EasyData

Integration Guide

Updated March 16, 2013

A Product of EasyData

Table of Contents

Legal Notice

Your use of EasyData, including online and offline components, is governed by the Terms of Use as specified at http://easydata.me/terms.

Contact Support

Email: :support@easydata.me
Website: http://easydata.me/contacts
Please sign up to follow http://community.doforms.com/doforms/topics/using_web_services to be alerted to recent development to EasyData web services.

Overview

EasyData provides “smart-forms” for Android-powered smartphones and tablets - everything you need in a turn-key, all-in-one, reliable, secure, and fully hosted mobile data collection solution. Supported data types include:

- Section labels
- Page Breaks
- Textual data
- Numeric data
- Calculations
- Action buttons
- Date:Time
- Single choice answers
- Multiple choice answers
- Category scores
- Lookup tables
- Grid Tables
- Barcode scanning
- Signatures
- Sketches
- Pictures
- Video recording
- Audio notes
- GPS locations
- Approvals
- Email reports

In addition to the possible data fields above, each form is stamped with a date, time and device ID whenever the form data is saved in the mobile device.

Mobile Data Collection App

EasyData mobile data collection software works with a wide selection of popular iOS, Android, BlackBerry smartphones and tablets; and it is coming soon to Windows. Unlike all browser-based forms, our mobile data collection software, or "mobile forms app", enables your workers to operate in both connected and disconnected environments. This is critical for workers in rural areas or urban settings with cellular dead spots.

Imagine using your mobile data collection software to instantly take a picture, then sketch on top of it to illustrate something of interest; or to scan the barcodes of materials being delivered to a job site. Imagine recording audio notes and video clips and embedding them right in your electronic mobile data collection forms; or using the GPS to precisely record a location. Think about the increased speed and accuracy of reporting.

EasyData provides a flexible, fast, and easy solution to deploying mobile data collection forms to your workers - anywhere in the world. And the mobile forms on your workers' smartphone and tablet

devices are automatically synchronized and remotely kept up-to-date. Control who gets which mobile data collection forms. Remotely control who can view or change the incoming data. EasyData centralizes this control on a website dashboard to save time and money.

Data Website

EasyData provides a fully integrated website for aggregating, sorting, querying, viewing and managing mobile forms data being collected by your workers. If GPS coordinates are included, your mobile forms data can be viewed on top of an interactive map. The website keeps track of which workers collected which mobile forms data. The website makes it easy to export mobile forms data to your other business applications, and to integrate your mobile forms data in real time with other IT systems by using web services.

Use our off-the-shelf mobile data collection forms library. Or create your own mobile forms using the most powerful, yet easy-to-use form creation software available. Our form builder provides simple and intuitive user interfaces for building mobile forms. You don't need to be a specialist to use our form creation software. You just need to have an understanding of what you want your mobile data collection software to do. With EasyData, anyone with office software experience can create their own mobile data collection forms for smartphones and tablets. With EasyData, there is no need for software programmers to build your mobile forms or IT support staff to deploy them.

EasyData allows easy exporting of mobile forms information to other business applications such as Microsoft Excel, Open Office, and Google Docs. Export options are also provided for CSV, HTML and PDF file formats. Additionally, EasyData also provides for direct integration of your mobile forms with CRM, database and GIS systems, such as Salesforce.com, Oracle, SAP, and ArcGIS, through the use of industry-standard SOAP web services. These web services can be set up and deployed in a matter of minutes with no software programmers involved.

EasyData Dispatch

EasyData Dispatch provides powerful dispatch forms and work order forms functionality. These special purpose forms contain important information to tell mobile workers where to go (dispatch) and what to do when they get there (work order). As your mobile workers complete their assigned tasks, EasyData lets them fill out data fields in the form, take pictures, capture GPS locations, and collect signatures. The completed data records are then sent back to you.

In addition to sending and receiving forms from your workers. EasyData Dispatch also tracks their current and past GPS locations, and these locations are plotted on an interactive map in the Dispatch tab. You can select which workers to view, as well as the time interval. The map also shows the locations where forms were filled out.

Your EasyData website account provides a specialized Dispatch tab where dispatch forms and work order forms can be filled out, managed, scheduled and sent to your mobile workers. The data sent back from your workers is also viewed in this tab, as well as the status of their assigned job. And your workers' past and present locations are shown on a map in this tab.

Dispatch data can also be sent from your existing dispatch and work order system, and forwarded via

our Data Exchange Server to your EasyData equipped mobile devices. Similarly, the completed forms can be sent from the mobile devices, and forwarded to your existing dispatch and work order system. Our Data Exchange Server makes this integration simple and quick.

Security for Paid Accounts

The security of the EasyData system is based on (i) data transmission encryption, and (ii) Google's App Engine IT infrastructure security.

Data transmission between your mobile devices and the EasyData website is encrypted using Secure Socket Layers (SSL3). This protects your data while traveling over the airwaves or internet. Browsing of data on your EasyData website may also be encrypted using SSL/HTTPS. Please be sure to use the **encrypted SSL3 connection** at https://myeasydata.appspot.com/ **followed by the name of your EasyData** website account.

You can also have peace of mind knowing that your data and forms are hosted on top of Google's App Engine IT infrastructure. Google App Engine has successfully undergone annual SAS 70 Type II audits which have evolved into the SSAE 16 Type II attestation and its international counterpart, ISAE 3402 Type II. Google App Engine is one of the first major cloud providers to be certified for compliance to these new audit standards.

Third party audits are only part of the security and compliance benefits of Google App Engine products. Google protects our customers' data by employing some of the foremost security experts, by executing rigorous safety processes, and by implementing cutting-edge technology. These protections are highlighted in the data center video tour. For more information visit the Google Apps Trust page.

Source: http://googleenterprise.blogspot.com/2011/08/security-first-google-apps-and-google.html

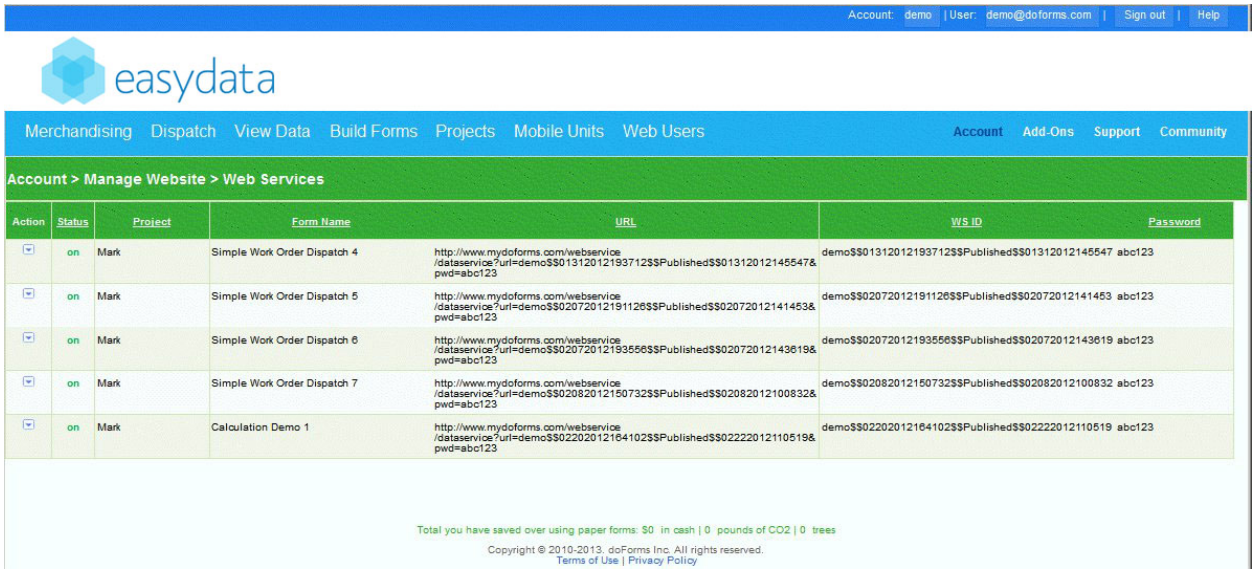
This Guide

This guide explains a number of options for integrating EasyData with a customer's existing IT systems.

