Adobe® Marketing Cloud
Dataset Configuration
Contents

Dataset Configuration .............................................................................................................. 6
  Understanding Dataset Construction ................................................................................. 6
    Log Processing ................................................................................................................. 6
    Transformation ............................................................................................................... 7
  Understanding Dataset Configuration ............................................................................. 8
    Required Configuration Files ....................................................................................... 8
    Additional Configuration Files ..................................................................................... 9
    Next Steps .................................................................................................................... 9

Log Processing Configuration File ...................................................................................... 11
  About the Log Processing Configuration File ................................................................ 11
  Editing the Log Processing Configuration File ............................................................. 11
  Log Sources .................................................................................................................. 15
    Sensor Files .................................................................................................................. 16
    Log Files ....................................................................................................................... 17
    XML Log Sources ......................................................................................................... 20
    Avro Data Feed ............................................................................................................. 23
  Configuring a Data Workbench Server File Server Unit ............................................. 24
    About Data Workbench Server File Server Units ....................................................... 24
    The File Server Configuration Process ..................................................................... 25
  Information About Specific Log Processing Parameters ........................................... 31
    Data Filters .................................................................................................................. 31
    Log Entry Condition .................................................................................................... 32
    Key Splitting ................................................................................................................ 32
  ODBC Data Sources ...................................................................................................... 33
  Considerations for the Log Processing Configuration File ........................................... 36

Transformation Configuration File ..................................................................................... 38
  About the Transformation Configuration File .............................................................. 38
  Editing the Transformation Configuration File ............................................................. 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information About Specific Transformation Parameters</td>
<td>41</td>
</tr>
<tr>
<td>New Visitor Condition</td>
<td>41</td>
</tr>
<tr>
<td>Time Zones</td>
<td>42</td>
</tr>
<tr>
<td>Considerations for the Transformation Configuration File</td>
<td>42</td>
</tr>
<tr>
<td>DeviceAtlas Distribution</td>
<td>43</td>
</tr>
<tr>
<td>Dataset Include Files</td>
<td>46</td>
</tr>
<tr>
<td>About Dataset Include Files</td>
<td>46</td>
</tr>
<tr>
<td>Working with Dataset Include Files</td>
<td>46</td>
</tr>
<tr>
<td>Editing Existing Dataset Include Files</td>
<td>46</td>
</tr>
<tr>
<td>Creating New Dataset Include Files</td>
<td>47</td>
</tr>
<tr>
<td>Types of Dataset Include Files</td>
<td>48</td>
</tr>
<tr>
<td>Log Processing Dataset Include Files</td>
<td>48</td>
</tr>
<tr>
<td>Transformation Dataset Include Files</td>
<td>57</td>
</tr>
<tr>
<td>Defining Parameters in Dataset Include Files</td>
<td>58</td>
</tr>
<tr>
<td>String and Numeric Parameters</td>
<td>59</td>
</tr>
<tr>
<td>Vector Parameters</td>
<td>60</td>
</tr>
<tr>
<td>Additional Configuration Files</td>
<td>61</td>
</tr>
<tr>
<td>Log Processing Mode.cfg</td>
<td>61</td>
</tr>
<tr>
<td>Server.cfg</td>
<td>63</td>
</tr>
<tr>
<td>Other Files</td>
<td>64</td>
</tr>
<tr>
<td>Data Transformations</td>
<td>65</td>
</tr>
<tr>
<td>About Transformations</td>
<td>65</td>
</tr>
<tr>
<td>Conventions for Constructing Transformations</td>
<td>65</td>
</tr>
<tr>
<td>Defining a Transformation</td>
<td>66</td>
</tr>
<tr>
<td>Types of Transformations</td>
<td>67</td>
</tr>
<tr>
<td>Standard Transformations</td>
<td>67</td>
</tr>
<tr>
<td>URI Transformations</td>
<td>90</td>
</tr>
<tr>
<td>DeviceAtlas Distribution</td>
<td>93</td>
</tr>
<tr>
<td>Integrating Lookup Data</td>
<td>95</td>
</tr>
<tr>
<td>Populating the Lookup Table</td>
<td>96</td>
</tr>
</tbody>
</table>

Last updated 2/22/2017
Dataset Configuration

Overview of the dataset construction process and the dataset configuration files that control the processing phases.

Understanding Dataset Construction

An Adobe dataset contains the data that has been loaded and processed by the data workbench server. The steps involved in the loading and processing of the data by the data workbench server (InsightServer64.exe) make up the dataset construction process.

Note: A data workbench server that processes and serves data from an Adobe dataset is called a data processing unit or DPU. It is sometimes referred to as a processing server or a query server. Data workbench and Report clients interact with DPUs directly.

During dataset construction, the data workbench server reads source data from log sources, applies transformations to specific fields of data, and defines extended dimensions to be created from the transformed fields. The construction process occurs in two phases: Log Processing and Transformation. After the dataset is constructed, you can use the dataset’s extended dimensions to create derived metrics and dimensions for your specific analysis purposes.

Dataset construction is like a manufacturing process. You select the data (the raw materials) to be used to build the dataset, and you define the data transformations (the process steps) that manipulate the information available in the data to create extended dimensions (the manufactured products).

Log Processing

During the log processing phase of dataset construction, the sources of data for the dataset are specified, and the desired data fields are extracted and manipulated into more useful forms.

The logs are filtered, and the fields of data that are to be passed to the transformation phase are identified. At the end of the log processing phase, the data is grouped by tracking ID (that is, all log entries with the same tracking ID are grouped together) and ordered in time. For information about tracking IDs, see Specifying Log Sources.

During the log processing phase, you cannot access the processed data to use for analysis.

The following sections describe the processing steps that occur during log processing:

• Specifying Log Sources
• Defining Transformations
• Filtering Logs
• Identifying Fields for Transformation

Specifying Log Sources

Log sources are files that contain the data to be used to build a dataset. The data available in the log sources is called event data because each data record represents a transaction record or a single instance of an event. In addition, each record, or log entry, contains a value referred to as a tracking ID.

Note: When selecting log sources, make sure that each log entry contains a tracking ID for the entity that is to represent the highest level at which your data is to be grouped. For example, if you are working with data collected from website traffic, you are likely to choose visitor to be this entity. Each visitor has a unique tracking ID, and all of the data about a particular site visitor can be grouped together. For assistance, contact Adobe.
A log source event data is collected in real-time by **Sensors** or extracted from archived data sources by Insight Server. Event data collected by Sensors from HTTP and application servers is transmitted to Insight Servers, which convert the data into highly compressed log (.vsl) files. Event data that resides in a flat file, XML file, or an ODBC data source is read by Insight Server, which provides decoders that you define to extract a common set of log fields from these different formats.

**Defining Transformations**

A transformation is a set of instructions that you can define to extract or manipulate information in the event data. Each transformation that you define is applied to each event data record (log entry) to update existing log fields or produce new fields. The results of transformations are used along with log entry conditions to evaluate which log entries are filtered out of the dataset during log processing.

Not all types of transformations can be used during the log processing phase of the dataset construction process. For information, see *Data Transformations*.

**Filtering Logs**

The dataset contains several parameters used to filter the data flowing out of the transformations. Filtering is used to specify which log entries are used in subsequent processing steps. For example, filters can be defined by, time range, the status of the server’s response, or IP address and user-agent information. The **Log Entry Condition** is a customizable filtering test. The test looks for certain conditions in the fields of each log entry to determine whether that entry should proceed further in the dataset construction process. If a log entry does not meet the condition, the entry is removed from the construction process.

