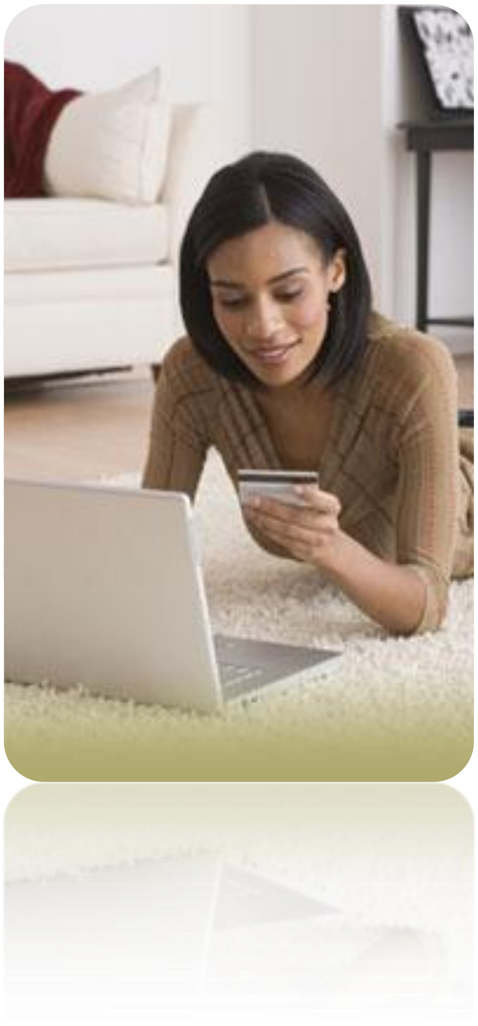
4-TELL BOOST™ RECOMMENDATIONS

Technical Integration Guide

VERSION 2



4TELL™

PREDICT MORE SALES

TABLE OF CONTENTS

OVERVIEW

Personalized recommendations have been shown to increase online sales by at least 10%. 4-Tell's solution provides effective recommendations for all products, whether they are popular, new to the catalog, or have few sales. Our system is sophisticated, providing the best shopping experience, and enabling multichannel recommendations, including websites, email, social media and mobile commerce.

4-Tell's solution is extremely easy to set up. The process to add recommendations to a website should take less than 20 hours for an existing website and even less to include them in a new site design or update.

This document covers the 4-Tell Boost hosted solution. A local version of the 4-Tell Solution is available if you prefer to install and run it on your own server. For more information on the local solution, please contact us.

RECOMMENDATION TYPES

Each shopping scenario requires different recommendation types. 4-Tell's sophisticated recommendations are categorized into six main types:

Cross-Sell: These recommendations highlight relevant complementary products. They are based on the aggregate sales history of the item currently under consideration by the shopper, whether a known or anonymous visitor. For example, recommendations for a shirt might include pants, jacket and a hat.

Personalized: These recommendations show items that the shopper will likely buy. They are driven by the shopper's individual buying habits and not by any specific product. For this reason, they are best used when the consumer first arrives at your site (i.e., a customized landing page) or in non-product specific email (i.e., monthly newsletter). They are one-to-one, meaning they are unique for each shopper.

Personalized Cross-Sell: As the name suggests, these recommendations are a combination of the first two types. The results are related to the shopping experience and influenced by the shopper. The focus of these recommendations is still primarily on the current shopping experience, so all of the results will be cross-sells as described above. The difference comes by automatically filtering these results to elevate items that are more relevant to the current shopper based on their past buying habits. Because these recommendations are related to the product(s) and personalized for the shopper, they are very powerful!

Blended: Just like personalized cross-sell, these recommendations are a combination of both cross-sell and personalized recommendations, so they are available when there are one or more products and a known shopper. The difference is that these recommendations give equal weight to all of the shopper's purchases and are not as focused on items from the current session. The system will display the most relevant recommendations based on our sophisticated algorithm, taking all known information into account. These recommendations are best used when a shopper has completed a shopping experience, such as order confirmation or user admin pages (if there are items in the cart).

Similar: These recommendations show comparable (i.e. alternative) products based on common attributes and allow you to present more of your product catalog to the shopper. This is helpful for browsers who have landed on a product page, but find it's not really what they were looking for. To go back to the shirt example, similar recommendations would include other shirts.

Top Sellers: These are overall top sellers, or top sellers in a specific category. They are usually provided by the ecommerce platform, but can also be generated by the recommendation solution.

In summary, the recommendations are created using the following parameters in the calls to our service.

Terminology	**Get Recommendations Call**		
Recommendation Type	**Product ID(s)**	**Customer ID**	**Recommendation Type**
Cross-Sell	Yes	No	1 = Cross-Sell
Personalized Cross-Sell	Yes	Yes*	1 = Cross-Sell
Personalized	Optional**	Yes	3 = Personalized
Blended	Optional	Optional	0 = Blended
Similar	Yes	Optional	4 = Similar
Top Sellers	Optional	Optional	5 = Top Sellers

* Automatically personalizes if include customer ID

** If product ID(s) are included, the personalized recommendations will use that data

INTEGRATION SUMMARY

The web integration includes two main stages:

1. **Generate** Recommendations (database export)
    a. Create configuration file
    b. Export data
    c. Upload files
    d. Generate recommendation tables (e.g. nightly at midnight)

2. **Display** Recommendations (website edits)
    a. Add a call to our service to get recommendations
    b. Display thumbnail images, information and links

GENERATING RECOMMENDATIONS

Generating recommendations includes four smaller steps:

a. Create configuration file
b. Export historical sales data, product catalog and configuration file
c. Upload files
d. Generate recommendations (e.g. nightly at midnight)

BOOST CONFIGURATION FILE

The Boost configuration file (ConfigBoost.txt) includes the parameters used to create the recommendation tables in a tab delimited file. This configuration file lives on the 4-Tell server alongside your uploaded sales data. A default configuration file is created for you when your client Alias is activated, but you can upload a new configuration file at any time.

The configuration parameters include:

- **Version:** The version number of this file (currently 2).
- **Owner:** Name of contact point (POC) for errors and issues.
- **Email:** Email address for the POC.
- **Report Level:** Types of messages (errors, warnings, etc.) that should be sent to the POC. The options (in decreasing order) are None, Error, Warning, Information, All. The default value is None. If a different level is chosen, all messages generated for your Client Alias at that level or higher will be emailed to the POC.
- **Currency:** The currency symbol to be displayed with product prices (e.g. $, €, ¥). This is only necessary for client-side JavaScript implementations.
- **Attribute1Name:** The plain text name to use for the title of the first product attribute exported (e.g. Category). This is used in the Merchandizing Dashboard as the name of the tab for that attribute's data display.
- **Attribute2Name:** The plain text name to use for the title of the second product attribute exported (e.g. Brand). This is used in the Merchandizing Dashboard as the name of the tab for that attribute's data display.
- **Resell**: Whether previously purchased products should be offered to the same customer.
- **MinLikelihood:** The minimum likelihood of purchase for a cross-sell or up-sell product to be included in the recommendation tables. In other words, products with a likelihood percentage below the minimum are not recommended, usually 5%
- **MinCommon:** The minimum number of common users for a cross-sell or up-sell product to be included in the recommendation tables. In other words, products with common users below the minimum are not included in the tables. Usually this is set to 2, such that product pairs bought by only 1 customer are not recommended.
- **ResultFormat:** (Optional) The format of text results from the "GetRecIDs/string" call in the Boost service. Available formats are SpaceDelimited, CommaDelimited, TabDelimited, and XML. Default value is

TabDelimited. This parameter is optional because you can also pass a format value in your call to the service, and that value would override this setting.

