mdTransit Reporting Administrator's Guide

Version 1.0

PUBLISHED BY

Morris Digital Works
A Division of Morris Communications LLC

P.O. Box 936
Augusta, GA 30903
Fax: 706-828-4339
Phone: 800-622- 6358
www.morrisdigitalworks.com

Revision History
March 2005

Contact Information
For questions or comments regarding this publication contact the Morris Digital Works Documentation Department at mdw.doc@morris.com.

Table of Contents

Introduction

MdTransit's robust web-based reporting system allows site administrators to compile valuable traffic and usage statistics about their Autos site. This document explains the backend implementation of mdTransit Reporting, how to collect data from the autos site, as well as explanations of the associated crumb logs and data files. It is intended for internal MDW use in setting up and managing the system resources necessary for mdTransit Reporting, and for implementing tracking crumbs to collect autos data for reports.

Providing Feedback

To maintain the quality of our publications and software, we welcome your comments on the accuracy, clarity, and value of this publication and all other product documentation. Send comments to the Documentation Department at: mdw.doc@morris.com, or directly to any of the documentation specialists.

Reporting Overview

Reporting data is compiled by tracking the number of image requests made to the reporting server. The image requests are referred to as tracking crumbs and track site activity when they are logged to the Access log and then parsed by the CrumbLog application. After parsing, the crumb data is inserted into the database.

What is a Tracking Crumb?

A crumb is a call for an image (1x1 pixel gif) located within the html code for the web page. Crumb calls are in the following format:
http://reporting_server/images/mdwtc-$PLACEMENT_CODE-$CRUMB_TYPE-$CRUMB_ID.gif.

Note: Crumbs may pass additional information through a query string.

The table below describes each step in the data accumulation process.

Step	Action
1	The web browser requests the autos page from a property's website.
2	The autos page contains tracking crumbs.
3	The web browser requests crumb images from the Apache reporting server
4	The Image requests are logged to the Access Log. Apache uses mod_rewrite to convert the image request into a request for a clear 1x1 pixel gif image and logs the original request in the following format: IP DATE HOUR_OF_DAY DAY_OF_WEEK REQUEST REFERER QUERY_STRING *Example:* 216.116.242.240 16-Dec-2004 08 4 /images/mdwtc-109-MDTRANSIT_DEALER_DETAILS-1917852.gif http://forge.mdtransit.morris.com/autos?property=pbp&tp=pbp&temp_type=detail&tl=25&classification=Autos&ads_per_page=20&ad_type=dealer+new&feature_make=Chevrolet&zi

p_search=33401&ZIP_CODE_RADIUS_SEARCH=25 ?r=1103203723

5 The Access Log is parsed by the crumb log application to extract the following values:

"CRUMB_ID" "CRUMB_TYPE" "KEY" "VALUE" "PARENT" "ACTION" "1967946" "MDTRANSIT_AD" "MAKE" "Jeep" "DATE(13-Dec-2004).PLACEMENT_CODE(100)" "INCREMENT"

6 The crumb log loads the parsed crumb objects into the database

Objects are loaded into MDW_UTIL.LOG_OBJECT using the CRUMB_ID (stored as the object_id and a translated OBJECT_TYPE based on the CRUMB_TYPE (translations are stored in a conf file). The KEY and VALUE are stored in the LOG_OBJECT_ATTRIBUTE table, using the PARENT value to determine the value of the parent_id and the action to determine whether the value should be stored, whether it should be used to lookup another value for storage based on the object type (for retrieving information in the db without having to pass it on the query string) and whether a child hit count attribute record should be created or incremented.

Nightly Cron Process

The nightly operation of mdTransit reporting consists of a scheduled cron process as described below.

