of subscribers who have paid for their prepaid accounts with credit cards over the last 90 days. This information is not summarized again after retrieval.

One piece of information is used from the each feed.

- The Ericsson information, after summarization, represents the total prepaid minutes used over a 4 week period. The IOM segment variable that represents this information in the segment is `TotalMinutes`.
- The Vesta information is taken directly from the feed and dropped directly into the `IOM_USAGE_SUMMARY` table. The IOM segment variable that represents this information is called `TotalTimes`.

Ericsson Data

This is a regular weekly process every Sunday at 9:30PM PST.



Details of feed processing:

1. Iterate over each line of the decrypted content.
2. Insert new records, along with start and end dates, into the `IOM_USAGE_HISTORY` table. The insert is done in groups of *threshold* rows followed by a pause of *sleepSecs* seconds
If Step 2 fails, roll back by deleting records with the same start and end dates. The process is then aborted, an alarm is raised, and the summary table is not updated. The existing data is still intact but is outdated by one week.
3. Aggregate data from the `IOM_USAGE_HISTORY` table by using the start and end dates for the last four weeks
4. Loop over the result set from step 3 and update the `IOM_USAGE_SUMMARY` table.
This update is done by performing a delete followed by insert, with the both the delete and insert being done in one transaction. Note that the record may or may not exist in the database prior to the update.
The delete/insert transactions are executed in groups of *thresholdSummary* rows followed by pause of *sleepSecsSummary* seconds.
If the transaction fails, it is retried *maxRetriesSummary* times. If the transaction still fails after *maxRetries* attempts, the process is aborted, At this point, the summary table is partially updated, and an alarm is raised.
5. Remove inactive summary records - last modified date is older than 4 weeks - if failed, process aborts, summary table may contain old data, and an alarm is raised
6. Update the `LastUpdated` column with the current date for the `IOM_DATA_ERICSSON` entry in the `DATEFEEDUPDATETIMES` table

If the update process fails and the text “summary table is out-of-synch” is in the error message, the part to update the `IOM_USAGE_HISTORY` table has been completed successfully, the summary step starting from Step 5 (above) needs to be repeated by running the `runFTPericssonSummary.sh` script manually. Otherwise, the entire process needs to be repeated by running the `runFTPericsson.sh` script manually.

A service monitor has been installed to monitor the `IOM_DATA_ERICSSON` entry. The status turns red if the value of `LastUpdated` is more than one week old.

Ericsson Data File Format

The Ericsson system provides a data file consisting of over 100 fields separated by spaces. Because some of the information is confidential, the datafeed file is encrypted using PGP encryption before being transferred to Nuance.

After decrypting and validating that the file was transferred correctly, the Ericsson datafeed parses and sums up the data fields that create the `TotalMinutes` usage information.

Ericsson Properties File

The `ericsson.properties` file configures various parameters for the database update process.

```
environment = engineering
# parsing parameters
delim = |
threshold = 5000
sleepSecs = 1
maxRetries = 3

# summary parameters
thresholdSummary = 5000
sleepSecsSummary = 1
maxRetriesSummary = 3

# table names in DB
historyTable = IOM_USAGE_HISTORY
summaryTable = IOM_USAGE_SUMMARY

# firstDB - for the history table
username = [not shown]
```



```

password = [not shown]
url = jdbc:oracle:thin:@10.0.100.86:1526:BVAPPS

# secondDB - for the summary table
username2 = [not shown]
password2 = [not shown]
url2 = jdbc:oracle:thin:@10.0.100.86:1526:BVAPPS

#alarming
alarmDest = public/mgupta-d.bevocal.com:162

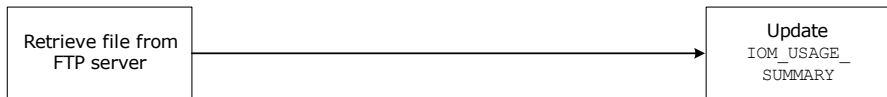
# summary parameters
backDaysValue = 28
source = Ericsson
carrier = GoPhone
historyType = Total Minutes

# ftp parameters
ftp.host=bvcaftp001
ftp.user=[not shown]
ftp.password=[not shown]
ftp.archiveDirectory=archive/vesta

```

Vesta Data

This is a regular weekly process every Wednesday at 9:30PM PST.



1. Data goes directly to the `IOM_USAGE_SUMMARY` table because no aggregation is needed
2. Remove all Vesta data from the `IOM_USAGE_SUMMARY` table. New data is a complete set. If failed, process aborts and old data in the summary table is still intact, and an alarm is raised.
3. Insert new records in groups of *threshold* rows followed by a pause of *sleepSecs* seconds. If failed, process aborts, summary table is partially updated, and an alarm is raised.
4. Update the `LastUpdated` column with the current date for the `IOM_DATA_VESTA` entry in the `DATEFEEDUPDATETIMES` table

If the update process fails, it must be repeated by running the `runFTPVesta.sh` script manually.

A service monitor has been installed to monitor the IOM_DATA_VESTA entry. The status turns red if the value of LastUpdated is more than 1 week old.