**Identifying Fields for Transformation**

If a field of data is to be passed from the log processing phase to the transformation phase for further processing, you must identify it during log processing. This requirement applies regardless of whether the field is available from the log sources or created from data transformations applied to the data during log processing.

**Transformation**

During the transformation phase of dataset construction, processing occurs on the grouped and ordered data that is output from log processing.

During the transformation phase of dataset construction, processing occurs on the grouped and ordered data that is output from log processing. Additional data transformations are performed and extended data dimensions are created for use in your analyses. During the transformation phase, you can access a statistical sample of the data that gets larger as the transformation phase nears completion.

The following sections describe the processing steps that occur during transformation:

- **Defining Transformations**
- **Filtering Logs**
- **Defining Extended Dimensions**

**Defining Transformations**

You can define transformations to be used during the transformation phase of the dataset construction process to facilitate the creation of the extended dimensions. Each transformation is applied to each event data record (log entry) passed from log processing.
Filtering Logs

The Log Entry Condition can be applied during transformation to look for specific conditions in the fields of each log entry coming from log processing. If a log entry does not meet the condition, the entry is removed from the construction process.

Defining Extended Dimensions

Extended dimensions are the final products of the dataset construction process. They represent relationships between the log fields in the data. You use them to create visualizations, build extended metrics, or perform analysis to understand the operations and issues specific to your business.

Understanding Dataset Configuration

Dataset configuration refers to the process of editing the configuration files whose parameters provide the rules for dataset construction.

The constructed dataset physically resides in the temp.db file stored on the data workbench server computer, but the configuration files for the dataset reside within a directory for a profile. A profile contains a set of configuration files that construct a dataset (including its extended dimensions) for a specific analysis purpose. In addition, a profile contains the definitions of entities such as metrics, derived dimensions, workspaces, reports, and visualizations that enable analysts to interact with the dataset and obtain information from it.

The profile whose dataset configuration files you are editing is referred to as your dataset profile. A dataset profile references multiple inherited profiles, which can be any profiles that you create and maintain so that you can configure your Adobe application to best fit your analysis needs. A dataset profile also may reference internal profiles that are provided with your Adobe application to form the basis for all of the functionality available in your application.

For more information about the different types of profiles that are available with Adobe applications, see the Data Workbench User Guide.

Required Configuration Files

The dataset profile's required configuration files control the processing phases (log processing and transformation) through which incoming data flows to keep the dataset up to date and specify the inherited profiles that are to be included in the dataset profile.

A dataset profile for any Adobe application must contain the following configuration files on the Insight Server machine:

- **Profile.cfg**: Lists the inherited profiles and processing servers for the profile. Processing servers are the Insight Server DPU’s that process the data for the profile. If you have installed an Insight Server cluster, you can specify multiple Insight Server computers to run a single profile.

    For instructions to add inherited profiles to a dataset profile’s Profile.cfg file, see the Server Products Installation and Administration Guide. For information about installing an Insight Server cluster or configuring a dataset profile to run on an Insight Server cluster, see the Server Products Installation and Administration Guide.

- **Dataset\Log Processing.cfg**: Controls the log processing phase of the dataset construction process. See Log Processing. For more information about the Log Processing.cfg file, see Log Processing Configuration File.

- **Dataset\Transformation.cfg**: Controls the transformation phase of the dataset construction process. See Transformation. The Transformation.cfg file typically configures the dataset for profile-specific analysis. For more information about the Transformation.cfg file, see Transformation Configuration File.
• **Dataset Include Files:** A dataset include file contains a subset of the parameters contained in the Log Processing.cfg or Transformation.cfg file for the dataset profile but is stored and managed within an inherited profile. **Dataset include** files supplement the main dataset configuration files. For more information, see [Dataset Include Files](#).

The dataset profile provided to you during the implementation of your Adobe application contains a set of dataset configuration files that you can open, edit, and save using the **Profile Manager**.

For information about the **Profile Manager**, see the [Insight User Guide](#).

**Additional Configuration Files**

Additional dataset configuration files reside in the profile’s **Dataset directory** on the data workbench server computer. Although not required for all datasets, these files enable you to control other aspects of the dataset construction process:

• **Log Processing Mode.cfg:** The Log Processing Mode.cfg file lets you pause processing of data into a dataset, specify offline sources, or specify the frequency at which the data workbench server saves its state files. See [Additional Configuration Files](#).

• **Server.cfg:** The Server.cfg file specifies the default data cache size (in bytes) for data workbench machines that connect to the data workbench server. See [Additional Configuration Files](#).

• **Transform.cfg and Transform Mode.cfg:** These files are available only if you have licensed the data transformation functionality to use with your Adobe application. The Transform.cfg file contains the parameters that define the log sources and data transformations for transformation functionality. The transformations that you define manipulate the source data and output it into a format that you specify. The Insight Transform Mode.cfg file enables you to pause processing of data into a dataset, specify offline sources, or specify the frequency at which the Insight Server running transformation functionality saves its state files. See [Transform Functionality](#).

**Next Steps**

Information about specific dataset configuration tasks.

For detailed information about the parameters contained in the dataset configuration files, see:

• **Log Processing Configuration File**

• **Transformation Configuration File**

• **Dataset Include Files**

• **Additional Configuration Files**

For information about specific dataset configuration tasks, use the table below to locate and read about the tasks of interest of you:

<table>
<thead>
<tr>
<th>If you would like to...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define log sources</td>
<td>Log Sources</td>
</tr>
<tr>
<td>Determine which log entries enter the dataset during log processing</td>
<td>Data Filters, Log Entry Condition</td>
</tr>
<tr>
<td>Enable the splitting of tracking IDs with large amounts of event data</td>
<td>Key Splitting</td>
</tr>
<tr>
<td>If you would like to...</td>
<td>See...</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Configure an Insight Server to run as a file server unit</td>
<td>Configuring a Data Workbench Server File Server Unit</td>
</tr>
<tr>
<td>Configure an Insight Server to run as a centralized</td>
<td>Configuring a Data Workbench Server File Server Unit</td>
</tr>
<tr>
<td>normalization server</td>
<td></td>
</tr>
<tr>
<td>Set the time zone to be used for creating time dimensions</td>
<td>Time Zones</td>
</tr>
<tr>
<td>and making time conversions</td>
<td></td>
</tr>
<tr>
<td>Make minor changes to the dataset configuration files</td>
<td>Editing Existing Dataset Include Files</td>
</tr>
<tr>
<td>included with the internal profiles provided by Adobe</td>
<td></td>
</tr>
<tr>
<td>Specify new fields of data to be passed from log processing</td>
<td>Creating New Dataset Include Files</td>
</tr>
<tr>
<td>to transformation</td>
<td>Log Processing Dataset Include Files</td>
</tr>
<tr>
<td>Define transformations</td>
<td>Data Transformations</td>
</tr>
<tr>
<td></td>
<td>Creating New Dataset Include Files</td>
</tr>
<tr>
<td></td>
<td>Transformation Dataset Include Files</td>
</tr>
<tr>
<td>Create extended dimensions</td>
<td>Extended Dimensions</td>
</tr>
<tr>
<td></td>
<td>Creating New Dataset Include Files</td>
</tr>
<tr>
<td></td>
<td>Transformation Dataset Include Files</td>
</tr>
<tr>
<td>Define parameters to use throughout log processing or</td>
<td>Defining Parameters in Dataset Include Files</td>
</tr>
<tr>
<td>transformation</td>
<td></td>
</tr>
<tr>
<td>Learn about the Insight interfaces that enable you to monitor</td>
<td>Working With Dataset Configuration Interfaces</td>
</tr>
<tr>
<td>or manage your dataset</td>
<td></td>
</tr>
<tr>
<td>Hide certain extended dimensions so they do not show on</td>
<td>Hiding Dataset Components</td>
</tr>
<tr>
<td>the dimension menu in Insight</td>
<td></td>
</tr>
<tr>
<td>Override certain dataset configuration files in a profile</td>
<td>Hiding Dataset Components</td>
</tr>
<tr>
<td>that you cannot or do not want to modify</td>
<td></td>
</tr>
<tr>
<td>Reprocess your dataset</td>
<td>Reprocessing and Retransformation</td>
</tr>
</tbody>
</table>
Log Processing Configuration File

Information about the LogProcessing.cfg file and how its parameters affect the log processing phase of the dataset construction process.