Step	Action
1	The scheduled Cron runs on lily at 2:03 am. Files are located in /var/opt/crumb-logs/
2	ftp.pl picks up the previous day's access log from rpt.mdtransit.morris.com after it is rotated but before it is processed and archived. This file is placed in the data/director
3	process.sh runs the command "CrumbLog -P" which picks up the file ftp.pl placed ir the data/directory, processes it and archives it to the archive/directory. The processed file is in the parsed/directory.
4	loader.sh runs the command "CrumbLog -L" which picks up the file from the parsed directory and loads the contents into the database.

Disk Space Allocation

If disk space becomes an issue on lily, the files in the archive directory can be deleted. The other large consumer of space is the atl.log file in the log directory which can be gzip'd (or blanked if you are sure no errors have occurred).

Data Collection Process

Report data is compiled from image requests(tracking crumbs) to the reporting server. These tracking crumbs are located on each page in the autos site. Below is a description of how data is collected and processed. Each step represents a potential point of failure for the mdTransit reporting system. Items 1-6 represent unrecoverable failure points.

Steps 1-6 are unrecoverable failure points

Step	Action
1	User requests mdTransit template.
2	Template contains tracking crumbs.
3	The user's browser requests crumb images from the reporting server.
4	The crumb request is rewritten to a clear gif by a Rewrite rule in the apache configuration.
5	The request for the crumb is logged in the reporting server's access_log in the CrumbLog format.
6	The Access log is archived after analog processing.
7	The archived data file is moved to processing area.
8	The archived data file is parsed and converted to a loadable format.
9	The loadable format data file is sorted to help efficiency.
10	The loadable format data file is loaded into the database.

Tracking Crumbs

This section explains how to implement tracking crumbs to collect autos data for reports, including the correct format and placement of crumbs.

Format

All tracking crumbs are in the following format:
/image/mdwtc-{$PLACEMENT_CODE}-OBJECT_TYPE-${OBJECT_ID}?otherdatayouwanttotrack=counts

Placement

The following explains where each type of reporting crumb is located within the mdTransit site. Refer to the table at the end of this section for the value to insert for the PLACEMENT CODE.

Report Name	Crumb Implementation
Site Traffic	This crumb appears on every site page. It contains an OBJECT_TYPE of MDTRANSIT_SITE and OBJECT_ID of the template property ID (in Template Manager). If the referring hostname is not identical to the current hostname (or nocache.current hostname) a referer_domain key/value should be added to the query string of the image.
Dealer Search	This crumb appears on the first page of a dealer search results page. It has an OBJECT_TYPE of MDTRANSIT_DEALER_SEARCH and OBJECT_ID of the template property ID (from template manager). The query string of the image contains the following data: make searched for (make=), zip code searched in (zipcode=) and the type of dealer (type=).

Types are freeform and what appears in the query string appears on the reports.

Example:

```
http://rpt.mdtransit.morris.com/images/mdwtc-${PLACEMENT_CODE}-
MDTRANSIT_DEALER_SEARCH-
${SITE ID}.gif?make=TOYOTA&zipcode=30909&type=(NEW/USED/NEW+AND
ED
```

Dealer Details

When dealership info is displayed (i.e. dealer searches) a crumb appears with an OBJECT_TYPE of MDTRANSIT_DEALER_DETAILS and OBJECT_ID of the customer ID for that dealer in the classifieds system.
Example:

```
http://rpt.mdtransit.morris.com/images/mdwtc-
${PLACEMENT_CODE}-MDTRANSIT_DEALER_DETAILS-${CUSTOMER_ID}.gif
```

Auto Search

On first page of any search results page a crumb appears with an OBJECT_TYPE of MDTRANSIT_INVENTORY_SEARCH and OBJECT_ID of the template property ID (in template manager). The query string of the image contains the following data:

- makes searched for (broken out using make.#= syntax) Digits are currently configured to search from 1 to 10.
- common model names searched for (i.e. F-Series for ford, not F-150, F-250, F-350) Broken out in the same way as the makes.
- zip codes searched in (zipcode=)
- type of auto (type=).