The following conventions are used in this guide:

- Links and buttons appear in **bold**

Web Services API



IMPORTANT: Please sign up for http://community.doforms.com/doforms/topics/using_web_services to be alerted to recent developments in EasyData web services.

IMPORTANT: The **Web Services** tab is available to EasyData users with Manage and Admin privileges only. Web services are available for paid accounts only.

To access the EasyData web services, go to the Accounts tab and click on Web Services. Web services are used to expose the data in your EasyData account for use with other IT systems.EasyData Web Services can be accessed using any program that supports SOAP web services (you MUST use SOAP in your client programs).

EasyData web services are organized on a per-form basis. When a form is added to the Web Services tab, EasyData assigns a [WSID] and a [password] for accessing the form data. The form data served by the web services is described using Web Service Descriptive Language (WSDL).

Managing Services

Add service – To add a new web service, select **Add service** in the Row Menu. You will be prompted to first select a project, then select the form that you wish to publish. Then enter a password. The WS ID is a system-generated value. Press **Add** when done. A new row will be added to the Web Services tab containing the web service URL generated by the EasyData system, [WSID], and the[Password] you entered (the use of these parameters is explained below).

Delete service – You can delete the web service by clicking **Delete service** in the Row Menu. A message will be displayed asking you to confirm that you wish to delete the web service. Press **OK** if you do. Otherwise, press **Cancel**.

Turn on/off – Existing web services can be turned off or on by clicking **Turn on/off** in the Row Menu.

Web services which are turned off will not be accessible.

Limits on Use

We monitor and restrict how many data records are read daily by each EasyData web service WSID. All use limits are on a "per day" basis. Each "day" starts and ends at 00:00 UTC.

Limits on Data Record Reads: The limit is currently set to 50 times the number of mobile units in your account per day. So as an example, if you have 30 mobile units in your account, you can read 50X30=1500 records per day.

Limits on Lookup Table Uploads: The limit is currently set to 1,000,000 write operations per day. A good rule of thumb is the number of write operations equals approximately 4 x number of records x number of fields that are uploaded. So for 10,000 records with 10 fields you would expect approximately 400,000 write ops.

Limits on Lookup Table Downloads: The limit is currently set to 1,000,000 read operations per day. A good rule of thumb is the number of read operations equals approximately just the number of records that are downloaded (i.e., read ops are independent of the number of fields). So for 10,000 records with 10 fields you would expect approximately 10,000 write ops.

Note that during the 7 days immediately following the creation of a web service, we provide higher limits than those above in order to permit adequate testing of your software. This is done on a WSID by WSID basis. So if you need to extend this 7 days you can do so by deleting the EasyData web service, and creating a new one for the same project/form (this will generate a new WSID).

Customers requiring higher use limits should contact support@easydata.me . We reserve the right to change these limits at any time with or without notice in order to maintain the performance and reliability of your EasyData website.

NOTICE: There are currently no use-limits on the BETA lookup table related web services. These use-limits will be implemented when the production version of these web services is released and additional charges may be applicable to these web services.

Connecting to a Web Service

EasyData web services **MUST be connected to using SOAP**. There are known problems with connecting to the web services via HTTP and we do not support this method.

Detailed instructions on how to connect with a SOAP web service are beyond the scope of this manual, but well known to any IT professional experienced in SOAP. EasyData Web Services strictly follow the W3C SOAP Specification Version 1.2 (http://www.w3.org/TR/soap/). We recommend using a tool such as **soapUI** (http://www.soapui.org/) for exploring and testing a EasyData web service.

The WSDL file for a EasyData web service looks something like the following, where the [WSID] and [Password] are provided in the Web Services tab (see above). The current WSDL can be obtained at http://www.mydoforms.com/wss?wsdl. The [URL] provide a quick view of the data in its XForms XML

format and can be used to verify that the web service is being used correctly.

Basic Methods for Reading Data

Most users of the EasyData web services use them to retrieve data records submitted from mobile devices so that these data records can be integrated into another information system. In most cases this process centers around one method: "**getUnReadData**". This method is used for reading data records that have not been previously read by your client application. The "**DataFormat**" argument determines if data is returned in CSV or XML format. The "**NumberOfRec**" determines how many records are returned with each call. Null records values will be returned when there are no more data records to read. We strongly recommend making this value less than 100 for reliability.

In the simplest case, this method is used with the "isAutoUpdate" argument equal to "1". Doing so automatically removes any record that is read with the getUnReadData from the unread data queue so it will not be re-read with subsequent calls of this method. In other cases, you may want to have more control over when a record is marked as read. For example, if performing integrity checks on each returned record. In this case, use the getUnReadData method with the **"isAutoUpdate" argument equal to "0"**. Then use the **"markUnReadDataAsRead"** method to manually mark successfully read records from the unread record queue.

If you need more **control o**ver which records are read than described above, please consider using the **getRecordKeyByReceivedDateRange** and **getDataByRecordKey** methods which will enable you to identify and get specific records based on the date:time in UTC when a record was received by the myEasyData website. Alternatively, use the **getRecordKeyByDateRange** method if you want to do this based on the "Date_Created" which is the date:time that the record was created on the mobile device(not recommended).

With all of the "get" methods above, you may also want to delete data records from the EasyData website after they have been read. In this case, use the "**deleteDatabyRecordKey**" method . Note that even if the record is to be deleted, it must also be marked as "marked as read" to avoid an error if using the getUnReadData method.

IMPORTANT NOTE: In EasyData there is a distinction between a record that is newly submitted by a mobile device to the EasyData website and a previously submitted record that is edited on the EasyData website. The getUnReadData method reads both newly submitted records and edited records.

IMPORTANT NOTE: See the Limits on Use section above before implementing any of the methods below.

Downloading Media Files

The output of the web services contains the html links for each media source file. With some coding, you can create a program sending an http request via the html links to download the media source files automatically.

Sample Code

Java - http://7.mydoforms.appspot.com/support/doForms_WS_Java_Sample.zip
VB.Net - http://7.mydoforms.appspot.com/support/doForms_WS_VBNet_Sample.zip
PHP - http://7.mydoforms.appspot.com/support/doForms_WS_PHP_Sample.zip
1C - http://easydata.me/files/EasyData_WS_1C_Sample.zip

API Support Forums

http://community.doforms.com/doforms/topics/using_web_services

Important Notes

Below are the most current recommended methods provided by the EasyData web services API. Note that in all cases the "[WS ID]" and "[Password]" refer to a specific EasyData project/form web service.

The API methods described herein use two different ways of referencing specific records: Please be sure to use to the correct reference when using a specific method:

RecordId – This is the same as the "Form_Record" field displayed in the View Data and Dispatch tabs and included in exports.

RecordKey – This is an internal database reference that is more efficient than RecordID.

Also, some of the API methods described here use and/or return two different date:time stamps:

Date_Created – This is the date:time when a record is first saved as complete on a mobile device (or on the website if it was first created there.)

Date_Recieved – This is the date:time when a record was first received by the website. Note that this is the most reliable date:time for specifying "non-overlapping" record ranges.

All date/time values are returned in the UTC/GMT time zone. However you can use the Time Zone parameter that is returned by the newest version of the getDataByRecordKey method to calculate the local Date_Created.