About the Log Processing Configuration File

The LogProcessing.cfg file controls the log processing phase of dataset construction, during which unordered data is read from the data sources (referred to as log sources), and filters and transformations are applied to the data.

You must edit the LogProcessing.cfg file to perform any of the following dataset configuration tasks:

- Specifying log sources. See Log Sources.
- Determining which log entries enter the dataset during log processing. See Data Filters and Log Entry Condition.
- Enabling the splitting of tracking IDs with large amounts of event data. See Key Splitting.
- Configuring a data workbench server to run as a file server unit. See Configuring a Data Workbench Server File Server Unit.
- Configuring a data workbench server to run as a centralized normalization server. See Configuring a Data Workbench Server File Server Unit.

Note: Log Processing Dataset Include files can contain additional instructions for the log processing phase of dataset construction. These files exist within the Dataset\Log Processing directory for any inherited profile. They typically define application-specific parameters (such as web-specific configuration parameters for the Site application). For information about Log Processing Dataset Include files, see Dataset Include Files. For information about web-specific configuration parameters for Site, see Configuration Settings for Web Data.

Editing the Log Processing Configuration File

Steps to edit the LogProcessing.cfg file for a dataset profile.

1. While working in your dataset profile, open the Profile Manager and click Dataset to show its contents.
   For information about opening and working with the Profile Manager, see the Data Workbench User Guide.

   Note: A Log Processing subdirectory may exist within the Dataset directory. This subdirectory contains the Log Processing Dataset Include files that have been created for one or more inherited profiles. See Dataset Include Files.

2. Right-click the check mark next to LogProcessing.cfg and click Make Local. A check mark for this file appears in the User column.

3. Right-click the newly created check mark and click Open > in Workstation. The LogProcessing.cfg window appears.

   You also can open the LogProcessing.cfg file from a Transformation Dependency Map. For information about transformation dependency maps, see Dataset Configuration Tools.

4. Edit the parameters in the configuration file using the following table as a guide.

   When editing the LogProcessing.cfg file within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z...
Note: A Log Processing Dataset Include file for an inherited profile contains a subset of the parameters described in the following table as well as some additional parameters. See Dataset Include Files.

### Log Processing.cfg

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Sources</td>
<td>The sources of data. See Log Sources.</td>
</tr>
</tbody>
</table>
| End Time                | Optional. Filter data to include log entries with timestamps up to but not including this time. Adobe recommends using one of the following formats for the time:  
  • January 1 2013 HH:MM:SS EDT  
  • Jan 1 2013 HH:MM:SS GMT  
  For example, specifying July 29 2013 00:00:00 EDT as the End Time includes data through July 28, 2013, at 11:59:59 PM EDT. See Data Filters.  
  You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by the data workbench server, see Time Zone Codes.  
  Note: The Use Start/End Times parameter for Sensor, log file, and XML sources is related to this parameter. See the sections of Log Sources that discuss these source types. |
| Fields                  | Optional. Adobe recommends defining Fields in one or more Log Processing Dataset Include files. See Log Processing Dataset Include Files.                                                                          |
| Group Maximum Key Bytes | Maximum amount of event data that the Server can process for a single tracking ID. Data exceeding this limit is filtered from the dataset construction process. This value must be set to 2e6 when key splitting is active and 1e6 when key splitting is not active. See Key Splitting.  
  Note: Do not change this value without consulting Adobe.                                                                                                                                                    |
| Hash Threshold          | Optional. A sampling factor for random sub-sampling of rows. If set to a number n, then only one out of each n tracking IDs enters the dataset, reducing the total number of rows in the dataset by a factor of n.  
  To create a dataset that requires 100 percent accuracy (that is, to include all rows), you would set Hash Threshold to 1.  
  Values:  
  Hash Threshold = 1 (100 percent of data including all rows.)  
  Hash Threshold = 2 (1/2 of data and includes half the rows.) |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Threshold = 3</td>
<td>(1/3 of data and includes one of three rows, but rounds to 34% in Query Completion)</td>
</tr>
<tr>
<td>Hash Threshold = 4</td>
<td>(1/4th of data and includes one out of four rows.)</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Using a Hash Threshold = 8 provides 1/8th of the data, which is 12.5%. However the Query Completion value in the rounds to 13% for this value. Additional examples include a Hash Threshold = 6 that results in 17% query resolution. A Hash Threshold = 13 results in 8% query resolution.</td>
</tr>
<tr>
<td></td>
<td>If Hash Threshold is specified in both the Log Processing.cfg and Transformation.cfg files, it is not applied in sequence; the maximum value set in either configuration file applies. See Data Filters.</td>
</tr>
<tr>
<td>Log Entry Condition</td>
<td>Optional. Defines the rules by which log entries are considered for inclusion in the dataset. See Log Entry Condition.</td>
</tr>
<tr>
<td>Reprocess</td>
<td>Optional. Any character or combination of characters can be entered here. Changing this parameter and saving the file to the data workbench Server machine initiates data reprocessing. See Reprocessing and Retransformation.</td>
</tr>
<tr>
<td>Split Key Bucket Space</td>
<td>Parameter involved in key splitting. Its value should be 6e6 when key splitting is active. See Key Splitting.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not change this value without consulting Adobe.</td>
</tr>
<tr>
<td>Split Key Bytes</td>
<td>Parameter involved in key splitting. Its value should be 1e6 when key splitting is active and 0 when key splitting is not active. See Key Splitting.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not change this value without consulting Adobe.</td>
</tr>
<tr>
<td>Split Key Space Ratio</td>
<td>Parameter involved in key splitting. Its value should be 10 when key splitting is active. See Key Splitting.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not change this value without consulting Adobe.</td>
</tr>
<tr>
<td>Stages</td>
<td>Optional. The names of the processing stages that can be used in Log Processing Dataset Include files. Processing stages provide a way to order the transformations that are defined in Log Processing Dataset Include files. This parameter is very helpful if you have defined one or more transformations within multiple Log Processing Dataset Include files and you want specific transformations to be performed at specific points during log processing.</td>
</tr>
</tbody>
</table>
The order in which you list the stages here determines the order in which the transformations in the **Log Processing Dataset Include** files are executed during log processing. Preprocessing and Postprocessing are built-in stages. Preprocessing is always the first stage, and Postprocessing is always the last stage. By default, there is one named stage called Default.

**To add a new processing stage**

- In the **Log Processing.cfg** window, right-click **Stages** and click **Add New > Stage**.
- Enter a name for the new stage.

**To delete an existing processing stage**

- Right-click the number corresponding to the stage that you want to delete and click **Remove<#stage_number>**.