- **DoNotRecommendExists:** Whether a file has been uploaded that lists out of stock and discontinued items, or products that you just don't want to recommended. This is the DoNotRecommend.txt file described below (see page 9).
- **ReplacementExists:** Whether a file has been uploaded that lists items that have been replaced in the product catalog. This allows new replacement items to take advantage of sales history that was gathered from the items that were replaced (see page 9 for details).
- **MaxSalesDataAgeInMonths:** The maximum age of data that is allowed to be used for recommendations. The value is an integer representing months prior to the current date. Any historical sales data older than this age will be discarded.
- **NumSalesFiles**: The number of files that are included in the Sales data list.
- **<Sales Data Filename(s)>**: The names of the sales data file(s), one per line. This could be a single file such as Sales.txt, or a list such as Sales01.txt to Sales24.txt.

For the case where there is one sales file that is overwritten each time, an example is listed below:

```
Version                   2
Owner                     Ken Levy
Email                     ken@4-tell.com
ReportLevel               Error
Currency                  $
Attribute1Name            Category
Attribute2Name            Brand
Resell                    0
MinLikelihood             5
MinCommon                 2
DoNotRecommendExists      1
ReplacementsExists        1
MaxSalesDataAgeInMonths   18
NumSalesFiles             1
Sales.txt
```

DATA EXPORT

Your exported sales data and catalog is used to generate the Boost recommendation tables. Filenames must match the list below. We suggest you export 1 year of sales data if you have seasonal products, and 2 years of sales data for non-seasonal products.

PLEASE NOTE:

1. *The IDs can be numeric or alpha-numeric*
2. *All dates can either be in mm-dd-yyyy or yyyy-mm-dd format*
3. *Product IDs should be a family-level ID rather than a SKU (i.e. do not include color or size codes)*

We require that you export the following two files:

1. Sales Data (Sales.txt or the filename(s) chosen in the config file above)
2. Product Catalog (ProductDetails.txt)

We recommend that you also export the following four files (if applicable):

3. Products to Never Recommend (DoNotRecommend.txt)
4. New Products that Replaced Old Products (Replacements.txt)
5. Attribute 1 Names (Attribute1Names.txt)
6. Attribute 2 Names (Attribute2Names.txt)

SALES DATA (REQUIRED)

Filename = Sales.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (for *date of export*)> <cr-lf>
Product ID <tab> **Customer ID** <tab> **Quantity** <tab> **Date** <cr-lf>
Data
Product ID <tab> Customer ID <tab> Quantity <tab> Date <cr-lf>
...

Each sales record consists of a Product ID and a Customer ID, and optionally includes a quantity, and a sales date, with one line for each sale. Repeat sales (i.e. same Product ID and Customer ID) are important and should be included as additional lines or in the quantity field because they are used to improve recommendations. If no quantity is included, it is assumed to be 1.

PRODUCT CATALOG (REQUIRED)

Filename = ProductDetails.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (for *date of export*)> <cr-lf>
Product ID <tab> **Name** <tab>**Att1 ID** <tab> **Att2 ID** <tab> **Price** <tab> **Filter** <tab> **Link** <tab> **Image Link** <tab> **Standard Code**<cr-lf>
Data
Product ID <tab> Name <tab>Att1 ID <tab> Att2 ID <tab> Price <tab> Filter <tab> Link <tab> Image Link <tab> Standard Code<cr-lf>
...

The Product Details file must include at least one line for each product. Each line includes the fields listed above, although some are optional. The table below shows which entries are required, depending on how you plan to integrate with our service. We offer both client-side(JavaScript) and server-side web service integration options (see Displaying Recommendations below for more details).

Field	Data Type	Server-Side	Client-Side (JavaScript)
Product ID	alpha-numeric	●	●
Name	Text	●	●
Att1 ID	alpha-numeric	◉	◉
Att2 ID	alpha-numeric	◉	◉
Price	Numeric	◉	◉
Filter	alpha-numeric	○	○
Link	URI	○	●
Image Link	URI	○	●
Standard Code	alpha-numeric	○	○

●	Required
◉	Recommended
○	Optional

Table 1: Product Catalog Fields

The data must be entered in the same order as shown above. If a field is skipped, you must still provide the correct number of tabs. (e.g. no second attribute, then ProductID <tab> Name <tab> Att1 ID <tab> <tab> Price <tab>…). It is best to be consistent for all lines in the file and either include a field in every entry or else exclude it for all entries. If no fields are included for the rest of the line, the carriage return and line feed can come early.

It is always best to include all of the fields. Some fields, like Attribute2, we use today, and others, like filter attribute, we will use in the future to make the optimal recommendations. At a minimum, the file must include each Product ID in the sales data along with a corresponding product name.

Note: any Product ID not included in the product catalog but included in sales data will not be recommended. For products that have been discontinued, it is best to either; (i) include the Product ID in the catalog and in the Do Not Recommend file, such that our solution can learn from the sales of the product to improve recommendations – or (ii) include the Product ID in the Replacements file such that a new Product ID is recommended in place of the old Product ID. For the latter option, it is still best to include the Product ID in the catalog so that the Merchandiser Dashboard can display the information, but it is does not have any effect on the recommendations.

Product ID: The Product ID is the unique product identifier, preferably at the family product level (e.g. not including size and color), but it can be at the SKU level. We don't suggest including SKU details, such as size and color, since the recommendation takes the shopper to a page where they can choose these options; however, if you have a business where this detail is important (e.g. you have common color palettes for your jewelry) you may want to include color. You CANNOT include commas within your Product ID.

Name: The Name is the display name of the product, and can include any standard characters.

Attribute1 ID: The Attribute1 ID is for an attribute or characteristic of the product, such as category or collection. It is multidimensional, meaning that you can have a product belong to several attribute 1 elements.

When a product belongs to multiple attribute elements, the product ID is either repeated with all of the other same information, or the attribute1 IDs are separated by commas (,). The most important category is preferably listed first, as this will be the default category used in the dashboard's display. However, this order does not affect the recommendations.

The attribute should be a deep attribute. For example, a category should not be high level like clothing, but deeper such as running shirts, dress shirts, t-shirts, and so on.

You CANNOT include commas within your attribute IDs. The attribute ID can be a number or alpha-numeric ID with the name linked via the attribute1 names file, or the attribute ID can be the name of the attribute (please be careful to spell the name the same each time so we don't think there are two different attribute elements).

Attribute2 ID: The Attribute2 ID is one to one for each product. It is most likely brand, but it may be collection, feature, department, or any one-to-one attribute. The attribute ID can be a number or name, as described for attribute 1.