Types, makes and zip codes are freeform and what appears in the query string appears on the reports. Only canonical model values are detailed below the makes, other values are hidden - canonical values are found in the classifieds codesets category MDTRANSIT, key of MODEL.
Example:

```
http://rpt.mdtransit.morris.com/images/mdwtc-
${PLACEMENT_CODE}-MDTRANSIT_INVENTORY_SEARCH-
${SITE_ID}.gif?make.1=TOYOTA&model.1=Camry&zipcode=30909&type
=(NEW/USED/NEW+AND+USED)
```

Auto Display

When auto information is displayed (i.e. inventory search returns; detail pages) a crumb appears with an OBJECT_TYPE of MDTRANSIT_AD and OBJECT_ID of the customer ID for that dealer in the classifieds system. The query string of the ad sends the ad id (ad_id=), make (make=) and model (model=). These are all free form and what appears in the query string is what appears in the reports. Ad ID is not currently stored or displayed.

Example:

```
http://rpt.mdtransit.morris.com/images/mdwtc-
${PLACEMENT_CODE}-MDTRANSIT_AD-
${CUSTOMER_ID}.gif?ad_id=12345&make=TOYOTA&model=CAMRY
```

Dealer Referral

When linking to a dealer's website, an intermediate page loads containing a crumb with an OBJECT_TYPE of MDTRANSIT_DEALER_URL and OBJECT_ID of the customer ID for that dealer in the classifieds system.

Example:

```
http://rpt.mdtransit.morris.com/images/mdwtc-
${PLACEMENT_CODE}-MDTRANSIT_DEALER_URL-${CUSTOMER_ID}.gif
```

Placement Codes

Code	Page	Code	Page
100	Vehicle Results	115	Vehicle Quote Thankyou
101	Vehicle Details	116	Research & Compare
102	Advanced Search	117	Finance Index
103	No Vehicle Results	118	No Dealer Results
104	Full Image	119	E-mail Notification Form
105	Quick Search/Index	120	Quick Quote Form
106	Specification Search	121	E-mail Notification Thankyou
107	Dealer Search	122	Quick Quote Thankyou
108	Specification Results	123	Parts Request
109	Dealer Search Results	124	Service Appointment
110	Comparison	125	Clipboard
111	Comparison Step 3	126	E-mail ad to friend
112	My Selected Vehicles	127	E-mail ad to friend thank you
113	Comparison Step 2	128	Printer Friendly Ad details
114	Calculator		

Crumb Data Storage

This section explains where the mdTransit report data is stored once it is received from the tracking crumb. Each section below provides a breakout of the log file pertaining to only that specific report.

Site Traffic Crumb

```
LOG_OBJECTS
   OBJECT_ID = PROPERTY_ID from Template Manager
   OBJECT_TYPE = MDTRANSIT_SITE

LOG_OBJECT_ATTRIBUTE (keys)
   DATE
      DATE HIT_COUNT

      PLACEMENT_CODE
         PLACEMENT_CODE HIT_COUNT

         REFERER_DOMAIN (this must be passed within the image call)
```

```
            REFERER_DOMAIN HIT_COUNT
```

Dealer Search Crumb

```
LOG_OBJECTS
   OBJECT_ID = PROPERTY_ID
   OBJECT_TYPE = MDTRANSIT_DEALER_SEARCH

LOG_OBJECT_ATTRIBUTE
   DATE
      DATE HIT_COUNT
         MAKE
            MAKE HIT_COUNT
         ZIP
            ZIP HIT_COUNT
         TYPE
                  TYPE HIT_COUNT
```

Dealer Details Crumb

```
LOG_OBJECTS
   OBJECT_ID = CUSTOMER_ID
   OBJECT_TYPE = MDTRANSIT_CUSTOMER_DETAILS

LOG_OBJECT_ATTRIBUTE (keys)
   DATE
      DATE HIT_COUNT
```