*Note: When you specify a **Stage** in a **Log Processing Dataset Include** files, the name of the stage must match exactly the name that you enter here. See **Dataset Include Files**.*

### Start Time

Optional. Filter data to include log entries with timestamps at or after this time. Adobe recommends using one of the following formats for the time:

- January 1 2013 HH:MM:SS EDT
- Jan 1 2013 HH:MM:SS GMT

For example, specifying "July 29 2013 00:00:00 EDT" as the Start Time includes data starting from July 29, 2013, at 12:00:00 AM EDT. See **Data Filters**.

You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by the data workbench server, see **Time Zone Codes**.

*Note: The **Use Start/End Times parameter for Sensor**, log file, and XML sources is related to this parameter. See the sections of **Log Sources** that discuss these source types.*

### Time Zone

Optional. Time zone of the the data workbench server that is used for time conversions (such as the conversion represented by the x-local-timestring field) during log processing.

*Note: You must specify the Time Zone if you want to access the converted time field during the log processing phase of dataset construction. Otherwise, the data workbench server records an error in the event logs.*

See **Time Zones**.

### Transformations

Optional. Adobe recommends defining transformations for log processing in one or more **Log Processing Dataset Include** files. See **Log Processing Dataset Include Files**.
5. Right-click (modified) at the top of the window and click Save.

6. In the Profile Manager, right-click the check mark for Log Processing.cfg in the User column, then click Save to > <dataset profile name> to make the locally made changes take effect. Reprocessing of the data begins after synchronization of the dataset profile.

   Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

   For more information about reprocessing your data, see Reprocessing and Retransformation.

Log Sources

Log sources are files that contain the data to be used to build a dataset.

The data available in the log sources is called event data because each data record represents a transaction record or a single instance of an event. The data workbench server can process log sources that are derived from data collected by Sensors or extracted from other data sources.

• Data Collected by Sensors: Data collected by Sensors from HTTP and application servers is transmitted to data workbench servers, which convert the data into highly compressed log (.vsl) files. See Sensor Files.

• Data Extracted by Insight Server: The data workbench server reads event data contained in flat files, XML files, or ODBC-compliant databases, and uses its decoders to extract the desired elements of the data. Such event data does not have to be memory-resident, but the records that contain the data must include a tracking ID. See Log Files, XML Log Sources, and ODBC Data Sources.

To add a log source

1. Open Log Processing.cfg in data workbench.
2. Right-click Log Sources, then click Add New.
3. Select one of the following:
   • Sensor
   • Log File
   • XML Log Source
   • ODBC Data Source

4. The specific parameters used to define a dataset vary based on the type of log source to be used in the dataset's configuration process. Specify the parameters as indicated in the section corresponding to the appropriate log source:
   • Sensor Files
   • Log Files
   • XML Log Sources
   • ODBC Data Sources

5. After you have defined your log source (and made changes to other parameters) in the Log Processing.cfg file, save the file locally and save it to your dataset profile on the data workbench server.
Note: A data workbench server File Server Unit can receive and store Sensor files, log files, and XML files and serve them to the data workbench server's Data Processing Units that construct the dataset. See Configuring a Data Workbench Server File Server Unit.

You can open the configuration of any log source from a Transformation Dependency Map. For information about Transformation Dependency Map, see Dataset Configuration Tools.

Sensor Files

Information about the requirements and parameters of Sensor files.

- Requirements
- Parameters

Requirements

Event data collected by Sensors from HTTP and application servers is transmitted to data workbench servers, which convert the data into highly compressed log (.vsl) files. The .vsl file format is managed by the data workbench server, and each file has a name of the format:

YYYYMMDD-SENSORID.VSL

where YYYYMMDD is the date of the file, and SENSORID is the name (assigned by your organization) that indicates which Sensor collected and transmitted the data to the data workbench server.

Parameters

For Sensor files, the following parameters are available:

Log Processing.cfg: Sensor Files

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Paths</td>
<td>The directories where the .vsl files are stored. The default location is the Logs directory. A relative path refers to the installation directory of the data workbench server. You can use wildcard characters to specify which .vsl files to process: • * matches any number of characters • ? matches a single character For example, the log path Logs*.vsl matches any file in the Logs directory ending in .vsl. The log path Logs*-SENSOR?.vsl matches files in the Logs directory with any date (YYYYMMDD) and a single character after SENSOR, as in SENSOR1. If you want to search all subdirectories of the specified path, you must set the Recursive parameter to true.</td>
</tr>
</tbody>
</table>

Note: If the files are to be read from a data workbench server's File Server Unit, then you must enter the appropriate URI(s) in the Log Paths parameter. For example, the URI /Logs/*-*.*.vsl matches any .vsl file in the Logs directory. See Configuring a Data Workbench Server File Server Unit.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Server</td>
<td>Information (Address, Name, Port, and so on) necessary to connect to a file server. If there is an entry in the Log Server parameter, the Log Paths are interpreted as URIs. Otherwise, they are interpreted as local paths. See Configuring a Data Workbench Server File Server Unit.</td>
</tr>
<tr>
<td>Log Source ID</td>
<td>This parameter's value can be any string. If a value is specified, this parameter enables you to differentiate log entries from different log sources for source identification or targeted processing. The x-log-source-id field is populated with a value identifying the log source for each log entry. For example, if you want to identify log entries from a Sensor named VSensor01, you could type from VSensor01, and that string would be passed to the x-log-source-id field for every log entry from that source. For information about the x-log-source-id field, see Event Data Record Fields.</td>
</tr>
<tr>
<td>Recursive</td>
<td>True or false. If set to true, all subdirectories of each path specified in Log Paths are searched for files matching the specified file name or wildcard pattern. The default value is false.</td>
</tr>
</tbody>
</table>
| Use Start/End Times | True or false. If set to true and Start Time or End Time is specified, then all files for this log source must have file names starting with dates in ISO format (YYYYMMDD). It is assumed that each file contains data for one GMT day (for example, the time range starting at 0000 GMT on one day and ending at 0000 GMT the following day). If the log sources files contain data that do not correspond to a GMT day, then this parameter must be set to false to avoid incorrect results.  

**Note:** By default, .vsl files containing data collected by Sensor automatically meet the naming and time range requirements described above. If you set this parameter to true, the data workbench server always processes data from files whose names include ISO dates that fall between the specified Start Time and End Time. If you set this parameter to false, the data workbench server reads all of the .vsl files during log processing to determine which files contain data within the Start Time and End Time range. For information about the Start Time and End Time parameters, see Data Filters. |

**Note:** Do not use the configuration parameters for Sensor data sources to determine which log entries within a log file should be included in a dataset. Instead, set up the data source to point to all of the log files within a directory. Then use the Start Time and End Time parameters of Log Processing.cfg to determine which log entries should be used in constructing the dataset. See Data Filters.

**Log Files**

Information about integrating event data from flat files that are not .vsl files.

The file containing the event data must meet the following requirements:

- Each event data record in the file must be represented by one line.
• The fields within a record must be separated, whether empty or not, by an ASCII delimiter. The data workbench server does not require you to use a specific delimiter. You may use any character that is not a line-ending character and does not appear anywhere within the event data itself.
• Each record in the file must contain:
  • A tracking ID
  • A time stamp
• To specify start and end times for data processing, each file name must be of the form:
  • `YYYYMMDD-SOURCE.log`

where `YYYYMMDD` is the Greenwich Mean Time (GMT) day of all of the data in the file, and `SOURCE` is a variable identifying the source of the data contained in the file.

**Note:** Please contact Adobe Consulting Services for a review of the log files that you plan to incorporate into the dataset.

Parameters
For log files log sources, the parameters in the following table are available.