Current API Methods

getWsIDList2: Return the list of active web service WSID's for an account which matching the password

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getWsIDList2>
         <arg0_java_lang_String>[Account Name]</arg0_java_lang_String>
         <arg1_java_lang_String>[WS Password]</arg1_java_lang_String>
         <arg2_int>[ResponseFormat]</arg2_int>
      </ser:getWsIDList2>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [Account Name] is the Account name or Website name
- [WS Password] is the WS Password
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

getFormTemplate2: get the form structure in XML format

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getFormTemplate2>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
      </ser:getFormTemplate2>
   </soapenv:Body>
</soapenv:Envelope>
```

checkValidWebservice: Check if a WS is valid or not

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:checkValidWebservice>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
</ser:checkValidWebservice>
   </soapenv:Body>
</soapenv:Envelope>
```

Return:

true: the WS is valid
false: the WS is invalid

getFormRecordCount: Get the total records for a specific form

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getFormRecordCount>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
</ser:getFormRecordCount>
   </soapenv:Body>
</soapenv:Envelope>
```

Return: number of records

getUnReadData: Just download records that have not been previously sent included the new records. Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getUnReadData>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
         <arg2_java_lang_String>[DataFormat]</arg2_java_lang_String>
         <arg3_int>[NumberOfRec]</arg3_int>
         <arg4_int>[isAutoUpdate]</arg4_int>
      </ser:getUnReadData>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [DataFormat]: 1=CSV, 2=XML
- [NumberOfRec]: the number of records to be downloaded. Due to the existing limits from GAE, we recommend to just read less than 100 records for each call.
- [isAutoUpdate]: 1-the downloaded records will be automatically set to “read” and will not be returned in next calls; 0-the downloaded records are still kept as “unread”
- Note: If using isAutoUpdate=0, you should use the markUnReadDataAsRead method to manually mark the records as read in order to avoid service limitations on number of records read daily.

markUnReadDataAsRead: Mark the unread data as “read” to remove the corresponding record from the getUnReadData call

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:markUnReadDataAsRead>
```

```
            <arg0_java_lang_String>[RecordKey]</arg0_java_lang_String>
         </ser:markUnReadDataAsRead>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [RecordKey]: The RecordKey which is extracted from the “@recordKey” field returned getUnReadData method.

getRecordKeyByReceivedDateRange: Return a list of the RecordKeys in a given received date range which is the date:time that the record is received by the website.

```
   <soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ser="services.wss.portal.doforms.mdt.com">
      <soapenv:Header/>
      <soapenv:Body>
         <ser:getRecordKeyByReceivedDateRange>
            <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
            <arg1_java_lang_String>[Password]</arg1_java_lang_String>
            <arg2_java_lang_String>[From Date]</arg2_java_lang_String>
            <arg3_java_lang_String>[To Date]</arg3_java_lang_String>
         </ser:getRecordKeyByReceivedDateRange>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [From Date]: the UTC date:time in MM/dd/yyyy HH:mm:ss format
- [To Date]: the UTC date:time in MM/dd/yyyy HH:mm:ss format

getRecordKeyByDateRange: Return a list of the RecordKeys in a given date range.

```
   <soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ser="services.wss.portal.doforms.mdt.com">
      <soapenv:Header/>
      <soapenv:Body>
         <ser:getRecordKeyByDateRange>
            <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
            <arg1_java_lang_String>[Password]</arg1_java_lang_String>
            <arg2_java_lang_String>[From Date]</arg2_java_lang_String>
            <arg3_java_lang_String>[To Date]</arg3_java_lang_String>
         </ser:getRecordKeyByDateRange>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [From Date]: the UTC date:time in MM/dd/yyyy HH:mm:ss format
- [To Date]: the UTC date:time in MM/dd/yyyy HH:mm:ss format

getDataByRecordKey4: Get the detail of a record with the specified RecordKey in a specified format. This newest version of the getDataByRecordKey method also returns the Time Zone for the record Date_Created (which is always returned in UTC/GMT). Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getDataByRecordKey>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
         <arg2_java_lang_String>[RecordKey]</arg2_java_lang_String>
         <arg3_int>[ResponseFormat]</arg3_int>
         <arg4_int>[DatetimeFormat]</arg4_int>
      </ser:getDataByRecordKey>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [RecordKey]: The RecordKey which is extracted from the “@recordKey” field returned by the “getRecordKeyByDateRange” method
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON
- [DatetimeFormat]:
- + “0” : yyyy-MM-ddTHH:mm:ss (UTC/GMT)
- + “1” : MM/dd/yyyy HH:mm:ss (UTC/GMT)

submissionDispatch2: Submit a record to given WebService.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:submissionDispatch2>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
          <arg2_java_lang_String>[recordsToSubmit]</arg2_java_lang_String>
      </ser:submissionDispatch2>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [recordsToSubmit]is a CSV data string including headers and corresponding values concatenated by @END_LINE; For example:

```
"@mobileNumber","string_question","integer_question","decimal_questi
on","date_question","Time_question","Date_Time_question","select_mul
tiple_question","select_one_question","geopoint_question","barcode_q
uestion"@END_LINE;"0909707606","sent using wss to the mobile device
0909707606 #9","11","12.5","2012-07-06","23:59:59","2012-07-
21T23:59:59","option_a option_c","option_3","10.687 106.23 54
20","898777907";
```

The “@mobileNumber” is a system field. If this field is blank or omitted, the submitted record will be set to “pending”. Otherwise,

this record will be automatically set to “sent” and delivered to the mobile unit whose mobile number is set in this field.

listDispatch: Return a list of Records headers based On Status Flag, only included RecordKey, and base header fields(MobileNumber, Date Completed, Date Submitted).

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:listDispatch>
         <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
         <arg1_java_lang_String>[Password]</arg1_java_lang_String>
         <arg2_int>[DispatchStatus]</arg2_int>
         <arg3_int>[ResponseFormat]</arg3_int>
      </ser:listDispatch>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [DispatchStatus]: 1=Pending, 2=Scheduled, 3=Sent, 4=Received, 5=Viewed, 6=Rejected, 7=Completed, 100=All
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

deleteDataByRecordKey: Delete records by the *RecordKey*

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:deleteDataByRecordKey>
         <arg0_java_lang_String>[RecordKeys]</arg0_java_lang_String>
      </ser:deleteDataByRecordKey>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [RecordKeys]: The list of the RecordKey separated by a comma (“,”). The RecordKey which is extracted from the “@recordKey” field returned by the “getRecordKeyByDateRange” method.

Return:

```
            true: the WS is valid
            false: the WS is invalid
```

deleteData: Delete records by the *record Id*

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:deleteData>
```

```
            <arg0_java_lang_String>[Record IDs]</arg0_java_lang_String>
         </ser:deleteData>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [Record IDs] The list of the record ID separated by a comma (“,”). Note that the record ID is the “Form_Record” field on the View Data tab.

Return:

0: Success
-1: System Error
-2: No record found

deleteDispatch: Deletes Dispatch records based on Ids

```
   <soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ser="services.wss.portal.doforms.mdt.com">
      <soapenv:Header/>
      <soapenv:Body>
         <ser:deleteDispatch>
            <arg0_java_lang_String>[WS ID]</arg0_java_lang_String>
            <arg1_java_lang_String>[Password]</arg1_java_lang_String>
            <arg2_java_lang_String>[RecordKeys]</arg2_java_lang_String>
            <arg3_int>[ResponseFormat]</arg3_int>
         </ser:deleteDispatch>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [RecordKeys]: the list of the RecordKey concatenated by a comma
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

getLookupTableList: Return the Lookup Table list that includes only the Lookup Table key, name, description and the column list.

```
   <soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ser="services.wss.portal.doforms.mdt.com">
      <soapenv:Header/>
      <soapenv:Body>
         <ser:getLookupTableList>
            <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
            <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
            <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
            <arg3_int>[ResponseFormat]</arg3_int>
         </ser:getLookupTableList>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData

- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

An example of a reponse in CSV format