Auto Search Crumb

```
LOG_OBJECTS
   OBJECT_ID = PROPERTY_ID
   OBJECT_TYPE = MDTRANSIT_INVENTORY_SEARCH

LOG_OBJECT_ATTRIBUTE
   DATE
      DATE HIT_COUNT
         TYPE
            TYPE HIT_COUNT
         ZIP
            ZIP HIT_COUNT
            MAKE
               MAKE HIT_COUNT
            MODEL
               MODEL HIT_COUNT
```

Auto Display Crumb

```
LOG_OBJECTS
   OBJECT_ID = AD_CUSTOMER_ID
   OBJECT_TYPE = MDTRANSIT_AD
```

```
LOG_OBJECT_ATTRIBUTE
   DATE
      DATE HIT_COUNT
      PLACEMENT_CODE
         PLACEMENT_CODE HIT_COUNT
         MAKE
            MAKE HIT_COUNT
         MODEL
            MODEL HIT_COUNT
```

Dealer Referral Crumb

```
LOG_OBJECTS
   OBJECT_ID = CUSTOMER_ID
   OBJECT_TYPE = MDTRANSIT_DEALER_URL

LOG_OBJECT_ATTRIBUTE (keys)
   DATE
      DATE HIT_COUNT
```

Implementing Tracking Crumbs

The following section explains how to implement tracking crumbs to gather report data. The following information is explained for each tracking crumb:

- who the report data is aimed at(i.e MBU or dealer)
- how to implement the tracking crumb

Summary of Traffic & Referring Sites

This report is provided to MBUs and provides the hit counts per autos page/page type and count of referring sites. Dealers do not receive this type of data.

Implementation

Place one crumb on each page of the site that passes a placement code as well as the referring domain.

Example:

mdwtc-${PLACEMENT_CODE}-MDTRANSIT_SITE-${SITE_ID}.gif?referer_domain=domain.com

Dealer Link Outs(Dealer Referral)

This report is provided to MBUs and provides a count by dealership. Dealers receive reports for the number of times the dealership was linked to.

Implementation

Open a JavaScript window that is bugged and has a meta refresh to perform the link.

Example:

mdwtc-${PLACEMENT_CODE}-MDTRANSIT_DEALER_URL-${CUSTOMER_ID}.gif

Inventory Summary

This report is provided to MBUs and provides the following:

- number of vehicles in inventory (live ads)

- sub-counts by number with photos, price, and exterior color
- Counts by categories of certified dealer, new dealer, used dealer, classifieds.

Dealers can view the following:

- number of vehicles in inventory (live ads)
- sub-counts of number with photos, with price, with exterior color

Implementation
This report data is pulled from live data.

Dealer Leads

This report provides both MBUs and Dealers with a downloadable count of generated leads.
Implementation
The data is stored in ad_response. Place the data in KEY=VALUE sequence, and pull back with loop for output to dealer or paper.

Dealer Locator (find a dealership)

This information provides MBUs with the number of dealer searches performed, broken out by make and zip code. Dealers see the number of times the dealership was shown in locator search results
Implementation
On the first dealer search return page, place a tracking crumb that pulls the make and zip code searched for data. Place tracking crumbs in each dealer details section as shown below:
Examples:
Enter the following crumb for each dealer:
mdwtc-${PLACEMENT_CODE}-MDTRANSIT_DEALER_DETAILS-${CUSTOMER_ID}.gif

The dealer search crumb is
mdwtc-${PLACEMENT_CODE}-MDTRANSIT_DEALER_SEARCH-
${SITE_ID}.gif?make=TOYOTA&zipcode=30909&type=(NEW/USED/NEW+USED)

Inventory Searches

MBUs view the number of inventory searches performed, broken out by:

- type (new/used/combined)
- make
- model
- zip code

Dealers view the number of inventory searches performed, broken out by:

- type (new/used/combined)
- make
- model
- zip code

Implementation

Place a crumb on the first inventory search return page, that pulls the following data: type, make, model and zip code searched for.
Example:
mdwtc-${PLACEMENT_CODE}-MDTRANSIT_INVENTORY_SEARCH-${SITE_ID}.gif?referer_domain=domain.com&make.#=TOYOTA&model.#=&zipcode=30909&type=(NEW/USED/NEW+AND+USED)

Car Display Details

MBUs do not need this information. Dealers view the number of times the inventory was shown on various pages, broken out by make and model.