**Note:** The processing of log file log sources requires additional parameters that are defined in a Log Processing Dataset Include file, which contains a subset of the parameters included in a Log Processing.cfg file as well as special parameters for defining decoders for extracting data from the log file. For information about defining decoders for log file log sources, see Text File Decoder Groups.

**Log Processing.cfg: Log Files**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The identifier for the log file source.</td>
</tr>
</tbody>
</table>
| Log Paths   | The directories where the log files are stored. The default location is the Logs directory. A relative path refers to the installation directory of the data workbench server. You can use wildcard characters to specify which log files to process:  
  • * matches any number of characters.  
  • ? matches a single character.  
  For example, the log path `Logs\*.log` matches any file in the Logs directory ending in `.log`.  
  If you want to search all subdirectories of the specified path, then you must set the Recursive parameter to true.  
  If the files are to be read from a data workbench server’s File Server Unit, then you must enter the appropriate URI(s) in the Log Paths parameter. For example, the URI/Logs/* .log matches any .log file in the Logs directory. See Configuring a Data Workbench Server File Server Unit. |
<p>| Log Server  | Information (Address, Name, Port, and so on) necessary to connect to a file server. If there is an entry in the Log Server parameter, the Log Paths are interpreted as URIs. Otherwise, they are interpreted as local paths. See Configuring a Data Workbench Server File Server Unit. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed</td>
<td>True or false. This value should be set to true if the log files to be read by the data workbench server are compressed gzip files.</td>
</tr>
<tr>
<td>Decoder Group</td>
<td>The name of the text file decoder group to be applied to the log file log source. This name must match exactly the name of the corresponding text file decoder group specified in the Log Processing Dataset Include file. See Text File Decoder Groups.</td>
</tr>
<tr>
<td>Log Source ID</td>
<td>This parameter's value can be any string. If a value is specified, this parameter enables you to differentiate log entries from different log sources for source identification or targeted processing. The x-log-source-id field is populated with a value identifying the log source for each log entry. For example, if you want to identify log entries from a log file source named LogFile01, you could type from LogFile01, and that string would be passed to the x-log-source-id field for every log entry from that source. For information about the x-log-source-id field, see Event Data Record Fields.</td>
</tr>
<tr>
<td>Mask Pattern</td>
<td>A regular expression with a single capturing subpattern that extracts a consistent name used to identify the source of a series of log files. Only the file name is considered. The path and extension are not considered for the regular expression matching. If you do not specify a mask pattern, then a mask is generated automatically. For the files Logs\010105server1.log and Logs\010105server2.log, the mask pattern would be [0-9]{6}(.*). This pattern extracts the string &quot;server1&quot; or &quot;server2&quot; from the file names above. See Regular Expressions.</td>
</tr>
<tr>
<td>Recursive</td>
<td>True or false. If this parameter is set to true, all subdirectories of each path specified in Log Paths are searched for files matching the specified file name or wildcard pattern. The default value is false.</td>
</tr>
<tr>
<td>Reject File</td>
<td>The path and file name of the file containing the log entries that do not meet the conditions of the decoder.</td>
</tr>
<tr>
<td>Use Start/End Times</td>
<td>True or false. If this parameter is set to true and Start Time or End Time is specified, then all files for this log source must have file names starting with dates in ISO format (YYYYMMDD). It is assumed that each file contains data for one GMT day (for example, the time range starting at 0000 GMT on one day and ending at 0000 GMT the following day). If the log sources file names do not begin with ISO dates, or if the files contain data that do not correspond to a GMT day, then this parameter must be set to false to avoid incorrect results. For information about the Start Time and End Time parameters, see Data Filters.</td>
</tr>
</tbody>
</table>

**Note:** If the naming and time range requirements described above are satisfied for the log files and you set this parameter to true, the specified text file decoder group limits the files read to those whose names have ISO dates that fall between the specified Start Time and End Time. If you set this parameter to false, the data workbench server reads all of the log files during log processing to determine which files contain data within the Start Time and End Time range.
In this example, the dataset is constructed from two types of log sources.

Log Source 0 specifies log files generated from event data captured by Sensor. This data source points to a directory called Logs and to all of the files in that directory with a .vsl file name extension.

Log Source 1 points to all of the files in the Logs directory with a .txt file name extension. The decoder group for this log source is called “Text Logs.”

XML Log Sources

Information on integrating event data from XML log files.

The file containing the event data must meet the following requirements:

• Event data must be included in a properly formatted XML file with appropriate parent-child relationships.
• A unique decoder group must exist for each XML file format. For information about constructing a decoder group, see XML Decoder Groups.
• Each visitor record in the file must contain:
  • A tracking ID
  • A time stamp
• To specify start and end times for data processing, each file name must be of the form
  YYYYYMDD-SOURCE.log
where YYYYMMDD is the Greenwich Mean Time (GMT) day of all of the data in the file, and SOURCE is a variable identifying the source of the data contained in the file.

For an example of an XML file that meets these requirements, see XML Decoder Groups.

⚠️ **Note:** Please contact Adobe Consulting Services for a review of the XML log files that you plan to incorporate into the dataset.

### Parameters

For XML log sources, the parameters in the following table are available.

⚠️ **Note:** The processing of XML log sources requires additional parameters that are defined in a Log Processing Dataset Include file, which contains a subset of the parameters included in a Log Processing.cfg file as well as special parameters for defining decoders for extracting data from the XML file. For information about defining decoders for XML log sources, see XML Decoder Groups.

#### Log Processing.cfg: XML Log Sources

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The identifier for the XML log source.</td>
</tr>
</tbody>
</table>
| Log Paths        | The directories where the XML log sources are stored. The default location is the Logs directory. A relative path refers to the installation directory of the data workbench server.  
You can use wildcard characters to specify which XML log sources to process:  
• * matches any number of characters  
• ? matches a single character  
For example, the log path Logs/*.xml matches any file in the Logs directory ending in .xml.  
If you want to search all subdirectories of the specified path, you must set the Recursive field to true.  
⚠️ **Note:** If the files are to be read from a data workbench server’s File Server Unit, you must enter the appropriate URI(s) in the Log Paths field. For example, the URI/Logs/*.xml matches any .xml file in the Logs directory. See Configuring a Data Workbench Server File Server Unit. |
<p>| Log Server       | Information (Address, Name, Port, and so on) necessary to connect to a file server. If there is an entry in the Log Server field, the Log Paths are interpreted as URLs. Otherwise, they are interpreted as local paths. See Configuring a Data Workbench Server File Server Unit. |
| Compressed       | True or false. This value should be set to true if the XML log sources to be read by the data workbench server are compressed gzip files.                                                                 |
| Decoder Group    | The name of the XML decoder group to be applied to the XML log source. This name must match exactly the name of the corresponding XML decoder group specified in the Log Processing Dataset Include file. See XML Decoder Groups. |</p>
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Source ID</td>
<td>This field's value can be any string. If a value is specified, this field enables you to differentiate log entries from different log sources for source identification or targeted processing. The x-log-source-id field is populated with a value identifying the log source for each log entry. For example, if you want to identify log entries from a log file source named XMLFile01, you could type from XMLFile01, and that string would be passed to the x-log-source-id field for every log entry from that source. For information about the x-log-source-id field, see Event Data Record Fields.</td>
</tr>
<tr>
<td>Mask Pattern</td>
<td>A regular expression with a single capturing subpattern that extracts a consistent name used to identify the source of a series of log files. Only the file name is considered. The path and extension are not considered for the regular expression matching. If you do not specify a mask pattern, then a mask is generated automatically. For the files Logs\010105server1.xml and Logs\010105server2.xml, the mask pattern would be [0-9]{6}(.*). This pattern extracts the string &quot;server1&quot; or &quot;server2&quot; from the file names above. See Regular Expressions.</td>
</tr>
<tr>
<td>Recursive</td>
<td>True or false. If this parameter is set to true, all subdirectories of each path specified in Log Paths are searched for files matching the specified file name or wildcard pattern. The default value is false.</td>
</tr>
<tr>
<td>Reject File</td>
<td>The path and file name of the file containing the log entries that do not meet the conditions of the decoder.</td>
</tr>
<tr>
<td>Use Start/End Times</td>
<td>True or false. If this parameter is set to true and Start Time or End Time is specified, then all files for this log source must have file names starting with dates in ISO format (YYYYMMDD). It is assumed that each file contains data for one GMT day (for example, the time range starting at 0000 GMT on one day and ending at 0000 GMT the following day). If the log sources file names do not begin with ISO dates, or if the files contain data that do not correspond to a GMT day, then this parameter must be set to false to avoid incorrect results. <strong>Note:</strong> If the naming and time range requirements described above are satisfied for the XML files and you set this parameter to true, the specified XML decoder group limits the files read to those whose names have ISO dates that fall between the specified Start Time and End Time. If you set this parameter to false, the data workbench server reads all of the XML files during log processing to determine which files contain data within the Start Time and End Time range. For information about the Start Time and End Time parameters, see Data Filters.</td>
</tr>
</tbody>
</table>