Price: The Price is the product price as a decimal value without the currency symbol. For client-side integration, this field enables the price to be shown in the recommendation. It will also be used in the future for both: (i) business rules (e.g. don't recommend anything over 25% more expensive than the product) and (ii) automatic filtering based upon each customer's previous price sensitivity.

Filter ID: Filter ID is used to create a general filter attribute (besides price) for the product. For example, for clothing, the filter ID is usually gender, and we suggest using five genders: man, woman, girl, boy, unisex. There can be any number of values for this filter. This field will be used in the future to automatically filter based on customer buying habits.

Link: This is the link to the product page for the product that you want to display with the recommendation for our client-side JavaScript solution (not necessary for server-side web service use).

Image Link: This is the link to the image of the product that you want to display with the recommendation for our client-side JavaScript solution (not necessary for server-side web service use). It is usually the thumbnail image that is used with category or search pages.

Standard Code: This is the standard product code for the product. It can be the UPC, ISBN, or any other standard code. Even if your product ID is the same code, please include the code in this field again. This standard code for each product will enable improvements of recommendations in the future. Please note that we only use your sales data for your own recommendations, and we will NOT change this model without your approval.

DO NOT RECOMMEND (OPTIONAL)

Filename = DoNotRecommend.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (*date of export*)> <cr-lf>
Product ID <cr-lf>
Data
Product ID <cr-lf>
...

The optional Do Not Recommend file consists of a list of out-of-stock and discontinued Product IDs, or any product that you do not want to recommend, one on each line. These IDs are never recommended, and can be updated at any time. Rather than omitting these items completely from your data export, it is best to include the product ID here, and also include it in the sales data and product catalog. This way our system can learn more about your customers' buying habits.

REPLACEMENTS (OPTIONAL)

Filename = Replacements.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (*date of export*)> <cr-lf>
Old Product ID <tab> **New Product ID** <cr-lf>
Data
Old Product ID <tab> New Product ID <cr-lf>
...

The optional Replacements file consists of a list of any out-of-date products and the new replacement product. This enables a new product to be recommended and have great recommendations immediately. Once a product is listed in this file, the old product will no longer be recommended on your site and the new product will benefit from all of the historical sales data. Initially, the new product replaces the old product, such that it uses the same recommendations and appears as a recommended product in the same places. Then, its new sales cause the recommendations to become unique to the new product. The Merchandiser Dashboard will show sales for the old products in the stats for the new product.

ATTRIBUTE1 NAMES (OPTIONAL)

Filename = Attribute1Names.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (*date of export*)> <cr-lf>
AttID <tab> **Name** <cr-lf>
Data
AttID <tab> Name <cr-lf>
...

The Attribute1 Names file contains one line for each attribute ID followed by its display name. This field usually corresponds to category names. The names are used only for display purposes in the Merchandiser Dashboard.

If you use categories or something similar, we suggest you use the lowest level category. For example, if there's a hierarchical system (men's clothing -> pants -> ski pants), use the ski pants category.

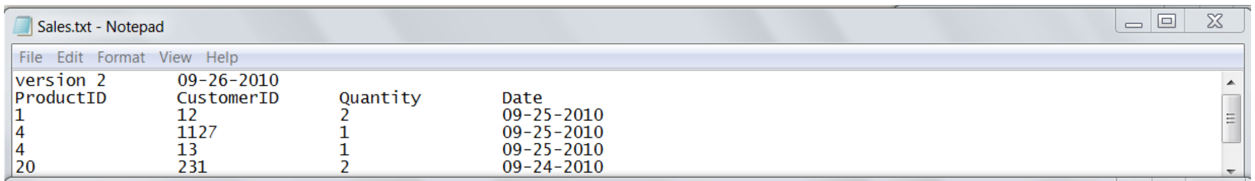
ATTRIBUTE2 NAMES (OPTIONAL)

Filename = Attribute2Names.txt
Header
Version <tab> **2** <tab> <mm-dd-yyyy (*date of export*)> <cr-lf>
AttID <tab> **Name** <cr-lf>
Data
AttID <tab> Name <cr-lf>
...

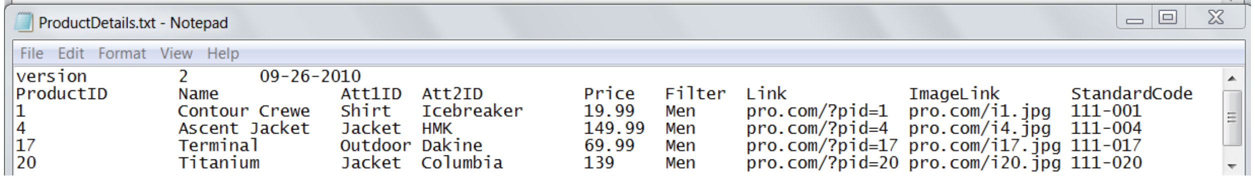
The Attribute 2 Names file contains one line for each attribute ID followed by its display name. This field usually corresponds to brand names. The names are used only for display purposes in the Merchandiser Dashboard.

EXPORT EXAMPLES

Example with category and brand names in the catalog, and without a do-not-recommend file (i.e. retailer handles out-of-stock filtering on their website).