Implementation

For each ad, place a crumb that pulls the placement indicator and ad id.
Example:
mdwtc-${PLACEMENT_CODE}-MDTRANSIT_AD-${DEALER_ID}.gif?ad_id=&make=&model=

The CrumbLog

After the tracking data has been stored in the Access logs, it must be parsed and inserted into a database for retrieval. This is accomplished by the CrumbLog program. The CrumbLog program processes the access logs collected by the mdTransit reporting server and uses a tracking item data file to convert the log lines into attributes that are then loaded into the database. The number of times each attribute is seen is also stored.

Notes:

- The CrumbLog program can also be run as an Apache module.
- All CrumbLog program operations on files must occur in the same logical file system. Thus, all data source files, archive directories, output directories, log and configuration directories must all be on the same logical file system.

Attribute Tables

After being parsed from the Access log, objects are loaded into MDW_UTIL.LOG_OBJECT using the CRUMB_ID (stored as the object_id and a translated OBJECT_TYPE based on the CRUMB_TYPE (translations are stored in a conf file). The KEY and VALUE (attributes described by the tracking item file associated with the crumb type) are stored in the LOG_OBJECT_ATTRIBUTE table, using the PARENT value to determine the value of the parent_id and which if the following actions to perform:

- whether the value should be stored
- whether it should be used to lookup another value for storage based on the object type (for retrieving information in the db without having to pass it on the query string)
- whether a child hit count attribute record should be created or incremented.

The CrumbLog attribute tables are shown below.

MDW_UTIL.LOG_OBJECT	MDW_UTIL.LOG_OBJECT_ATTRIBUTE
LOG_OBJECT_ID	LOG_OBJECT_ATTRIBUTE_ID
OBJECT_ID	LOG_OBJECT_ID
OBJECT_TYPE	KEY
DATE_CREATED	VALUE

LAST_UPDATED	VALUE_TYPE VALUE_FORMAT PARENT_ID DATE_CREATED LAST_UPDATED

Data Files

The following files are necessary for the proper functioning of the CrumbLog program. All new and modified values in the files must be placed in quotes and be separated by a tab.

- **object_codes.dat** - contains information used to translate crumb_types to object_types. If no translation is found, the crumb_type is stored as object_type.

  Format: "CRUMB TYPE" "OBJECT TYPE"

- **placement_codes.dat** - contains information used to translate placement codes to easily readable strings.

  Format: "PLACEMENT CODE" "PLACEMENT"

- **tracking_items.dat** - contains information used to create attributes for storage out of crumb requests.

  Format: "ID" "Crumb Type" "Key" "Source" "Transformation" "Parent" "Value Type" "Value Format" "Failure Action"

Data Tables

MDW_UTIL.OBJECT_CODE	**MDW_UTIL.PLACEMENT_CODE**	**MDW_UTIL.TRACKING_ITEM**
CODE	CODE	TRACKING_ITEM_ID
TRANSLATION	TRANSLATION	OBJECT_TYPE
		KEY
		VALUE_DATA_SOURCE
		VALUE_TRANSFORMATION
		PARENT_ID
		HIT_COUNT VALUE_TYPE
		VALUE_FORMAT
		PARSE_FAILURE

Attribute	Explanation
TRACKING_ITEM_ID	Printed during logging to aid in tracking down errors.
OBJECT_TYPE	Indicates which crumb the tracking item is for. An asterisk(*) is used to indicate all crumbs. This type name must match the object type requested in the image name (not the translated type used for storage).