**Note:** You should not delete or move XML log sources after the data sources for a dataset have been defined. Only newly created XML files should be added to the directory for the data sources.
Avro Data Feed

The Apache Avro data feed is a new format used by Data Workbench to more efficiently integrate data and provide access to new fields with the updated evars, custom events, and solution variables used in Adobe Analytics Premium.

The Avro data feed provides a more efficient way to integrate data into Data Workbench:

- Avro provides a single-source format for traffic and commerce data.
- The Avro feed is compressed data of multiple source chunks provided per day. It provisions only populated fields and provides monitoring and notification features, access to historical data, and auto-recovery.
- The schema, a self-defining layout of Avro log files, is included at the beginning of each file.
- New fields are added with supporting information to ingest Data Workbench data without any changes required to the decoder. These include:
  - Evars: 1-250 (previously 1-75)
  - Custom Events: 1-1000 (versus 1-100)
  - Access to solution variables for mobile, social, and video data

💡 Note: In addition, using the Avro feed allows immediate access to any new fields in the feed without a shutdown, allowing the fields to be updated with no service hour requirements.

The Avro data feed is set up in separate files:

- An **Avro Log file**: This is the Avro log format generated from the decoder to format traffic and commerce data.
- An **Avro Decoder file**: This file lets you map values into the new Avro format. You can set up the decoder using the Avro Decoder Wizard.

Avro Decoder Wizard

This wizard sets up the Avro decoder log file.

To open, right-click in a workspace and select **Admin > Wizards > Avro Decoder Wizard**.

**Step 1: Select an Avro Log File.**

In this step, you can select a source file for the Avro schema. Schemas can be accessed from a log file (.log) or an existing decoder file (.avro). Schemas can be pulled from either file.

<table>
<thead>
<tr>
<th>Avro Log File</th>
<th>Click to open a log (.log) file to view the schema at the top of the log file and generate decoder file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avro Decoder File</td>
<td>Click to open and edit the schema of an existing decoder (.avro) file.</td>
</tr>
</tbody>
</table>

**Step 2: Select Input Fields.**

Select the input fields to be used in the data set to pass through log processing. All fields in the file will be displayed, allowing you to select fields for the feed.

<table>
<thead>
<tr>
<th>Select Defaults</th>
<th>Select fields to identify as a standard set of default fields.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select All</td>
<td>Select all fields in the file.</td>
</tr>
<tr>
<td>Deselect All</td>
<td>Clear all fields in the file.</td>
</tr>
</tbody>
</table>
Note: A \textit{x-product} (Generates row) field is provided if an array is encountered in the data. This field generates new rows for the nested data in an array as input fields. For example, if you have a Hit row with many Product values in an array, then rows will be generated in the input file for each product.

Step 3: Select fields that get copied to generate rows.

Because new rows can be created from nested values in an array, every new row created must have a tracking ID and a timestamp. This step allows you to select the fields to be copied to rows from the parent record, such as a tracking ID and timestamp. You can also select other values you want added to each row.

<table>
<thead>
<tr>
<th>Select Defaults</th>
<th>Select a standard set of default fields that require new column values added to each row, such as a tracking ID and timestamp. For example, a \texttt{hit_source} field is a default value required to be added to each new row (it is defined as a default value in the list). You can add other column values to each row as needed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select All</td>
<td>Select all fields in the file.</td>
</tr>
<tr>
<td>Deselect All</td>
<td>Clear all fields in the file.</td>
</tr>
</tbody>
</table>

Use the \textbf{Search} box to find values in the list.

Step 4: Specify the decoder name

Assign a name for the group of fields and save as a decoder file. The name should match the Decoder group name specified in your log source.

Step 5: Save the decoder file.

The file menu will open to name the decoder file and save as a .cfg file in the \textbf{Logs} folder.

Configuring a Data Workbench Server File Server Unit

Information about Insight Server file server units and the file server configuration process.

About Data Workbench Server File Server Units

Information about the data workbench server FSU.

You can configure the data workbench server (InsightServer64.exe) to run as a File Server Unit (FSU) by completing the parameters in the Log Sources > Log Server node of the Log Processing.cfg file. When the data workbench server is configured to run as an FSU, it stores source files (.vsl files, text files, or XML files) that can be accessed quickly by multiple processing servers (DPUs). When the DPUs in a data workbench server cluster accesses the FSU to read the log files, they divide the log files among them and guarantee that the same file is not processed more than once.

\textbf{Note:} When setting up an FSU that serves a data workbench server cluster consisting of five to ten DPUs, you should make the cluster’s master server the FSU.

For information about installing a data workbench server cluster, see the \textit{Server Products Installation and Administration Guide}. 
The File Server Configuration Process

The Log Processing.cfg file specifies the location of a dataset's log sources.

If the location is a remote location, the data workbench server machine that is processing the data connects to the designated remote machine to read the logs.

On the data workbench server machine designated to run as an FSU, the Access Control.cfg file lets the DPUs connect to the FSU, and the Communications.cfg file maps the location of the remote data files. The process steps to configure an FSU are as follows:

1. In the Log Processing.cfg file on your master data workbench server, specify the type of data source and the location of the source. See Specifying the Data Source.
2. In the Access Control.cfg file on the FSU, edit the permissions to allow the DPUs to connect to the FSU to read the log data. See Editing the Permissions on the File Server Unit.
3. In the Communications.cfg file on the FSU, edit the settings for the LoggingServer and FileServer entries to specify the location of the log files. See Specifying the Location of the Log Files.
4. If you are configuring your dataset profile to run on a data workbench server cluster, you also must make the cluster's FSU the server where all of the profile's dimensions are constructed:
   (For data workbench server clusters only) In the Communications.cfg and cluster.cfg files on the FSU, add entries for a "normalize server" to make the FSU the server within the cluster where all dimensions are constructed. See Creating a Centralized Normalization Server for a Cluster.

For instructions to configure a dataset profile to be processed by a data workbench server cluster, see the Server Products Installation and Administration Guide.

Note: The following instructions assume that all of the log files reside in the default directory. If you want to store logs in another directory or create multiple log paths, contact Adobe Consulting Services to discuss your specific configuration.