Vesta Data File Format

The Vesta system provides a data file consisting of a list of MINs. Each entry represents a wireless subscriber that has used a credit card to refill their prepaid wireless account more than once within the last 90 days.

Vesta Properties File

The `vesta.properties` file configures various parameters for the database update process.

```
environment = engineering
# parsing parameters
threshold = 5000
sleepSecs = 1
maxRetries = 3

# table names in DB
historyTable = IOM_USAGE_HISTORY
summaryTable = IOM_USAGE_SUMMARY

# firstDB - data goes directly to the summary table
username = [not shown]
password = [not shown]
url = jdbc:oracle:thin:@10.0.100.86:1526:BVAPPS

# alarming
alarmDest = public/mgupta-d.bevocal.com:162

# summary parameters
summaryType = Upsell AutoPay
variableName = TotalTimes
durationUnit = days
backDaysValue = 90

# ftp parameters
ftp.host=bvcaftp001
ftp.user=uploaduser
ftp.password=g04bvocal
ftp.archiveDirectory=archive/vesta
```

Shell Scripts

The following shell scripts process the Ericsson and Vesta feeds. These scripts do not require any command line parameters and get their configuration information from their respective property files.

Table 7 Shell Scripts

Script	Description
<code>runFTPEricsson.sh</code>	shell script to invoke the Ericsson upload process
<code>runFTPEricssonSummary.sh</code>	shell script to invoke the Ericsson summary process
<code>runFTPVesta.sh</code>	shell script to invoke the Vesta upload process

Cron Setup

The following are the crontab entries used to schedule the Ericsson and Vesta feed processes.

```
# Ericsson's auto update script
30 21 * * sun sh absolutePath/runFTPEricsson.sh
# Vesta's auto update script
30 21 * * wed sh absolutePath/runFTPVesta.sh
```

Output is sent to:

```
absolutePath/log/[ericsson|vesta]/
log[Ericsson|Vesta]_month_day_year_hour_minute_second.log
```

Service Monitor Setup

The following two service monitor entries monitor the two datafeeds. *hostname* is the actual name of the service monitor host.

Name	IOMVestaFeed
Monitor Settings	<code>http://<i>hostname</i>/appsupporttools/feeds_monitoring/checkgeneric.jsp?servicename=IOM_DATA_VESTA&tsecs=612000 SERVICE-UP.*</code>
Up frequency	43200000 (12 hours)
Down frequency	3600000 (1 hour)
Record Status	Yes

Name	IOMEricssonFeed
Monitor Settings	<code>http://hostname/appsupporttools/feeds_monitoring/checkgeneric.jsp?servicename=IOM_DATA_ERICSSON&tsecs=612000 SERVICE-UP.*</code>
Up frequency	43200000 (12 hours)
Down frequency	3600000 (1 hour)
Record Status	Yes

If any of the services is down, it implies a problem with the datafeed process. Examine the log file (see “Cron Setup” on page 35).

FTP Server Failures

If the primary FTP server fails and the datafeed file has been saved to another FTP server. The properties named `ftp.*` can be changed in the properties file to correspond to the new FTP server and the update scripts need to be invoked manually.

Segment File Format for Cingular *NOW

The Cingular *NOW datafeed implementation relies on FTP scripts, cron, shell scripts, alarms, and service monitoring similar to the earlier examples.

The data file passed for Cingular *NOW is called a *fulfillment segment file* and is an example of the data needed to implement the IOM datawarehouse for a fulfillment segment. For background information, see the *IOM Concepts Guide* and *IOM Tool User Guide*.

The data are stored in a flat text file named with the following naming convention:

`productcode_startdate_enddate.extension`

where:

<i>productcode</i>	The Cingular product code used for provisioning the product. Product codes should not contain underscores or periods. The current set we have from Cingular conforms to this format (e.g. MBP1, MBP2, 4N1S, MN09).
--------------------	--

<i>startdate</i>	The starting date of the campaign encoded in 8 digits in year, month, day order (e.g. 20060810 representing August 10, 2006)
<i>enddate</i>	The ending date of the campaign encoded in 8 digits in year, month, day order (e.g. 20060825 representing August 25, 2006)
<i>extension</i>	The encoding format of the file. This must be one of the following: .zip – File compressed with ZIP compression .pgp – File encoded in PGP encryption .txt – Plain text. Note if this option is used, an MD5 checksum must be included either in the ECR or the notification to the NOC for verification purposes.