KEY	The key for the attribute to store for this tracking item. In cases where the value is URL_KEY_VALUE, this is the key for the parent attribute, stored as a blank value. If the key is blank, it is ignored.
VALUE_DATA_SOURCE	Identifies the portion of the access_log containing the value to store. Allowable values include: blank, IP, DATE, HOUR_OF_DAY, DAY_OF_WEEK, REQUEST, REFERER, QUERY_STRING or DATABASE_LOOKUP
VALUE_TRANSFORMATION	• FEATURE_QUANTITY(KEY) - put into loadable format, looked up from database using translated OBJECT_TYPE. Information comes from feature_quantity column with key of KEY. • FEATURE_VALUE(KEY) - put into loadable format, looked up from database using translated OBJECT_TYPE. Information comes from feature_value column with key of KEY. • LOOKUP(KEY) - put into loadable format, looked up from database using translated OBJECT_TYPE. Value comes from classifieds table column with name matching KEY. • URL_KEY_VALUE - treat the source as a sequence of KEY/VALUE pairs to be stored. One attribute is created for each key/value pair found. These are stored as children of an attribute with the given tracking item key, and has a null value.
PARENT	The parent ID of the tracking crumb. When converted to the tracking item data file, the parent is specified in the format KEY(VALUE).KEY(SOURCE.VALUE).KEY . • SOURCE is optional and used to select any of the allowable sources except DATABASE_LOOKUP. • VALUE is treated as a translation and can be any of the values (VALUE, URL_KEY(KEY), PLACEMENT_CODE_ID, PLACEMENT_CODE, OBJECT_TYPE, TRANSLATED_OBJECT_TYPE, OBJECT_ID, DOMAIN, URI_PATH). **Note:** Values which may contain parentheses when translated are not allowed.
PARSE_FAILURE	Action to take if parsing for the attrribute fails. Allowable values are: LOG or IGNORE. This should usually be set to IGNORE for tracking items where data may or may not be present for a crumb. This prevents spurious errors.

Tracking Item Data File

The tracking item data file is used by the CrumbLog program to convert the process the access logs and convert the data into attribute tables. The tracking items are composed of:

Attribute	Explanation
ID	Created during logging to aid in debugging.
OBJECT_TYPE	Indicates which crumb the tracking item is for. An asterisk(*) is used to indicate all crumbs. This type name must match the object type requested in the image name (not the translated type used for storage).
KEY	The tracking item key for the attribute to store. If the value is URL_KEY_VALUE, this is the key for the parent attribute, stored with a blank value. If key is blank, it is ignored.
SOURCE	The location of the value to store. Identifies the part of the access_log containing the desired values. Allowable values are IP, DATE, HOUR_OF_DAY, DAY_OF_WEEK, REQUEST, REFERER, QUERY_STRING and DATABASE_LOOKUP
VALUE	This modifies the source to obtain the value for storage. Allowable values include: • BLANK - Stores the value of the source. BLANK means no value, not the literal string BLANK. • DOMAIN- Searches for a domain in the source. • FEATURE_QUANTITY(KEY)- Converts data into loadable format, looked up from database using translated OBJECT_TYPE information. • FEATURE_VALUE(KEY)- Converts data into loadable format, looked up from database using translated data. • OBJECT_TYPE • PLACEMENT- Searches the source for a placement code, then translates and stores it. • PLACEMENT_CODE- Searches the source for a placement code and stores it. • OBJECT_TYPE- Searches the source for an object type and stores it. • OBJECT_ID- Searches the source for an object id and stores it. • TRANSLATED_OBJECT_TYPE- Searches the source for an object type, then translates and stores it. • URI_PATH- Searches the source for the path part of a URL and stores it. • URL_KEY_VALUE- Treats the source as a sequence of KEY/VALUE pairs to be stored. One attribute is created for each key/value pair. • URL_KEY(KEY)- Gets the value referenced by KEY.