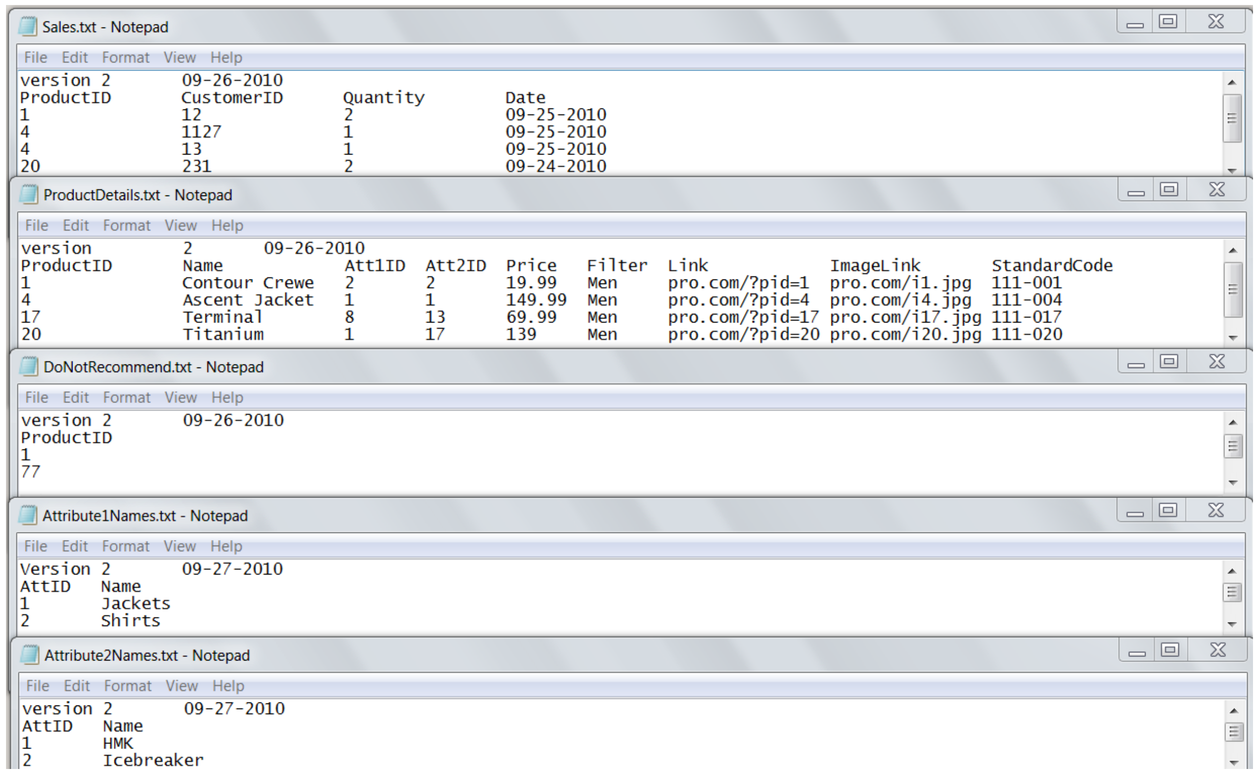


```
version 2       09-26-2010
ProductID       CustomerID      Quantity        Date
1               12              2               09-25-2010
4               1127            1               09-25-2010
4               13              1               09-25-2010
20              231             2               09-24-2010
```



```
version         2       09-26-2010
ProductID       Name            Att1ID  Att2ID          Price   Filter  Link            ImageLink       StandardCode
1               Contour Crewe   Shirt   Icebreaker      19.99   Men     pro.com/?pid=1  pro.com/i1.jpg  111-001
4               Ascent Jacket   Jacket  HMK             149.99  Men     pro.com/?pid=4  pro.com/i4.jpg  111-004
17              Terminal        Outdoor Dakine          69.99   Men     pro.com/?pid=17 pro.com/i17.jpg 111-017
20              Titanium        Jacket  Columbia        139     Men     pro.com/?pid=20 pro.com/i20.jpg 111-020
```

Example with products to not recommend and normalized attributes, but no replacement products.


Sales.txt - Notepad
File Edit Format View Help

```
version 2       09-26-2010
ProductID       CustomerID      Quantity        Date
1               12              2               09-25-2010
4               1127            1               09-25-2010
4               13              1               09-25-2010
20              231             2               09-24-2010
```

ProductDetails.txt - Notepad
File Edit Format View Help

```
version         2       09-26-2010
ProductID       Name            Att1ID  Att2ID  Price   Filter  Link            ImageLink       StandardCode
1               Contour Crewe   2       2       19.99   Men     pro.com/?pid=1  pro.com/i1.jpg  111-001
4               Ascent Jacket   1       1       149.99  Men     pro.com/?pid=4  pro.com/i4.jpg  111-004
17              Terminal        8       13      69.99   Men     pro.com/?pid=17 pro.com/i17.jpg 111-017
20              Titanium        1       17      139     Men     pro.com/?pid=20 pro.com/i20.jpg 111-020
```

```
version 2       09-26-2010
ProductID
1
77
```

```
Version 2       09-27-2010
AttID   Name
1       Jackets
2       Shirts
```

```
version 2       09-27-2010
AttID   Name
1       HMK
2       Icebreaker
```

UPLOAD DATA

You only need to upload new or changed files. This means that the first time you upload, you should send the sales data, product catalog and any other optional data files, along with the configuration file (if you changed the default). Then for future uploads, you usually only need to upload the new sales file. If data errors are found when generating recommendations, you will receive an email with specific problems explained, and our web service will continue to use the last good set of data that was uploaded.

To upload your data, use our web service and call one of our UploadData functions discussed below. Examples showing how to connect to the web service are provided in the Appendix (starting on page 21).

UPLOAD DATA — STREAM

address: www.4-tell.net/Boost2.0/UploadData/stream
verb: POST
query parameters: none
body: data

example: UploadData/stream

NOTE: No parameters cannot be passed to this function, so you must pre-pend them to the beginning of the stream in the format of:
clientAlias<tab>filename<tab>beginGenerator<cr-lf>
For example: MyAlias<tab>Sales.txt<tab>true<cr-lf>
The last parameter tells whether the generator should be run after this file is uploaded (code examples are included in the appendix starting on page 21).

This operation allows you to upload one file at a time, passing the file as a raw stream, and specifying the filename to use for saving the file on the server. The filename should match one of those specified in the Data Export Section on page 5. New files uploaded will overwrite existing files, and there is no need to upload files that have not changed since the last upload. Uploads are currently limited to a maximum file size of 10 MB per file. Contact 4-Tell if this limit needs to be increased, but keep in mind that sales data can be split into a number of files (e.g. one per month or one per quarter).

There are two ways to complete the data update. For the simpler method, set the parameter "beginGenerator" to true on the last file that is uploaded and everything will be handled automatically. As an alternative, you can leave this flag false, and then call the other two operations discussed in the Appendix to generate the data tables and reload them into the web service (see page 20). This latter method will provide you with more control over the timing of these operations, and it will let you troubleshoot each operation individually; however, it will become your responsibility to ensure that each operation is completed successfully before the next operation is called. This testing is handled automatically if the simpler method is used.

The operation returns a text string stating either "Upload complete" or else an error message with a description of the problem. In some cases, the error message may refer you to the server error log for details. In that case, contact 4-Tell to retrieve more error details. Depending on your ErrorLevel setting in ConfigBoost, you may also receive these details by email. If "beginGenerator" was true and the upload was successful, then the return string will also contain additional lines listing the success or failure of the additional operations.

UPLOAD DATA — BYTE ARRAY

address: www.4-tell.net/Boost2.0/UploadData/byteArray
verb: POST
query parameters:
clientAlias
fileName
beginGenerator

body: data (as a byte array)

example: UploadData/byteArray?clientAlias=MyStore&filename=Sales.txt&beginGenerator=false

This operation allows you to upload one file at a time, passing the file as a byte array and specifying the filename to use for saving the file on the server. Other than the method of passing the parameters and data, this function behaves identically to the streaming version described above. Please refer to that description for further details. Parameters described below can be passed in any order.

Parameters Descriptions:

- **clientAlias** – An eight or less character ID (e.g. MyStore), usually the first eight characters if your company name. This ID is provided by 4-Tell during activation.
- **fileName** – Name used to save the uploaded file on the server. This filename must match one of the names listed in the export directions above.
- **beginGenerator** – Flag (true or false) to kick-off the Boost Generator after the current file is uploaded

GENERATE AND RELOAD RECOMMENDATIONS

Whenever data is uploaded, it is necessary to generate new recommendation tables and then reload the new tables into the service. If any errors occur during this process, the service will continue running with the last successfully loaded data. As described above, a simple method is provided during the upload process to ensure that the recommendation tables are automatically generated and reloaded. If you want to specifically create recommendation tables and reload the tables into the service, please see the details of the GenerateTables call in Appendix B: Web Service Details.