This section discusses the following topics:

• Specifying the Data Source
• Editing the Permissions on the File Server Unit
• Specifying the Location of the Log Files
• Creating a Centralized Normalization Server for a Cluster

Specifying the Data Source

When specifying remote data sources for a dataset, you must specify the type of data source and the location of the log files on your master data workbench server.

To specify the data source and its location

2. Add a Sensor, log file, or XML data source. See To add a log source.
3. Complete the Log Paths parameter. See Sensor Files, Log Files, or XML Log Sources. Be sure to specify a valid URI.
4. Complete the Log Server parameters defined in the following table:

Log Server Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name identifying the remote file server.</td>
</tr>
<tr>
<td>SSL Server Common Name</td>
<td><strong>Server Common Name</strong> listed on the file server's SSL certificate. This parameter is optional if <strong>Use SSL</strong> is set to false.</td>
</tr>
<tr>
<td>Address</td>
<td>Address of the file server machine. Can be left blank if <strong>Name</strong> matches <strong>SSL Server Common Name</strong>. For example: <code>visual.mycompany.com</code> or <code>192.168.1.90</code>.</td>
</tr>
<tr>
<td>Port</td>
<td>Port through which the data workbench server machine communicates with the file server.</td>
</tr>
<tr>
<td>SSL Client Certificate</td>
<td>Name of the <strong>SSL certificate</strong> file for the data workbench server (<code>server_cert.pem</code>).</td>
</tr>
<tr>
<td>Use SSL</td>
<td>True or false. True indicates that the file server uses <strong>SSL</strong>.</td>
</tr>
</tbody>
</table>

If a proxy server is required for the DPUs to connect to the FSU, you need to complete the following parameters:

**Log Server Parameters for Proxy Servers**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy Address</td>
<td>The address of a proxy server that the data workbench server must use to access the file server.</td>
</tr>
<tr>
<td>Proxy Password</td>
<td>Optional. The password to the proxy server.</td>
</tr>
<tr>
<td>Proxy Port</td>
<td>The port of the proxy server. The default is 8080.</td>
</tr>
<tr>
<td>Proxy User Name</td>
<td>Optional. The user name for the proxy server.</td>
</tr>
</tbody>
</table>

Following is an example of a defined **Log Server** in the **Log Processing.cfg** file. **Log Source #1** is a LogFile source that points to a directory called Logs (note the URI specified in the Log Paths parameter) on the machine named FSU01.
Editing the Permissions on the File Server Unit

In the previous process, you configured a profile for a given dataset to read log files from an FSU. Now you must edit the permissions on the FSU to allow connections from the DPUs that are running the profile. The following steps walk you through editing the permissions file `Access Control.cfg`.

**To edit permissions on the FSU**

1. Open the **Server Files Manager** for the data workbench server machine that you are setting up as your FSU and click **Access Control** to show its contents.

   For information about opening and working with the **Server Files Manager**, see the **Data Workbench User Guide**.

2. In the **Server Files Manager** window, click **Access Control** to show its contents. The **Access Control.cfg** file is located within this directory.

3. Right-click the check mark in the server name column for **Access Control.cfg**, then click **Make Local**. A check mark appears in the **Temp** column for **Access Control.cfg**.

4. Right-click the newly created check mark under the **Temp** column and click **Open > in Workstation**.

5. In the **Access Control** window, click **Access Control Groups** to show its contents.

6. Right-click the numeric label for the final **AccessGroup** in the list and click **Add new > Group**.

7. Enter a **Name** for the new **AccessGroup**. Example: Connecting Servers.

8. Right-click **Member** under the new **AccessGroup**, then click **Add new > Member**.

9. Enter the IP address for the data workbench server’s DPU that connects to this file server.

10. Repeat steps 4 and 5 for any other data workbench server DPUs that connect to this FSU, including the data workbench server DPUs in a cluster that must access the log files.

11. Right-click **Read-Only Access** under the new **AccessGroup**, then click **Add new > URI**.
12. Enter the location of the stored log files on the file server machine. Use forward slashes (/) in the path specification. The default location is /Logs/.

13. Right-click (modified) at the top of the window, then click Save.

14. In the Server Files Manager window, right-click the check mark for Access Control.cfg in the Temp column, then click Save to > <server name> to save the locally made changes to the data workbench server's FSU.

Specifying the Location of the Log Files

You must edit the Communications.cfg file on the FSU to specify the location of the log files.

To specify the location of the log files

1. In the Server Files Manager window, click Components to show its contents. The Communications.cfg file is located within this directory.

2. Right-click the check mark in the server name column for Communications.cfg, then click Make Local. A check mark appears in the Temp column for Communications.cfg.

3. Right-click the newly created check mark under the Temp column and click Open > in Workstation.

4. In the Communications.cfg window, click component to show its contents.

5. In the Communications.cfg window, click Servers to show its contents. Several servers may appear: File Servers, Logging Servers, Init Servers, Status Servers, Send Servers, or Replicate Servers.

6. (For Sensor log sources only) Find the LoggingServer, which is where Sensor writes its log files to be processed by the data workbench server, then click its number to view the menu. Edit the Log Directory parameter to reflect the desired location of the log files. The default log directory is the Logs folder within the data workbench server's installation directory.

Do not modify any other parameters for the LoggingServer.

```
-Communications.cfg
  -component CommServer
    -Servers
      +0 InitServer
      -1 LoggingServer
      Flush Interval 10000
      Log Directory Logs\%
      URI /Submit.vsp
      +2 SendServer
      +3 FileServer
      +4 FileServer
      +5 FileServer
```

7. Find the FileServer that specifies the location of log files. There may be several File Servers listed under Servers, so you may need to view the contents for many of them (by clicking the server number) to find the desired server.

8. Edit the Local Path and URI parameters for the FileServer to reflect the location of the log files. The following example shows that the log files reside in the Logs folder within the data workbench server's installation directory:
Log Processing Configuration File

Creating a Centralized Normalization Server for a Cluster

If you are configuring your dataset profile to run on a data workbench server cluster, then you should make the cluster's FSU the server where all of the profile's dimensions are constructed.

Adobe strongly recommends that the cluster's FSU serves as the cluster's master server and its centralized normalization server.

To make the FSU the centralized normalization server, you must open and edit the `Communications.cfg` and `Cluster.cfg` files on the FSU.

To make the FSU the centralized normalization server

1. Add a `NormalizeServer` entry to the `Communications.cfg` file on the FSU.

   Note: If you have installed the complete release package for data workbench server v5.0 or later, the `Communications.cfg` file on your FSU should have a `NormalizeServer` entry already. You can follow the steps below to confirm that the entry exists.

   a. Open the `Communications.cfg` file in data workbench as described in Specifying the Location of the Log Files.
   b. Click component to show its contents.
   c. Right-click Servers and click Add New > Centralized Normalization Server.
d. In the URI parameter for the `NormalizeServer`, type `/Cluster/`. 

```
<table>
<thead>
<tr>
<th>Servers</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>InitServer</td>
</tr>
<tr>
<td>1</td>
<td>LoggingServer</td>
</tr>
<tr>
<td>2</td>
<td>SendServer</td>
</tr>
<tr>
<td>3</td>
<td>FileServer</td>
</tr>
<tr>
<td>4</td>
<td>FileServer</td>
</tr>
<tr>
<td>5</td>
<td>FileServer</td>
</tr>
<tr>
<td>6</td>
<td>FileServer</td>
</tr>
<tr>
<td>7</td>
<td>FileServer</td>
</tr>
<tr>
<td>8</td>
<td>FileServer</td>
</tr>
<tr>
<td>9</td>
<td>FileServer</td>
</tr>
<tr>
<td>10</td>
<td>StatusServer</td>
</tr>
<tr>
<td>11</td>
<td>FileServer</td>
</tr>
<tr>
<td>12</td>
<td>FileServer</td>
</tr>
<tr>
<td>13</td>
<td>FileServer</td>
</tr>
<tr>
<td>14</td>
<td>FileServer</td>
</tr>
<tr>
<td>15</td>
<td>FileServer</td>
</tr>
<tr>
<td>16</td>
<td>FileServer</td>
</tr>
<tr>
<td>-17</td>
<td>NormalizeServer</td>
</tr>
</tbody>
</table>

URI: `/Cluster/`
```

e. Right-click (modified) at the top of the window, and click `Save`.

f. In the `Server Files Manager` window, right-click the check mark for `Communications.cfg` in the `Temp` column, then click `Save to > <server>` name to save the locally made changes to the data workbench server FSU.