	• VALUE- Stores the value of the source.
ACTION	• INCREMENT- Stores the value and increments the child hit count attribute. • STORE_VALUE- Stores the value. • HIT_COUNT- Stores the hit count as a value instead of in the child record.
PARENT	This is specified in the following format: KEY(VALUE).KEY(SOURCE.VALUE).KEY . • Source is optional and used to select any of the allowable sources except DATABASE_LOOKUP. • VALUE is treated as an action and can be any of the values (VALUE, URL_KEY(KEY), PLACEMENT_CODE_ID, PLACEMENT_CODE, OBJECT_TYPE, TRANSLATED_OBJECT_TYPE, OBJECT_ID, DOMAIN, URI_PATH).
VALUE_TYPE	Type for the attribute to store.
VALUE_FORMAT	Format for the attribute to store.

Log Setup

This section explains the initial setup steps that must be performed before running the CrumbLog. To setup the CrumbLog program to run you must do the following things:

1. Verify that the Access log uses the proper data format.
2. Create a directory structure for the CrumbLog program to operate in.
3. Create an initial configuration file.

Setting up the Access Log

The CrumbLog program expects access log data in the format describe by the following Apache format string:
"%h %{%d-%b-%Y %H %u}t %U %{Referrer}i %q"

Which breaks down to:
IPADDRESS DATE HOUR_OF_DAY DAY_OF_WEEK CALLED_FILE REFERER QUERY_STRING.

Generally a URL rewrite rule is employed to convert requests for crumbs into requests for a 1x1 pixel clear gif file. The regular expression that describes a crumb request is:
mdwtc-[_a-zA-Z0-9]+\-[_a-zA-Z0-9]+\-[0-9]+.gif$
This is usually requested from an images directory (/images/mdwtc-$PLACEMENT_CODE-$CRUMB_TYPE-$OBJECT_ID.gif). This regular expression may work as is in your server rewrite rule but that depends on the regular expression library compiled into your server. You should check the current regular expression as a precaution.

Setting Up the Configuration File

To set up the CrumbLog operation for the first time, you must create an initial configuration file. This file dictates the operation of the CrumbLog program through its use of variables. After initial setup, you will only need to modify the variables in this file if you want to change one of the variables.

To setup the configuration file:

1. Create a configuration file as shown in the example below.
2. Verify that the variables are all set to your desired values.
3. Point the CrumbLog program to the configuration file by running the CrumbLog using the -c option or by setting the CRUMBLOG_HOME environmental variable.

   Note: If you are using the CRUMBLOG_HOME environmental variable, the configuration file must be located at $CRUMBLOG_HOME/conf/CrumbLog.cfg

4. Run the CrumbLog program with the -R option to create the tracking items, placement code and object types data files from the data stored in the database.
5. You can now run the CrumbLog program for processing and loading data.

Sample Configuration File

```
# the following is for appsvr
dbconnect=kKWbsb+uY6GdrdG4laSPd8GjlqU
# the following is for prod(mdw_util)
#dbconnect=kKSknNO2jKxtrdCxh3qisceu
PLACEMENT_CODE_FILE=/var/spool/crumb_log/conf/placement_codes.dat
OBJECT_CODE_FILE=/var/spool/crumb_log/conf/object_codes.dat
TRACKING_ITEMS_FILE=/var/spool/crumb_log/conf/tracking_items.dat
CRUMBLOG_FILE=/var/spool/crumb_log/log/atl.log
DB_COMMIT_THRESHOLD=1000
LOAD_THRESHOLD=10000
#DEBUG_LEVEL default = 0 (0=off, 1=basic, 2=errors, 4=warnings,
# 6=alerts, 8=debugging,
# 10=tattle, you don't want to do that)
DEBUG_LEVEL=2
# If loader run time falls in one of the hours listed below then
# ignore LOAD_THRESHOLD (process/load whole file)
# format of value: comma delimited, or specify * for all hours
OFF_HOURS=1,2,3,4
# ARCHIVE=TRUE/FALSE
ARCHIVE=FALSE
ARCHIVE_DIRECTORY=/var/spool/crumb_log/archive/
ACCESS_LOG_DIRECTORY=/var/spool/crumb_log/data/
PARSED_LOG_DIRECTORY=/var/spool/crumb_log/parsed/
ERROR_DIRECTORY=/var/spool/crumb_log/log/errors/
LEFTOVER_FILE_SUFFIX=.left
EMAIL_TO=chris.johnson@morris.com
EMAIL_SUBJECT=ATL: An error occurred
EMAIL_PROGRAM=/usr/sbin/sendmail
# value should be set to level of log message necessary to
```

```
# generate email, generally 2 or higher
# (0=off, 2=errors, 4=warnings, 6=alerts, 8=debugging, 10=tattle
# basic log messages (level 1) won't generate emails
EMAIL_LEVEL=0
```