```
[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root version="1.0">
<StatusCode>200</StatusCode>
<ErrorMessage>Success</ErrorMessage>
<Data>
<error_message/>
<error_code/>
<Result><![CDATA[key, tableName, description, columnsName
aglteWRvZm9ybXNyFQsSDkxvb2t1cFRhYmxlTXN0GIIGDA,"DS1","Sample
1","STATION,STATION_NAME,ELEVATION,LATITUDE,LONGITUDE,DATE,HLY-CLDH-
NORMAL,Completeness Flag,HLY-HTDH-NORMAL,Completeness Flag"
aglteWRvZm9ybXNyFQsSDkxvb2t1cFRhYmxlTXN0GJEGDA,"DS2","Sample
2","STATION,STATION_NAME,ELEVATION,LATITUDE,LONGITUDE,DATE,HLY-CLDH-
NORMAL,Completeness Flag,HLY-HTDH-NORMAL,Completeness Flag"
</Data>
</Root>]
```

getLookupTableData: Get the content of a Lookup Table. Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getLookupTableData>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
         <arg3_java_lang_String>[LU KEY]</arg3_java_lang_String>
         <arg4_int>[OFFSET]</arg4_int>
         <arg5_int>[LIMIT]</arg5_int>
         <arg6_int>[ResponseFormat]</arg6_int>
      </ser:getLookupTableData>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [LU KEY]: The Lookup Table key which returned by getLookupTableList
- [OFFSET]: The row position will be returned
- [LIMIT]: The number of rows will be returned
  Note: return all rows if [LIMIT] is -1.
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

An example of a reponse in CSV format

```
[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Root version="1.0">
<StatusCode>200</StatusCode>
<ErrorMessage>Success</ErrorMessage>
<Data>
<error_message/>
<error_code/>
```

```
<Result><![CDATA["STATION","STATION_NAME","ELEVATION","LATITUDE","LONGI
TUDE","DATE","HLY-CLDH-NORMAL","Completeness Flag","HLY-HTDH-
NORMAL","Completeness Flag"
"GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO
US","306.3","39.2972","-94.7306","20100101 00:00","0","C","375","C"
"GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO
US","306.3","39.2972","-94.7306","20100101 01:00","0","C","380","C"
"GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO
US","306.3","39.2972","-94.7306","20100101
02:00","0","C","384","C"]]></Result>
</Data>
</Root>]
```

addLookupTableData: Add a new Lookup Table. Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:addLookupTableData>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
         <arg3_java_lang_String>[LU NAME]</arg3_java_lang_String>
         <arg4_java_lang_String>[DESCRIPTION]</arg4_java_lang_String>
         <arg5_java_lang_String>[DATA SOURCE]</arg5_java_lang_String>
         <arg6_int>[ResponseFormat]</arg6_int>
      </ser:addLookupTableData>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [LU NAME]: The Lookup Table name
- [DESCRIPTION]: The Lookup Table description
- [DATA SOURCE]: The datasource in CSV format. The @END_LINE; is used as a carriage return for each row and the first row is the header.
- For example:
  "STATION","STATION_NAME","ELEVATION","LATITUDE","LONGITUDE","DATE","HLY-CLDH-NORMAL","Completeness Flag","HLY-HTDH-NORMAL","Completeness Flag"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 00:00","0","C","375","C"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 01:00","0","C","380","C"
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

updateLookupTableData: Update an existing Lookup Table. Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:updateLookupTableData>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
         <arg3_java_lang_String>[LU NAME]</arg3_java_lang_String>
         <arg4_java_lang_String>[DESCRIPTION]</arg4_java_lang_String>
         <arg5_java_lang_String>[LU KEY]</arg5_java_lang_String>
         <arg6_java_lang_String>[DATA SOURCE]</arg6_java_lang_String>
         <arg7_int>[ResponseFormat]</arg7_int>
      </ser:updateLookupTableData>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [LU NAME]: The Lookup Table name
- [DESCRIPTION]: The Lookup Table description
- [LU KEY]: The Lookup Table key which returned by getLookupTableList
- [DATA SOURCE]: The datasource in CSV format. The @END_LINE; is used as a carriage return for each row and the first row is the header. Note: the number of fields in the new datasource must be same with the current datasource.
  For example:
  "STATION","STATION_NAME","ELEVATION","LATITUDE","LONGITUDE","DATE","HLY-CLDH-NORMAL","Completeness Flag","HLY-HTDH-NORMAL","Completeness Flag"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 00:00","0","C","375","C"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 01:00","0","C","380","C"
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

appendLookupTableData: Append new rows to an existing Lookup Table. Note that this method is subject to the Use Limits described above.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:appendLookupTableData>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
```

```
            <arg3_java_lang_String>[LU NAME]</arg3_java_lang_String>
            <arg4_java_lang_String>[DESCRIPTION]</arg4_java_lang_String>
            <arg5_java_lang_String>[LU KEY]</arg5_java_lang_String>
            <arg6_java_lang_String>[DATA SOURCE]</arg6_java_lang_String>
            <arg7_int>[ResponseFormat]</arg7_int>
         </ser:appendLookupTableData>
      </soapenv:Body>
   </soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [LU NAME]: The Lookup Table name
- [DESCRIPTION]: The Lookup Table description
- [LU KEY]: The Lookup Table key which returned by getLookupTableList
- [DATA SOURCE]: The datasource in CSV format. The @END_LINE; is used as a carriage return for each row and the first row is the header. Note: the number of fields in the new datasource must be same with the current datasource.
  For example:
  "STATION","STATION_NAME","ELEVATION","LATITUDE","LONGITUDE","DATE","HLY-CLDH-NORMAL","Completeness Flag","HLY-HTDH-NORMAL","Completeness Flag"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 00:00","0","C","375","C"@END_LINE;
  "GHCND:USW00003947","KANSAS CITY INTERNATIONAL AIRPORT MO US","306.3","39.2972","-94.7306","20100101 01:00","0","C","380","C"
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

deleteLookupTableData: Delete all rows of an existing Lookup Table.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:deleteLookupTableData>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
         <arg3_java_lang_String>[LU KEY]</arg3_java_lang_String>
         <arg4_int>[ResponseFormat]</arg4_int>
      </ser:deleteLookupTableData>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [LU KEY]: The Lookup Table key which returned by getLookupTableList
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON

[BETA – access via http://7.myeasydata.appspot.com/wss?wsdl]

getMobileUnits: Get a list of all the mobile units connected to an account

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getMobileUnits>
         <arg0_java_lang_String>>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</</arg2_java_lang_String>
         <arg3_int>[ResponseFormat]</arg3_int>
      </ser:getMobileUnits>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData

[ResponseFormat]: 1=CSV, 2=XML, 3=JSON

[BETA – access via http://7.myeasydata.appspot.com/wss?wsdl]

getGPSTrackingPoints: Get all the tracking points that are within a certain date range (created and received on the mobile devices) for a specific mobile unit.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="services.wss.portal.doforms.mdt.com">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:getGPSTrackingPoints>
         <arg0_java_lang_String>[ACCOUNT NAME]</arg0_java_lang_String>
         <arg1_java_lang_String>[EMAIL]</arg1_java_lang_String>
         <arg2_java_lang_String>[PASSWORD]</arg2_java_lang_String>
         <arg3_java_lang_String>[MOBILE KEY]</arg3_java_lang_String>
         <arg4_java_lang_String>["FROM" DATE]</arg4_java_lang_String>
         <arg5_java_lang_String>["TO" DATE]</arg5_java_lang_String>
         <arg6_int>[ResponseFormat]</arg6_int>
      </ser:getGPSTrackingPoints>
   </soapenv:Body>
</soapenv:Envelope>
```

Where:

- [ACCOUNT NAME]: your EasyData account name
- [EMAIL]: The email address which used to login into myEasyData
- [PASSWORD]: The password which used to login into myEasyData
- [MOBILE KEY]: The mobile key which returned by the getMobileUnits method
- ["FROM" DATE]: the date:time when the GPS values are created and received on the mobile device
- ["FROM" DATE]: the date:time when the GPS values are created and received on the mobile device
- [ResponseFormat]: 1=CSV, 2=XML, 3=JSON, 4=KML