DISPLAYING RECOMMENDATIONS

This second stage involves changing the desired webpage templates to display recommendations in a recommendation area. The recommended product IDs are returned from 4-Tell's web service and used to display the thumbnail image, description, price, and/or link to recommendation's product detail page.

> *NOTE: this integration does NOT require modification to your shopping cart, content management system (CMS), or any other third-party web component*

There are two steps to display the recommendations:

a. Add a call to our service in your website page templates
b. Receive recommended product IDs and display recommendations on the web page

Our service provides a REST API that can be called from any programming language. We have customers accessing our service now from a number of popular programing languages including .NET, PHP, Ruby, Pearl, and JavaScript.

There are two main options to access our service:

a. Server-side using your language of choice to call our web service
b. Client-side using JavaScript to call our web service

For most sites, the server-side solution is preferred since it will load the information before the page content is sent to the client browser. This makes the page-load happen faster and provides better content for SEO (search engine optimization). In cases where the server code cannot be altered (e.g. some closed shopping cart systems), then the client-side solution provides a simple method to add the same functionality. Examples of server-side and client-side calls to the Boost service can be found in the Appendix (starting on page 21).

WEB SERVICE REST API

The main functions of 4-Tell Boost are:

- GetRecIds to retrieve recommendations as a list of product IDs.
- GetRecDisplayList to retrieve more detailed records for each recommendation including the name, price, link, and image link. (This function is more valuable for client-side applications where this additional data is not readily available).

These calls are provided to support both server-side and client-side integrations. Server-side users will prefer to use GetRecIds, while client-side (JavaScript) users will need to use GetRecDisplayList. Details are provided below. The Boost API also includes the Upload Data functions discussed above, and three additional calls for advanced usage: ServiceTest, GenerateDataTables, and ReloadDataTables. These calls are discussed in the Appendix (see page 20).

GET RECOMMENDATIONS IDS

address: www.4-tell.net/Boost2.0/GetRecIDs/array --or-- www.4-tell.net/Boost2.0/GetRecIDs/string
verb: GET
query parameters:
clientAlias
productIDs
cartIDs
blockIDs
customerID
numResults
startPosition
resultType
fillTopSell
format
callback
body: none

Example:
GetRecIds/array?clientAlias=MyStore&productIDs=ID1,ID2&customerID=joe_cool26&numResults=5
&startPosition=1&resultType=0&format=json

This is the main operation for the 4-Tell Boost Web Engine. It is used in your server-side code to retrieve recommendations from our service for display your websites and emails. Detailed parameter descriptions are listed below. Not all parameters are required, and parameters can be listed in any order. As listed above, there are two options for the call depending on whether an array or string response is desired. Please see the description of the **format** parameter below for more details.

Parameters Descriptions:

- **clientAlias** – (REQUIRED) An eight or less character ID (e.g. MyStore), usually the first eight characters of your company name. This ID is provided by 4-Tell during activation.
- **productIDs** – (RECOMMENDED) The numeric or alpha-numeric product ID, or a list of product IDs separated by commas. The IDs supplied could include any products relevant to the current customer's visit to your site. For example: the current product being viewed on a product detail page, previous items viewed in this user's session, items already in the user's shopping cart, products displayed in a search result, or the featured item on a campaign landing page. An empty string can be used if no product IDs are available, for instance on a landing page or home page. These items are automatically added to the blockIDs (below).
- **cartIDs** – (OPTIONAL) A list of items currently in the shopping cart. This should be used when you don't want products in the cart to be included in the recommendations, but you still want them to help influence the recommendations. A prime example is this is the product detail page. On other pages, such as the cart or admin pages, all products in the cart should be listed in the productIDs parameter above, such that the recommendations can be related to any product in the cart and influenced by all products in the cart.
- **blockIDs** – (OPTIONAL) This is a list of items that you would NOT like to have recommended in the current call to our service. This is in addition to the DoNotRecommend.txt file that you may have uploaded to our service (see page 9). Items in this list only affect the current call to our service. Note that the productIDs and cartIDs are automatically added to this list, so you do not need to list them again here.

- **customerID** – (RECOMMENDED) The numeric or alpha-numeric ID for the current customer. This could be obtained when the customer logged into the website, or from a cookie. An empty string can be used if the customer is browsing anonymously.
- **numResults** – (RECOMMENDED) The number of recommended product IDs to return, with a default of 1. The web Engine will always supply this number of results, choosing the best recommendations available from the information provided and guided by the result type requested.
- **startPos** – (RECOMMENDED) The starting position of the recommended product IDs, with a default of 1. This can be used so different recommendations are provided each time the page is viewed.
- **resultType** – (RECOMMENDED) The list below defines the different types of recommendations that can be returned by the Web Engine. Detailed descriptions are provided in the Recommendation Types section on page 2 above.

  0. **Cross-Sell**: Products likely to be purchased with the provided product ID(s) since they have been bought with the provided product(s) before. If multiple product IDs are provided, recommendations that repeat with several products are ranked higher. *Note: Providing a customer ID with this type automatically changes the results to Personalized Cross-Sell.*
  1. **Personalized**: Products that the current customer (logged in or identified via cookie) will likely purchase since they are related to (e.g. cross-sell items for) the products that the customer has purchased in the past.
  2. **Blended** (Default): The best combined recommendation for the provided product ID(s) and/or customer ID. If both customer ID and product ID(s) are provided, the recommendations include a combination of cross sell and up sell items. Otherwise, it will just use the type below that is best suited for the data available.
  3. **Similar**: Products that are comparable with the provided product ID(s) based on their attribute(s).
  4. **Top Sellers**: The top sellers for the historical sales data.

- **fillTopSell** – (OPTIONAL) For all result types discussed above, if not enough of the requested type of items are available, the next best recommendations will be used to fill in the blanks. This assures that the number of requested recommendations is always returned. If this behavior is not desired, you can set fillTopSell to false so that top sellers will never be used to fill the recommendation list. Keep in mind that this means you will have to test the returned list to make sure it is not null and to see how many results have been found.
- **format** – (OPTIONAL) The format of the results that are returned. The options for this parameter depend on whether you select GetRecIds/array or GetRecIds/string when you call the service. The array version of the call can return either JSON (JavaScript object notation) or XML. The string version of the call can return TabDelimited CommaDelimited, or SpaceDelimited. If no format is specified, the respective default values are JSON and TabDelimited.
- **callback** – (OPTIONAL) A callback function to pass the JSON array to. This parameter is ignored unless format=json is also provided. These two parameters together create a JSONP response, required for client-side JavaScript integrations.

Response: 4-Tell Web Engine returns data that you can easily use in your website or emails without imposing any design or layout requirements. Web developers already know how to work with product IDs to pull images and descriptions from your database, so we just provide the IDs of recommended products. The reply from this call is an array or string of recommended product IDs in the format specified by the **format** parameter above. To retrieve all of the product details, see the GetRecDisplayList call described below.