CrumbLog Configuration Options

The following variables are contained in the CrumbLog configuration file and can be modified.

Variable	Function
dbconnect	Encrypted connection string for the database.
PLACEMENT_CODE_FILE	The file path of the placement_code table data.
OBJECT_CODE_FILE	The file path of the object_code table data.
TRACKING_ITEMS_FILE	The file path of the tracking_items table data.
CRUMBLOG_FILE	The file path and name that contains the log messages.
DB_COMMIT_THRESHOLD	Defines the number of loading objects to handle before performing a commit. Approximately 1 insert/update is performed per object. The number of objects created by a crumb depends on the number of tracking items applied for that crumb and what they create.
LOAD_THRESHOLD	Sets the maximum number of objects to load in a single run. When parsing, or parsing and loading, this has no effect. When loading, this is the number of distinct attributes that are handled. **Note:** Two or more consecutive identical lines count as 1.
DEBUG_LEVEL	Defines the level of messages that are logged to the CRUMBLOG_FILE. 0 = disables logging 1= logs basic messages (start/stop/action requested) 2 = errors 4 = warnings 6 = alerts 8 = debugging 10 = tattle **Caution:** Debug and Tattle record more information than standard logging. It is helpful in troubleshooting problems, however, can create large log files and increase the system load.
OFF_HOURS	Hours during which LOAD_THRESHOLD is

	ignored. Enter an asterisk (*) for all hours. Hours must be comma separated, and in military time (i.e. 0,1,2 . . 21,22,23,24).
ARCHIVE	Moves data files to the ARCHIVE_DIRECTORY after processing. Acceptable Values: TRUE or FALSE.
ARCHIVE_DIRECTORY	Directory where data files are stored after processing.
ACCESS_LOG_DIRECTORY	Directory where unprocessed files are located.
PARSED_LOG_DIRECTORY	Directory to store files that have been created by requesting parsing of data files.
ERROR_DIRECTORY	Directory in which error files are stored.
LEFTOVER_FILE_SUFFIX	Suffix to append to files when objects are left after the LOAD_THRESHOLD is reached.
EMAIL_TO	E-mail address to send messages to when errors occur.
EMAIL_SUBJECT	Subject of the error e-mail messages. (See above)
EMAIL_PROGRAM	Program used to send e-mails. The program should support sendmail command line options: -t and -F
EMAIL_LEVEL	Level of message necessary to generate an e-mail. Levels are the same as DEBUG_LEVEL except that level 1 messages never trigger e-mails.

CrumbLog Command Line Options

The CrumbLog program is run from the command line using the syntax and options listed below.

Command Option	Function
-?	Print usage instructions
db *string*	Override dbconnect string in configuration file with given string.
-c *file name*	Specify a configuration file to use other than that found in $CRUMBLOG_HOME/conf/CrumbLog.cfg
-f *filename*	File to process instead of those found in directories specified in the configuration file (directory searched depends on action requested)
-L	Load objects into database.
-log *filename*	Specify a log file to use instead of that found in the configuration file

-o *output file*	Specify an output file name instead of allowing program to derive one based on input file name and processing time.
-R	Regenerate configuration files (placement codes, object codes and tracking items)
-P	Process data file and output to file (file output is skipped if specified in combination with -L)
-v#	Override debug level set in configuration file.
-version	Print version string.