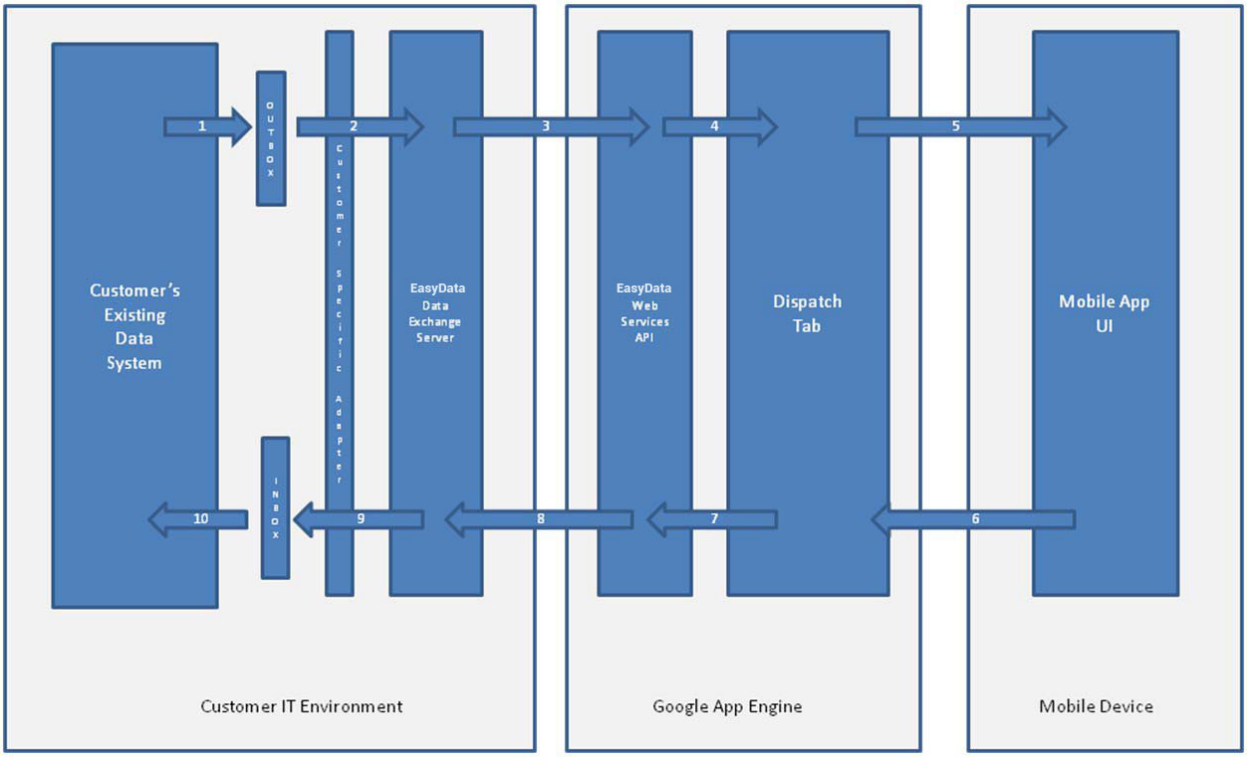
Depreciated Methods

Although deprecated methods remain in the API, their use is discouraged, and deprecation may indicate that the feature will be removed in the future. Features are deprecated—rather than immediately removed—in order to provide backward compatibility, and give programmers who have used the feature enough time to bring their code into compliance with the new standard.

The following methods are depreciated:

```
getDataByRecordKey2
getDataByRecordKey3
```

Data Exchange Server [Sold Separately]



Functional Overview

1. Data system places a "new dispatch record" into the inbox as a CSV or XML file.
2. DXS scans the OUTBOX every n seconds, and picks up any new files, and uses the "adapter" to translate the CSV or XML file into its internal data structure.
3. DXS pushes the "new dispatch record" to the web services.
4. Web services places the "new dispatch record" into the myEasyData Dispatch tab.
5. Dispatch tab automatically sends "new dispatch record" to the mobile device where the mobile user completes the form.
6. Mobile App sends the "completed dispatch record" to the Dispatch tab.
7. Dispatch tab sends a "copy" of the "completed dispatch record" to the web services.
8. DXS scans the web services for new "completed dispatch record".
9. DXS uses the "adapter" to convert the "completed dispatch record" into CSV or XML and places the file in the INBOX.
10. Data system scans the INBOX every n seconds and picks up the "completed dispatch record".

IMPORTANT: You will need to supply your own adapter that connects your existing systems, writes CSV or XML files into the OUTBOX, and reads CSV or XML files from the INBOX. Your adapter will also need to be responsible for deleting unused files and space management of the INBOX. The Data Exchange Sever will manage unused files and space management of the OUTBOX.

Prerequisites

Install Java SE

The Data Exchange Server is a Java application that requires the Java Standard Edition runtime to be installed on the same computer. To download and install Java SE:

1. Go to http://www.java.com/en/download/inc/windows_upgrade_ie.jsp
2. Follow the instructions to download and install the latest version of Java SE

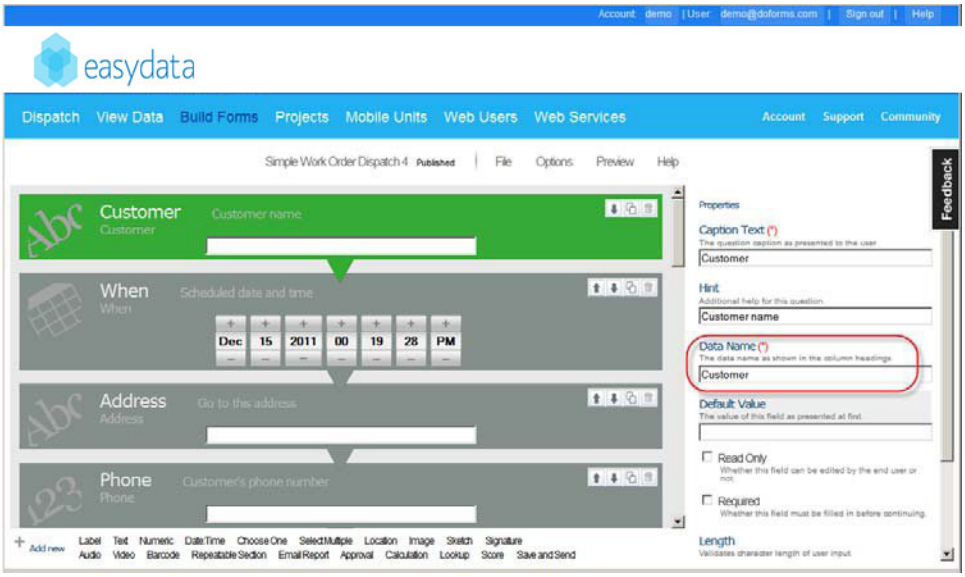
Internet Access

Your installation of Java SE and the Data Exchange Server will need to have access to the Internet in order to communicate with the EasyData system. Make sure that you do not have any Firewall programs blocking Internet access for the Data Exchange Server.

Inbox/Outbox

The Data Exchange Server will need to have read/write access to two directories – "Inbox" and "Outbox". These directories must also be accessible by whatever internal business systems you plan to connect to EasyData using the Data Exchange Server. You will need to create these directories prior to setting up the Data Exchange Server.

Step I – Build a Form



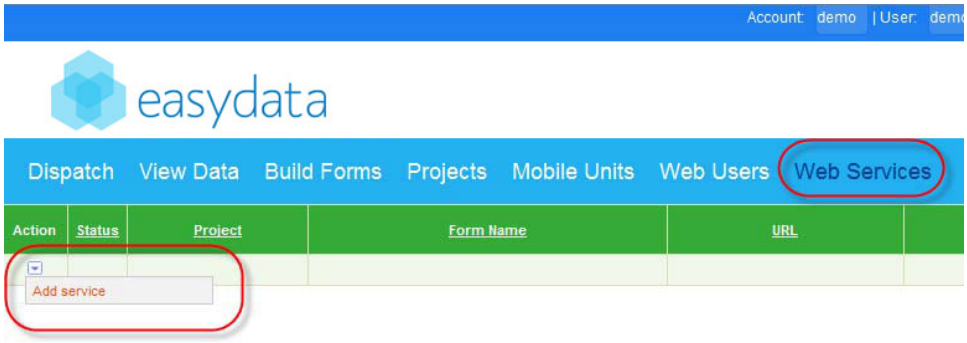
The first step in using the Data Exchange Server is having a Published form for it to connect to. The form provides all the field definitions that control how data fields from your existing system will be mapped to the EasyData website and the EasyData mobile apps (and vice versa). Some important notes:

Use the EasyData “data_name” values to map data fields.