An example comma delimited return when 5 recommendations are requested is:

    1024, 1072, 1349, 1063, 1049

The same response formatted as a JSON array is:

    [”1024”,”1072”,”1349”,”1063”,”1049”]

Errors: In the case of an error, the service will attempt to interpret the command that was sent and reply with the number of recommendations requested (with a minimum of one). An error message will then be logged on the server and emailed to the address in the configuration file. In the case of a critical error that prevents a valid response, such as the wrong Client Alias, the service will return a Web Exception with a standard web status code (i.e. 400, 404, etc.) and followed by a detailed description of the error in the response message.

GET RECOMMENDATION DISPLAY ITEMS

address: www.4-tell.net/Boost2.0/GetRecDisplayList
verb: GET
query parameters:
- clientAlias
- productIDs
- cartIDs
- blockIDs
- customerID
- numResults
- startPosition
- resultType
- fillTopSell
- format
- callback

body: none

Example:
GetRecDisplayList?clientAlias=MyStore&productIDs=ID1,ID2&customerID=joe_cool26&numResults=5
&startPosition=1&resultType=0&format=json

This is an alternate method for retrieving recommendations from the 4-Tell Boost service. Instead of only returning a list of product IDs, this method returns a detailed set of information for each item. The extra data provided is intended to include all the information necessary to display the recommendations to the shopper. This allows quick display without the need for further database lookup. It also makes it possible to call our service and display recommendations from client-side JavaScript. For scenarios that do not need all of the result fields, you are not required to export all of the fields. Our service will respond with the complete array, passing back as much or as little information as is available from your product catalog export.

Parameters: This function uses all of the same parameters as listed above for GetRecIds. Please refer to the list above for details

SUMMARY

It should take under 20 hours to integrate 4-Tell Boost into the website and schedule the automatic export of sales data to the 4-Tell server. Once these steps are completed, future exports and display of recommended items occur automatically.

An automated script sends sales data every night to the 4-Tell Boost server so that updated recommendations are generated. Then, when the consumer opens a web page, the web page calls the Boost webservice to Get Recommendations, with different options requested depending on the information available for the user and page being viewed. The web service uses the result format requested, and returns the requested number of recommended product IDs. The web page then displays the thumbnail image, product link, and possibly the price and short description from the product database for each recommended product ID.

CONTACT INFORMATION

For Sales Questions: 4-Tell Sales sales@4-tell.com (503) 746-9070	For Technical Questions: 4-Tell Support support@4-Tell.com (503) 746-9070

APPENDICES

APPENDIX A: MONTHLY AND DAILY EXPORTS

In order to keep the 4-Tell recommendations up to date, it is important that you setup a script to automatically export your sales data on a regular schedule; for example, you could export every night at 2 am, or once a week on Sunday nights. After the data is exported, it must be uploaded to our server so that the Generator can analyze the data and update lookup tables for the web service. Each time that the Generator is run, it recreates all of the recommendation tables using only the information supplied for that run.

If your sales data is large (e.g. over 10MB), you do not need to export all of the data every night. Obviously, uploading a year's worth of data every night will include a lot of data that is already on the server. Instead, you may choose to only export and upload the current quarter, month, or week of sales data. For example, if exporting each month, the name of the file would be Sales<Month>.txt. This enables the other eleven months of data to remain on the server from previous uploads. A partial configuration file for this example would be:

```
Version             2
...
NumSalesFiles       12
SalesJan.txt
SalesFeb.txt
...
SalesNov.txt
SalesDec.txt
```

As an alternative, you could export only the new data each day, and then send any other changed files as needed. The preferred method of accomplishing this is to export each day's sales into a new file, and name it using a numbered file such as Sales<num>.txt, where the <num> is replaced by a three digit number (e.g., 001 for the first file). When <num> reaches the maximum number of days of sales, <num> is started over. For example, if sales are exported nightly and 1 year of data is used, 365 is the maximum <num>. This format enables the Boost configuration file (see Boost Configuration File on page 4) to list the expected sales files without the need for the file names to be changed.

A partial configuration file for this example would be:

```
Version             2
...
NumSalesFiles       365
Sales001.txt
Sales002.txt
...
Sales 364.txt
Sales0365.txt
```

APPENDIX B: WEB SERVICE DETAILS

CONNECTING TO THE WEB SERVICE

The address to our service is: www.4-tell.net/Boost2.0/rest/

The method for connecting to the service and calling exposed functions will vary depending on the language used for your website. We are working on examples in various languages and will make these available as they are completed. Please refer to the examples below, or contact our technical support for more details (support@4-tell.com). The Boost web service conforms to standard REST protocols and is therefore compatible with any programming or scripting language. Google the term "REST Client" for your programming language to find examples. In most languages, you will be able to easily connect to our service and call the service operations.

SERVICE TEST

address: www.4-tell.net/Boost2.0/ServiceTest
verb: GET
query parameters: None
body: None

example: ServiceTest

This operation requires no parameters and is intended mainly to verify proper connection to the service during initial setup. It returns a text string stating either "4-Tell Web Engine startup successful" or else an error message followed by details of the error.

GENERATE DATA TABLES

address: www.4-tell.net/Boost2.0/GenerateDataTables
verb: POST
query parameters:
clientAlias
reloadTables

body: None

example: GenerateDataTables?clientAlias=MyStore&reloadTables=false

This operation tells the service to call the Generator tool on the server to generate new recommendation lookup tables. The tables will be generated from the most recent files that were uploaded. This call is not necessary if UploadData was called with the "beginGenerator" flag set to true (see Upload Data on page 11). The "reloadTables" parameter can be used to automatically reload data tables after a successful run of the Generator. If this flag is false, then the tables will not be loaded until the next time the service is recycled.

The operation returns a text string stating either "Generator complete" or else an error message with a description of the problem. In some cases, the error message may refer you to the server error log for details.

In that case, contact 4-Tell to retrieve more information. If "reloadTables" was true and the Generator was successful, then the return string will also contain additional lines listing the success or failure of reloading the tables into the service.

APPENDIX C: C#.NET EXAMPLE

CONNECTING TO THE BOOST SERVICE

The easiest way to connect to the Boost service using .Net languages is to use the .Net `Uri` class to create the service address with all desired parameters. Then use `HttpWebRequest` and `HttpWebResponse` classes to interact with the service and get results. An example class is provided below that you can insert into your project. Then just call `BoostClient`.GetResults() in each page of your site where you want to add recommendations.