The data in the file consists of telephone numbers that represents the subscribers at whom the fulfillment campaign was targeted. The following is a sample of the data contents of the file:

```
4083943920
6509320920
4150923422
.
.
.
```


Appendix A API Configurable Properties



The Intelligent Offer Management (IOM) HTTP API relies on a property files to determine its behavior in sending alarms:

- `iomapi.properties`
- `JBoss properties-service.xml`
- Generated Alarms

iomapi.properties

The `iomapi.properties` configuration file contains settings related to alarms and is installed in the application server's `WEB-INF` directory where the Intelligent Offer Management (IOM) is installed.

```
#alarming
isAlarmDisable=false
alarmName = iomapi
serviceVersion = 230
serviceName = iomapi
## production alarmDest
alarmDest = 20net1Sec/encact1001:162
apiTimeLimit = 500
```

JBoss properties-service.xml

Some settings can be overridden in the configuration file `/usr/local/jboss/server/bevocal/deploy/properties-service.xml` used by the JBoss application server. These Jboss settings are contained in the `<attribute name="Properties">` section of the configuration file.

Key in <code>iomapi.properties</code>	Equivalent in <code>properties-service.xml</code>
<code>alarmDest</code>	<code>bevocal.snmp.destination</code>
<code>apiTimeLimit</code>	<code>iomapi.alarm.apitimelimit</code>

Key in iomapi.properties	Equivalent in properties-service.xml
none	iomapi.cache.refreshtimeout See discussion in “Cache Refresh Timeout” on page 40.

Example

For example, in a development environment, you might want to set the alarm destination to somewhere other than the destination used in production. You could set the destination in properties-services.vxml like the following.

```
bevocal.snmp.destination=20net1Sec/encact9999:999
```

Cache Refresh Timeout

The default time period for clearing and refreshing the IOM API cache is every 24 hours. You can set the frequency of cache refresh in properties-service.xml:

```
iomapi.cache.refreshtimeout = valueInMilliseconds
```

For example, the following sets the cache refresh time frequency to every two hours:

```
iomapi.cache.refreshtimeout = 7200000
```

Generated Alarms

The IOM API generates alarms when error conditions occur during the execution of any of the APIs. Warning alarms are generated for minor errors like missing required fields. Error alarms are generated when other exceptions are thrown by the player, such as “Promotion player not found” or “Slot not found”.

Generated Alarm	Action
Warning: Missing required field 'Field1, Field2'	Please check the parameters to make sure that all the required fields have been populated.
Warning: TimeLimitExceeded	The API handleRequest took XX milliseconds to execute. Please inform the engineering development team about the slow performance.

Generated Alarm	Action
Error: Could not find PromotionPlayer for sessionID	PromotionPlayer was not found for this sessionID. Any reporting data has been lost.
Error: PromotionPlayer for sessionID "XX" exists	The PromotionPlayer create() api is being called twice. Make sure destroy() is called before calling create() api again.
Error: Could not find slot 'slotName'	Please check that the correct slot name is being used
Error: Application 'appName' not found.	Please check that the appName being used exists
Error: All Reports have been written.	WriteReports can only be called once to avoid any report duplication. Please check that write reports is not being called more than once
Error: Parameter 'appName' does not match the appName 'appName1' used in creating PromotionPlayer for sessionID 'sessionID'	Please check that the same app is being invoked
Error: No file of type 'type' and locale 'locale' found for promoId = 'promotionId'	Please check that the promotion with 'promotionId' has valid content in the database for 'type' and 'locale'

Index

A

alarmDest 39
alarmName 39
API. See “HTTP API.”
apiTimeLimit 39
application/xml 11

C

Cache refresh rate 40
CLASSPATH
 for IOM Datafeed software 23
create.do 7
 responses 13
 status codes 9
 XML format of request 13

D

Data warehouse
 core IOM JAR file 23
 FTP 26
 IOM_USAGE_HISTORY 24
 IOM_USAGE_SUMMARY 25
 LineParser class 28
 properties files 29
Datafeeds. See “Data warehouse.”
DemographicInfo 12
DemographicInfo on create.do 11
Destination file 22
destroy.do 7, 14

F

FTP 26
Fulfillment segment
 example of datafeed file format 36

G

GET 14, 17
getPromotion.do 7, 15
getPromotionData.do 7, 14, 17

H

HTTP API 7
 create.do 7
 responses 13
 status codes 9
 XML format of request 13
 destroy.do 7, 14
 getPromotion.do 7, 15
 getPromotionData.do 7, 14, 17
 refreshCache.do 7, 18
 refreshFileCache.do 7
 reportPromotion.do 7, 19, 20
 reportPurchase.do 7
 request URL 7
 update.do 7

I

IOM_USAGE_HISTORY table 24

IOM_USAGE_SUMMARY table 25
iomapi.properties 39
isAlarmDisable 39

J

Java Runtime Engine 23
JavaScript Object Notation. See “JSON.”
JSON 9

L

LineParser class 28
log4j 29

M

Maximum number of promotions per call 9

P

Promotions
 maximum per call 9
properties-service.xml 39

R

refreshCache.do 7, 18
refreshFileCache.do 7
refreshTimeout 40
reportPromotion.do 7, 19, 20
reportPurchase.do 7

Request URL 7

S

segment variable 12
Segments
 example of fulfillment segment file for-
 mat 36
 segment file format 36
serviceName 39
serviceVersion 39

U

update.do 7

V

Variables
 examples of 30

W

WEB-INF 39

X

X-Schema Definition for Demographic Info
 on create.do 12