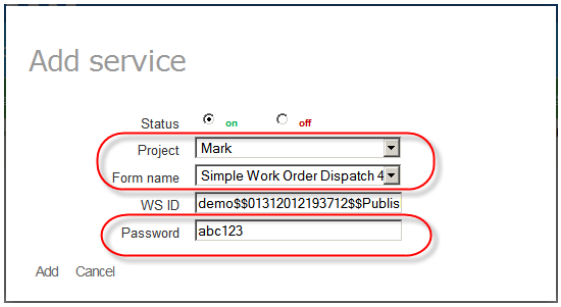
The form must be “Published” (see the EasyData Website Users Guide - Build Forms tab)

The form must be assigned to a Project (see the EasyData Website Users Guide - Projects tab)

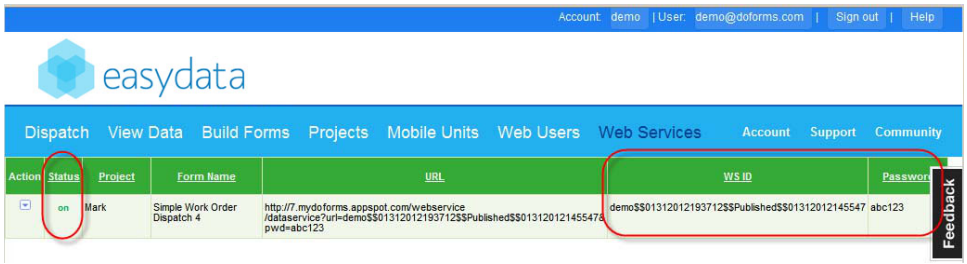
Step II - Create a Web Service for your Form [Paid Accounts Only]



1. Click on the **Web Services** tab
2. Click on the **Row Menu** (above-left)
3. Select **Add service.** An "Add service" dialog will be displayed.



4. Select a Project and Form. These will be what Data Exchange Server will be connected to.
5. Enter a Password for this web service
6. Click **Add**. The web service will be created.

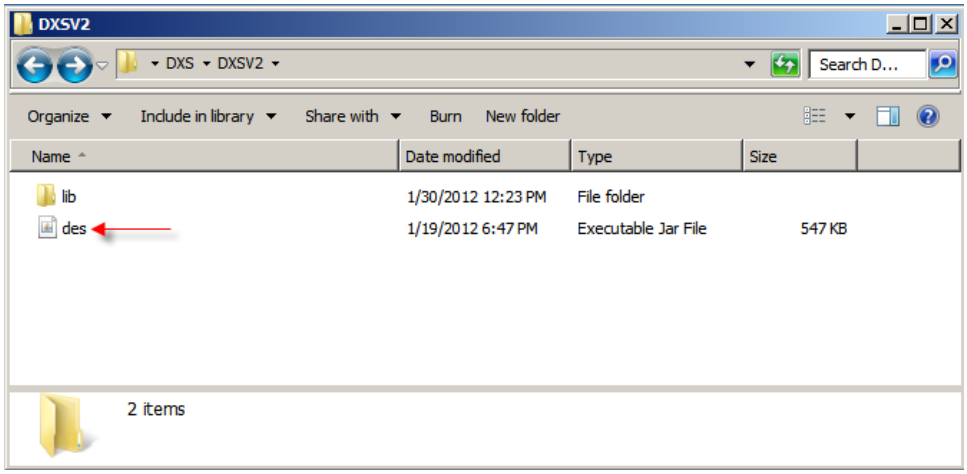


7. Make sure the Status is set to "on". If it is not, click on the Row Menu and select "Turn on".
8. Record the **Account**, **WSID** and **Password**. You will use these to connect the Data Exchange Server to this web service (and the previously specified project/form).

If you wish to connect the Data Exchange Server to more than one project/form, then repeat the steps above to create web services for each of the additional projects/forms.

For more information about using the web services - see the EasyData Website Users Guide – Web Services tab.

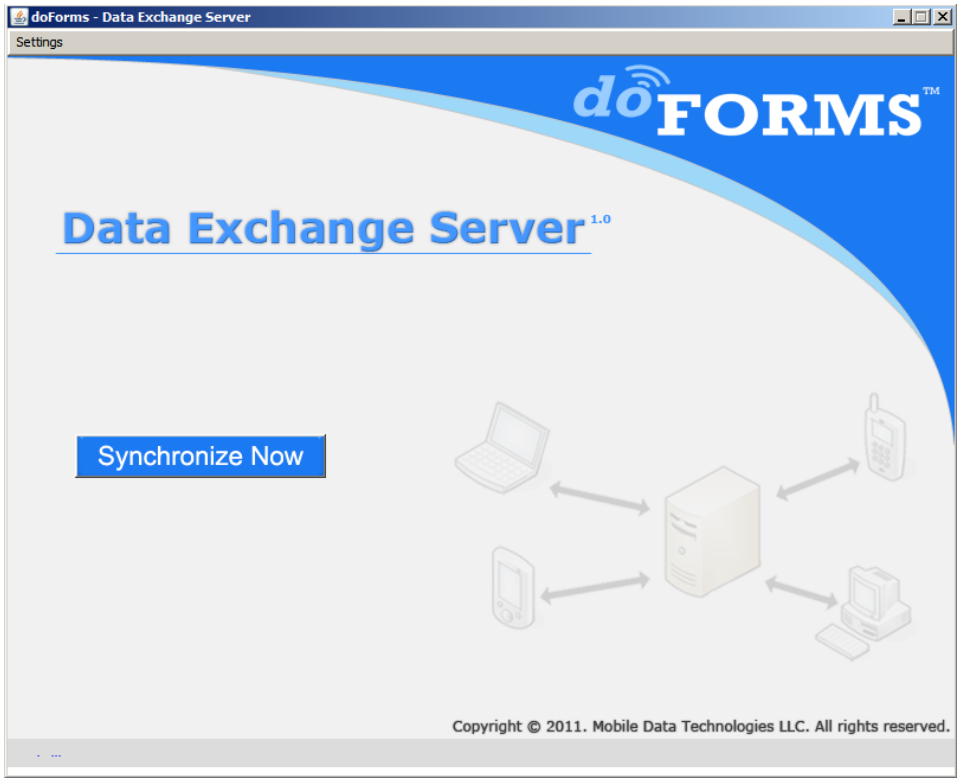
Step III – Start the Data Exchange Server



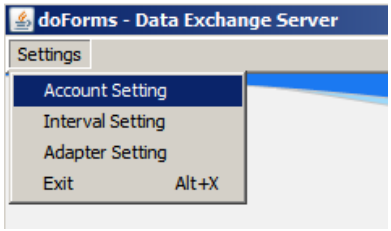
Unzip and copy the Data Exchange Server folder to the same computer where you installed Java SE.