GET RECOMMENDATIONS

```
//This is an example class that you can insert into your project
//Then just call BoostClient.GetResults() on each page where you want recommendations
using System;
using System.IO;
using System.Net;
using System.Collections.Generic;

class BoostClient
{
  public enum pageType
  {
      //add or replace page types here as needed
      //then update switch statement below with your settings
      home,
      searchResults,
      category,
      productDetails,
      addToCart,
      cartView,
      checkout,
      admin,
      afterPurchaseEmail,
      abandonedCartEmail,
      newsletter
  }

  //configuration values
  private string BoostService = "http://www.4-tell.net/Boost2.0/rest/";
  private string ClientAlias = "<yourClientAlias>";


  public string GetResults(pageType page, string productIDs = "",
            string cartIDs = "", string blockIDs = "", string customerID = "")
  {
      string function = "GetRecIds/array";
      string resultType;
string numResults;

//Best Practices: Select parameters based on the page being displayed
//Update these to customize settings for your site
switch (page)
{
    case pageType.home:
        resultType = "4";  //Top Sellers
        numResults = "3";
        break;
    case pageType.searchResults:
    case pageType.category:
        resultType = "3";  //Similar
        numResults = "5";
        break;
    case pageType.productDetails:
    case pageType.addToCart:
    case pageType.cartView:
        resultType = "0";  //Cross-sell (and personalized Cross-sell)
        numResults = "5";
        break;
    case pageType.checkout:
        resultType = "2";  //Blended
        numResults = "5";
        break;
    default:
    case pageType.admin:
        resultType = "2";  //Blended
        numResults = "3";
        break;
    case pageType.afterPurchaseEmail:
    case pageType.abandonedCartEmail:
        resultType = "0";  //Cross-sell
        numResults = "4";
        break;
    case pageType.newsletter:
        resultType = "1";  //Personalized
        numResults = "4";
        break;
}

//setup the
string queryData = "?clientAlias=" + ClientAlias
                 + "&productIDs=" + productIDs
                 + "&cartIDs=" + cartIDs
                 + "&blockIDs=" + blockIDs
                 + "&customerID=" + customerID
                 + "&numResults=" + numResults
                 + "&startPosition=1"
                 + "&resultType=" + resultType;

HttpWebRequest request = null;
HttpWebResponse response = null;
string result = null;
try
{
    Uri serviceURI = new Uri(BoostService + function + queryData);
```

```
            request = WebRequest.Create(serviceURI) as HttpWebRequest;
            request.Method = "GET";
            request.ContentType = "text/plan";
            request.ContentLength = 0;

            using (response = request.GetResponse() as HttpWebResponse)
            {
                // Get the response stream
                StreamReader reader = new StreamReader(response.GetResponseStream());
                result = reader.ReadToEnd();
            }
        }
        catch (WebException wex)
        {
            result = "Status = " + wex.Status.ToString();
            result += "\nWebException = " + wex.Message;
            // Get the response stream
            HttpWebResponse wexResponse = (HttpWebResponse)wex.Response;
            if (wexResponse != null)
            {
                StreamReader reader = new StreamReader(wexResponse.GetResponseStream());
                result += "\n" + reader.ReadToEnd();
            }
        }
        return result;
    }
```

UPLOAD DATA FILES

```
//Add this function to the class above to loop through and upload all the files that exist in your data folder
//You could also read the data directly from your database and upload it without saving intermediate files
        public string UploadFiles(string filePath)
        {
            //This is an example way to loop through and upload all the files that exist

            string[] fileList = {"ConfigBoost.txt", "ProductDetails.txt",
                                 "Attribute1Names.txt","Attribute2Names.txt",
                                 "DoNotRecommend.txt", "ProductsSoldOut.txt", "Sales.txt"};
            string resultText = "";

            for (int i = 0; i < fileList.Length; i++)
            {
                bool lastFile = (i == fileList.Length - 1);
                try
                {
                    // get the exact file name from the path
                    string fileName = filePath + fileList[i];
                    if (!File.Exists(fileName))
                    {
                        resultText += "File: " + fileList[i] + "----Not found\n";
                        continue;
                    }

                    // get the file information for the selected file
                    FileInfo fInfo = new FileInfo(fileName);
```

```
long numBytes = fInfo.Length;
double dLen = Convert.ToDouble(fInfo.Length) / 1000000.0;

// current limit of 10 MB on web server
//--contact 4-Tell if your files too large
if (dLen > 10)
    resultText += "File: " + fileList[i] + "----Size is over limit\n";

else
{
    // set up a file stream and binary reader for the selected file
    FileStream fStream = new FileStream(fileName, FileMode.Open,
                                               FileAccess.Read);
    BinaryReader br = new BinaryReader(fStream);

    // pre-pend header line with parameters
    List<byte> data = new List<byte>();
    char[] tempParam = ClientAlias.ToCharArray(); //clientAlias
    foreach (char c in tempParam)
        data.Add((byte)c);
    data.Add((byte)'\t'); //tab between parameters
    tempParam = fileList[i].ToCharArray(); //fileName
    foreach (char c in tempParam)
        data.Add((byte)c);
    data.Add((byte)'\t'); //tab between parameters
    data.Add((byte)(lastFile ? '1' : '0')); //beginGenerator
    data.Add((byte)'\n'); //new line at end

    // convert the file to a byte array
    byte[] buffer = br.ReadBytes((int)numBytes);
    br.Close();
    fStream.Close();
    fStream.Dispose();
    foreach (byte b in buffer)
        data.Add(b);
    byte[] bData = data.ToArray();

    // pass byte array to the web service
    string function = "UploadData/stream";
    HttpWebRequest request = null;
    HttpWebResponse response = null;
    string result = null;
    try
    {
        Uri serviceURI = new Uri(BoostService + function);
        request = WebRequest.Create(serviceURI) as HttpWebRequest;
        request.Method = "POST";
        request.ContentType = "text/plan";
        request.ContentLength = 0;
        request.ContentLength = bData.Length;

        // Write data
        using (Stream postStream = request.GetRequestStream())
        {
            postStream.Write(bData, 0, bData.Length);
        }
```

```
                            using (response = request.GetResponse() as HttpWebResponse)
                            {
                                // Get the response stream
                                StreamReader reader = new StreamReader(
                                                          response.GetResponseStream());
                                result = reader.ReadToEnd();
                            }
                        }
                        catch (WebException wex)
                        {
                            result = "Status = " + wex.Status.ToString();
                            result += "\nWebException = " + wex.Message;
                            // Get the response stream
                            HttpWebResponse wexResponse = (HttpWebResponse)wex.Response;
                            if (wexResponse != null)
                            {
                                StreamReader reader = new StreamReader(
                                                          wexResponse.GetResponseStream());
                                result += "\n" + reader.ReadToEnd();
                            }
                        }
                        resultText += "File: " + fileList[i] + "----" + result + "\n";
                    }
                }
                catch (Exception ex)
                {
                    resultText += "File: " + fileList[i] + "----" + ex.Message + "\n";
                }
            }
            return resultText;
        }
```

APPENDIX D: PHP EXAMPLE

CONNECTING TO THE BOOST SERVICE

Coming soon…

GET RECOMMENDATIONS

Coming soon…

UPLOAD DATA FILES

Coming soon…

APPENDIX E: JAVASCRIPT EXAMPLE

CONNECTING TO THE BOOST SERVICE

Our JavaScript solution allows you to add the service call and recommendation display in your client page code. This enables users of proprietary shopping cart platforms to use our service without making any changes to the code that runs on the server. To add this to your site, simply call our REST API using a JSONP response format as demonstrated below.

GET RECOMMENDATIONS

The following four files demonstrate how to retrieve and display recommendations using client-side code in your site. Both vertical and horizontal examples are provided.