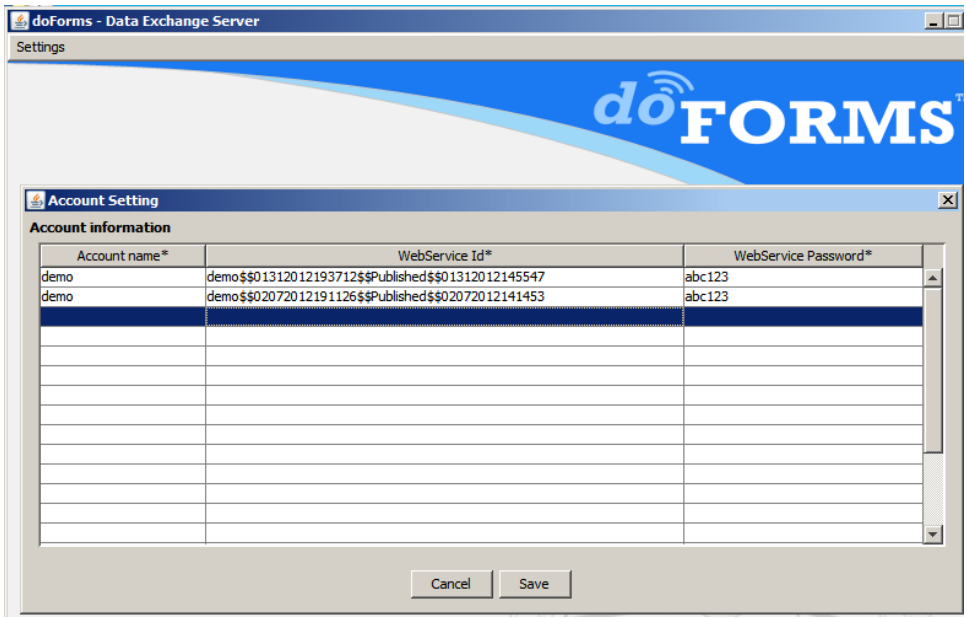
The Data Exchange Server is an executable .Jar file that does not require any special installation.

To start the Data Exchange Server, simply double click on its Jar icon.



Step IV. Configure the Account Settings



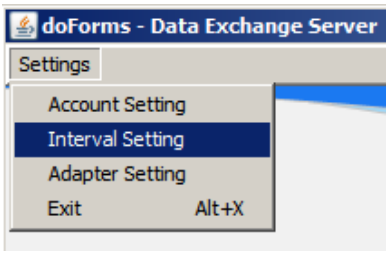


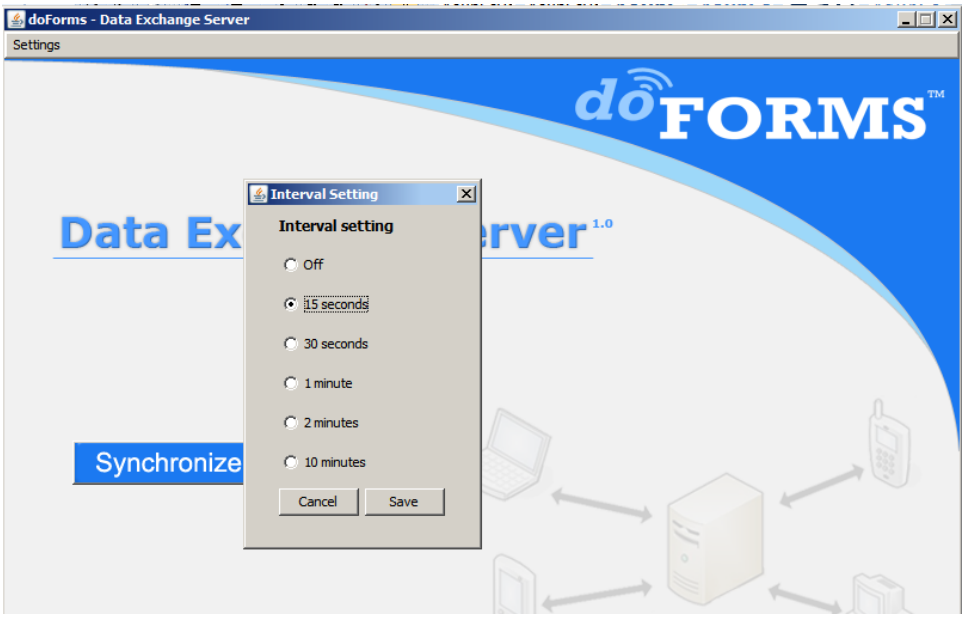
This step will "tell" the Data Exchange Server which accounts/projects/forms to connect with. See Step II above for instruction on how to create web service for a specific account/project/form

1. Select **Account Settings** in the **Setting** menu. The Account Setting page will open.
2. Enter the EasyData account name that you wish to connect to.
3. Enter the WebService ID (WSID) of the project/form that you wish to connect to.
4. Enter the password for this WebService ID.
5. Enter a mobile number.
6. Click on **Save.**
7. Repeat the steps above to connect the Data Exchange Server to additional web services.

Tip: You can double-click on a field then paste the value from the clipboard

Step V. Configure the Interval Settings

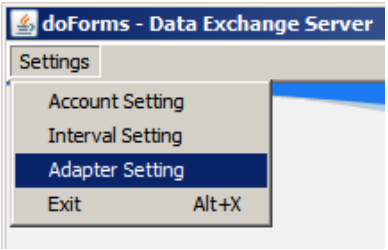


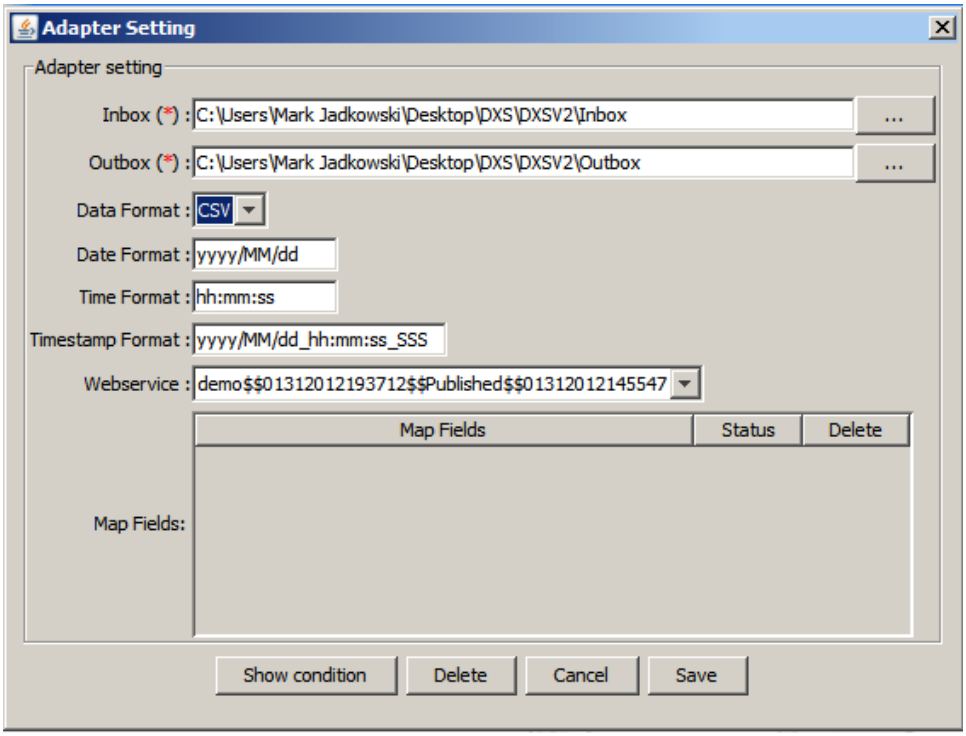


This step will "tell" the Data Exchange Server how often to synchronize with the EasyData website

1. Select **Interval Settings** in the **Setting** menu. The Account Setting page will open.
2. Select the desired interval setting.
3. Click on **Save.**

Step VI. Configure the Adapter Settings





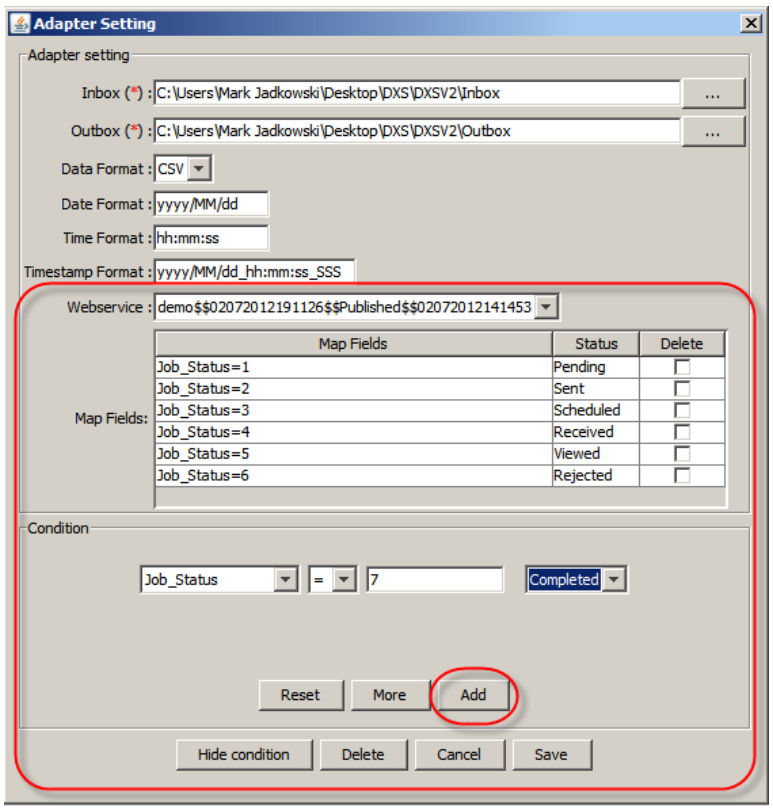
Basic Adapter Settings

1. Select **Adapter Settings** in the **Setting** menu. The Adapter Settings page will open.
2. Select the **INBOX** directory. This directory needs to be read/write accessible to both the Data Exchange Server and whatever internal business systems you plan to connect to EasyData.
3. Select the **OUTBOX** directory. This directory needs to be read/write accessible to both the Data Exchange Server and whatever internal business systems you plan to connect to EasyData.
4. Enter the **Data Format**. Options are XML or CSV (CSV files can have either .csv or .txt file extensions)
5. Enter the **Date Format** used in the CSV or XML file. The default time format used in EasyData is MM/dd/yyyy.. (note the capital M used to designate month).
6. Enter the **Time Format** used in the CSV or XML file. The default time format used in EasyData is hh/mm/ss/SSS (military time - note the capital S used to designate milliseconds).
7. Enter the **Date:Time Format** used in the CSV or XML file. The default time format used in EasyData is MM/dd/yyyy_hh/mm/ss/SSSS.
8. Click on **Save**

Advanced Adapter Settings

Use these to create a “mapping” of dispatch status system indicators for use in your client application. Available dispatch status system indicators include:

- Pending
- Sent
- Received
- Scheduled
- Viewed
- Rejected
- Completed



NOTE: The Advanced Adapter Settings are still a “work in progress” for Q2 2012.

Use Case #1 – Sending a Dispatch Record via the Outbox

To send a dispatch file to the EasyData website, from where it will be forwarded to the specified mobile device, follow these steps:

1. Create a CSV file using Excel or a text editor as illustrated below. The name of the file can be anything.
2. Place the CSV file into the Outbox directory.
3. The Data Exchange Server will read the CSV file per the interval set in the Interval Settings. The CSV file will be automatically deleted after it has been read.

Outgoing CSV in Excel

	A	B	C	D	E	F
1	@mobileNumber	@webserviceId	Customer	Address	Tasks	Satisfaction
2	2078628225	demo$$013120121937 12$$Published$$ 01312012145547	Mark Test 123	Address Test 123	Tool_1 Tool_2 Tool_3	Very_satisfied

Outgoing CSV in Text Editor

```
@mobileNumber,@webserviceId,Customer,Address,Tasks,Satisfaction
2078628225,demo$$013120121937 12$$Published$$01312012145547,Mark Test
123,Address Test 123,Tool_1 Tool_2 Tool_3,Very_satisfied
```

Outgoing CSV File Format

The first column in the file must be the "@mobileNumber" and must contain the mobile number of the mobile device to which the record is being dispatched..

The second column in the file must be the "@webserviceID" and must contain the WebService ID (WSID) of the project/form that corresponds to this record. This must be one of the WSID's specified in the Account Setting.

Additional columns can be any or all of the question fields of the project/form that corresponds to this record. Each column is labelled with the "data_name" property as defined in the Build Forms tab for that question. Data_names are strings containing letters, numbers and underscores only, with no spaces. The values below the headings must be formatted per the corresponding question type (text, numeric, date, date:time, choose-one, select multiple ….. etc.). See the additional notes below.

Notes:

a. Choose-One Questions – The answers to choose-one questions are encoded using the "underlying_value" property as defined in the Build Forms tab. (see "Satisfaction" in the example above).

b. Select-Multiple Questions – The answers to select-multiple questions are encoded using the "underlying_value" property as defined in the Build Forms tab. Underlying values are strings containing letters, numbers and underscores only, with no spaces. If more than one underlying value is used they are separated by a space (see "Tasks" in the example above).

c. Location Questions – Latitude and longitude values are in decimal degrees with four decimals placed. Western and southern hemisphere values are negative.

d. Date:Time Questions – Date and time values must be encoded as specified in the Adapter Settings.

e. Media Files – Signature, sketch, image, audio and video files are not currently supported via the Outbox. They will be supported in version 3.

Use Case #2 – Receiving a Dispatch Record via the Inbox

Completed dispatch files sent from a mobile device will be routed automatically to any Data Exchange Servers that are connected to the corresponding WebService ID. The files will be placed into the Inbox per the interval set in the Interval Settings.

It will be up to your client application to delete the incoming files after they have been read in order to manage disk space.

See the note above reading data formats.

Incoming CSV in Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Customer	When	Address	Phone	Job_Description	Time_of_Arrival	Tasks	Job_Notes	Picture	Signature	Satisfaction	GPS_Autostamp	Time_of_Departure	edit_datestring	@formKey	@recordKey	mobilekey
2	Mark Test 10	2012/02/07_04:39:03_000	Address Test 10	3.655E+09	Hftd ff f ydf	2012/02/07_04:39:28_000	Tool_1 Tool_2 Tool_3	Ff			Very_satisfied	no-gps	2012/02/07_04:39:47_432	2012-02-07T16:37:39.781	agIteWRvZm9ybXNyDQsSBEZvcm0Y3oSsAgw	agIteWRvZm9ybXNyDRRc1NCRVp2Y20wWTNvU3NBZ3cYvfiyAgw	agIteWRvZm9ybXNyEwsSCk1vYmlsZVVuaXQYlNCfAgw

Incoming CSV in Text Editor

```
Customer,When,Address,Phone,Job_Description,Time_of_Arrival,Tasks,Job_Notes,Picture,Signature,Satisfaction,GPS_Autostamp,
Time_of_Departure,score-data,edit_datestring,@formKey,@recordKey,mobilekey
Mark Test 10,2012/02/07_04:39:03_000,Address Test 10,3654789023,Hftd ff f ydf,2012/02/07_04:39:28_000,Tool_1 Tool_2
Tool_3,Ff,,,Very_satisfied,no-gps,2012/02/07_04:39:47_432,,2012-02-
07T16:37:39.781,agIteWRvZm9ybXNyDQsSBEZvcm0Y3oSsAgw,agIteWRvZm9ybXNyNAsSK2Rpc3BhdGNoYWdsdGVXUnZabTl5YlhOeURRc1NCRVp2Y20wW
TNvU3NBZ3cYvfiyAgw,agIteWRvZm9ybXNyEwsSCk1vYmlsZVVuaXQYlNCfAgw
```

Custom Development Services

Need programming services to help you with your integration?

Our software developers are experts at smartphone application development and related backends. If you have used EasyData, you know how good their applications are.. Now this expertise is at your disposal for developing customized mobile business applications for your organization's specific needs. Our software development services start at $65 per hour.

Contact us for a free consultation.