4TELLBOOST.JS

This is the main JavaScript code. It is listed here as a separate file for clarity, but could also be embedded directly in the html page <head> section by placing it between the <script> and </script> tags.

```
var RecCaption = 'Customers also bought...'; //global variable to set display caption

/*
 * getRecommendations:
 * This function sets the number and type of recommendations based on the type of page
 * where they will be displayed. You can use this function to set global policies or
 * else you can call get4TellResults directly from each page (see below).
 *
 */

function getRecommendations(pageType, productIDs, cartIDs, blockIDs, customerID) {
    var resultType = 0;
    var numResults = 4;
    var startPos = 1;

    //These are example settings for different pages of your site.
    //Feel free to add or remove page types here to match your site architecture.
    switch (pageType) {
        case 'ProductDetail': //product detail page (PDP)
            resultType = 0; //Cross-sell
            startPos = 1;
            RecCaption = 'Customers also bought...';
            break;
        case 'AddToCart': //intermediate add to cart page
            resultType = 0; //Cross-sell
            startPos = 1;
            RecCaption = 'Customers also bought...';
            break;
        case 'OrderShipping': //delivery options page
            resultType = 0; //Cross-sell
            startPos = 1 + numResults; //second block
            RecCaption = 'You may also like...';
            break;
        case 'OrderPayment': //payment options page
            resultType = 2; //Blended
            startPos = 1;
            RecCaption = 'You may also like...';
            break;
        case 'OrderConfirm': //order confirmation page
            resultType = 2; //Blended
            startPos = 1 + numResults; //second block
            RecCaption = 'You may also like...';
            break;
        case 'OrderComplete': //checkout complete page
            resultType = 2; //Blended
            startPos = 1 + (2 * numResults); //third block
            RecCaption = 'Suggestions for you...';
            break;
        default: //any page not listed
            resultType = 2; //Blended
            startPos = 1;
            RecCaption = 'Suggestions for you...';
            break;
    }
    get4TellResults(resultType, numResults, startPos,
                    productIDs, cartIDs, blockIDs, customerID);
}

/*
 * get4TellResults:
 * This function sets the parameters and calls the 4-Tell Boost web service
 * to retrieve recommended product data. This function can be called directly
 * or you can use GetRecommendations listed above to call by page type instead.
 *
 * You must adjust the clientAlias to match the ID assigned by 4-Tell
 * when you activate your account.
 *
 */
function get4TellResults(resultType, numResults, startPos,
              productIDs, cartIDs, blockIDs, customerID) {

    var webService = 'www.4-Tell.biz/Boost2.0/rest/';
    var operation = 'GetRecDisplayList';
    var clientAlias = 'MyStore';
    var startPosition= 1;

    // Assemble the url to call
    var jsonUrl   =   webService + operation
                 + '?clientAlias=' + clientAlias
                 + '&productIDs=' + productIDs
                 + '&cartIDs=' + cartIDs
                 + '&blockIDs=' + blockIDs
                 + '&customerID=' + customerID
                 + '&numResults=' + numResults
                 + '&startPosition=' + startPosition
                 + '&resultType=' + resultType
                 + '&format=json'
                 + '&callback=displayRecs';
    (function() {
        var forTell = document.createElement('script'); forTell.type =
                          'text/javascript'; forTell.async = true;
        forTell.src = ('https:' == document.location.protocol ? 'https://' :
                          'http://') + jsonUrl;
        var s = document.getElementsByTagName('script')[0];
        s.parentNode.insertBefore(forTell, s);
    })();
}


/*
 * displayRecs:
 * This function parses the JSON array returned by the web service and builds the
 * display opjects. It is called automatically by the callback function that wraps
 * the response from the web service.
 *
 */
function displayRecs(data) {
    // If the data is passed in then proceed
    if (data) {

        var items = data['GetRecDisplayListResult'];
        var prods = document.getElementById('products');

        //set the caption
        var caption = document.createElement('div');
        caption.setAttribute('class', 'caption');
        caption.innerHTML = RecCaption;
        prods.appendChild(caption);

        //setup each recommendation in the list
        for (var p=0; p<items.length; p++) {
            // This is the main container for one product
            var prod = document.createElement('div');
            prod.setAttribute('class', 'product');
            // This is the link to the product
            var link = document.createElement('a');
            link.setAttribute('href', items[p].pageLink);

            // This is a image and gets put into the link
            var img = document.createElement('img');
            img.setAttribute('src', items[p].imageLink);
            img.setAttribute('class', 'productImage');
            link.appendChild(img);

            // This is the product title and gets put into the link
            var prodTitle = document.createElement('a');
            prodTitle.setAttribute('class', 'productTitle');;
            prodTitle.setAttribute('href', items[p].pageLink);
            prodTitle.innerHTML = items[p].title;

            // This is the product price
            var prodPrice = document.createElement('div');
            prodPrice.setAttribute('class', 'productPrice');
            prodPrice.innerHTML = items[p].price;
```

```
                // Put the elements into the main product container
            prod.appendChild(link);
            prod.appendChild(prodTitle);
            prod.appendChild(prodPrice);

            // Put the main product container into the products container
            prods.appendChild(prod);
        }
    }
}
```

SAMPLEPAGE.HTML

Now, calling this from any page on the site is as simple as making a call to either GetRecommendations or to Get4TellResults. For example, here is a simple example page that calls the sevice when it loads.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <link rel="stylesheet" type="text/css" href="horizontal.css">
    <script src="4TellBoost.js" type="text/javascript">
        //NOTE: contents of the script file listed above can be pasted here
        //         if external files are not desirable.
    </script>
</head>

<!--
    *   The onload shows how to get recommendations by page type.
    *   See the 4TellBoost.js file for details
    *  The recommendations are then displayed in the <div …> tag
    -->
<body onload="getRecommendations('OrderComplete', 'ID1,ID2',
                         'ID3,ID4,ID5', ' ' , 'customerID1')">
    <h1>4-Tell JavaScript Horizontal Display Example</h1>
    <div id="products"></div>
</body>
</html>
```

HORIZONTAL.CSS

The only difference between horizontal and vertical display is the style sheet that is used. Here is an example of a horizontal style sheet.

```
.product {
    float: left;
    clear: none;
    padding: 10px;
    margin: 0 10px 0 0;
    border: none;
}
.product .productImage {
    border: 0px;
    float: left;
    clear: both;
}
.product .productTitle {
    color: #0000ff;
    float: left;
    clear: both;
}
.product .productPrice {
    color: #ff0000;
    float: left;
    clear: both;
}
```

VERTICAL.CSS

Here is an example of a vertical style sheet. To test this, modify the first line under the <head> tag in the sample html page above so that it points to this file instead of to the horizontal.css file.

```
.product {
    float: left;
    clear: both;
    padding: 10px;
    margin: 0 0 10px 0;
    border: none;
}
.product .productImage {
    border: 0px;
    float: left;
    clear: both;
}
.product .productTitle {
    color: #0000ff;
    float: left;
    clear: both;
}
.product .productPrice {
    color: #ff0000;
    float: left;
    clear: both;
}
```