2. Define the centralized normalization server in the `Cluster.cfg` file on the master server in your data workbench Server cluster.

   **Note:** If the FSU on which you are setting up your centralized normalization server is not the master data workbench Server in your cluster, you must add the IP addresses of the DPUs in the cluster to the `Cluster Servers` access group in the FSU's `Access Control.cfg` file. For instructions to add servers to the `Cluster Servers` group, see Updating the Access Control File for a Cluster section in the Server Products Installation and Administration Guide.

   a. Open the `Profile Manager` within your dataset profile, then click `Dataset` to show its contents. The `Cluster.cfg` file is located within this directory.
   b. Right-click the check mark next to `Cluster.cfg`, then click `Make Local`. A check mark for this file appears in the `User` column.
   c. Right-click the newly created check mark and click `Open > in Notepad`.
   d. Add the text that is highlighted in the following file fragment:

```
Cluster = ClusterConfig:
```

Log Processing Configuration File

---

30
Normalize Server = serverInfo:
Address = string:
Port = int: 80
SSL Server Common Name = string: server common name
Use SSL = bool: false

Note: When you enter the common name of FSU for the SSL Server Common Name parameter, the FSU uses its .address file to resolve the common name. For information about the .address file, see the Server Products Installation and Administration Guide.

e. Save the file.
f. In the Profile Manager, right-click the check mark for Cluster.cfg in the User column, then click Save to > <dataset profile name> to save the locally made changes to the dataset profile.

Information About Specific Log Processing Parameters

Links to additional information about specific parameters in the Log Processing.cfg file.

• Data Filters  
• Log Entry Condition  
• Key Splitting

Data Filters

Data filters in the Log Processing.cfg file limit the log entries that are considered in the dataset construction process.

The filters defined in the Log Processing.cfg file include the following:

• End Time  
• Hash Threshold  
• Start Time

The filtering defined by these parameters occurs after log entries leave the decoders and after transformations but before their evaluation by the Log Entry Condition. See Log Entry Condition. In general, changing any of these parameters results in changes to the composition of the dataset.

The recommended technique for using Sensor data sources to construct a dataset that covers a specific period of time is to use the Start Time and End Time parameters for the dataset.

Using the Start Time and End Time parameters is preferred to other techniques, such as moving log files to separate them by directory. By setting the start and end times for the dataset, the data workbench server automatically uses only those log entries that occurred within the given interval. Assuming that the End Time is in the past, the data workbench server typically updates the dataset using the same set of log entries, even if the dataset is updated by, for example, adding a new transformation.
Log Entry Condition

The Log Entry Condition determines which log entries from the total set of entries available in the dataset's data sources should continue to flow through the configuration process.

In essence, it is a filtering process on the available log entries. If the Log Entry Condition returns a value of false, the log entry is filtered out of the available set of log entries.

The Log Entry Condition is described through the use of condition operations (see Conditions) and can use any of the input fields collected by Sensor (see the Data Workbench Sensor Guide) or any extended fields produced by transformations contained within the Log Processing.cfg file to define the test conditions. Log Entry conditions are applied during log processing and optionally can be applied during transformation.

This example demonstrates the use of the log entry condition for website data. You can use the Log Entry Condition to create datasets that focus on a specific portion of the website or visitors performing some specific action on the site.

The Log Entry Condition in this example creates a dataset that includes only those log entries that are part of the site's store. By using the RECondition test with the matching pattern "/store/.*" and the cs-uri-stem field as input to the regular expression, only web pages that start with the string "/store/" are included in the dataset.

<table>
<thead>
<tr>
<th>Log Entry Condition</th>
<th>AndCondition</th>
<th>RECondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>/store/.*</td>
</tr>
<tr>
<td>Matches: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case Sensitive: true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments: Comment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input: cs-uri-stem</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key Splitting

The key splitting capability of the data workbench server enables tracking IDs associated with a large number of log entries to be split into multiple tracking IDs.

The number of tracking IDs in the dataset is artificially increased, but the total number of log entries processed by the data workbench server is not artificially increased, thereby preserving the total number of countable events in the dataset. After the data for a single element is split, the data is forever associated with two different tracking IDs and cannot be related.

For example, if you are working with web data, each tracking ID represents a unique visitor. If you enable key splitting, the visitors in your dataset with large amounts of event data are split into multiple visitors. While the number of visitors in the dataset is artificially increased, the total number of countable events such as page views or bookings is not artificially increased. After splitting occurs, the data for the sub-visitors cannot be related.

Key splitting uses a probabilistic algorithm. As a result, there is a trade-off between memory usage, the failure probability, the key splitting threshold (Split Key Bytes), and the dataset size. With the recommended settings (as listed below), the failure rate is low. Of those elements whose event data exceeds the key splitting threshold, approximately 1 in 22,000 (usually less than 1 per dataset) will have some of their data truncated rather than split.

The recommended values for each parameter (without and with key splitting) are shown in the following table.

Log Processing.cfg: Key Splitting
<table>
<thead>
<tr>
<th>Parameter</th>
<th>No Key Splitting</th>
<th>Key Splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Maximum Key Bytes</td>
<td>1e6</td>
<td>2e6</td>
</tr>
<tr>
<td>Split Key Bucket Space</td>
<td>6e6</td>
<td>6e6</td>
</tr>
<tr>
<td>Split Key Bytes</td>
<td>0</td>
<td>1e6</td>
</tr>
<tr>
<td>Split Key Space Ratio</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

**Group Maximum Key Bytes** specifies the maximum amount of event data that can be processed for a single tracking ID. Data exceeding this limit is filtered from the dataset construction process. **Split Key Bytes** represents the number of bytes at which a single tracking ID is split into multiple elements. Elements are split at approximately this number of bytes according to a probability distribution. **Split Key Space Ratio** and **Split Key Bucket Space** control the memory utilization and failure rate of key splitting.

*Note: Group Maximum Key Bytes, Split Key Bytes, Split Key Space Ratio, and Split Key Bucket Space all must be declared for key splitting to work properly. Do not change the values of these parameters without consulting Adobe.*

### ODBC Data Sources

The data workbench server (InsightServer64.exe) can read event data from any SQL database (for example, Oracle or Microsoft SQL Server) that has an ODBC 3.0 compliant driver.

The data workbench server's ODBC support is similar to existing support for loading data from Sensors or from log files generated by external processes. There are, however, some additional considerations and limitations:

- The data workbench server's ODBC support is compatible with the clustering capabilities. Data is distributed among all processing servers, and all subsequent processing (including query processing) benefits fully from clustering.
- ODBC support depends on third-party ODBC drivers. For ODBC support to work, these drivers must be configured on the machine on which the data workbench server runs, using tools external to the Adobe platform. Data workbench machines do not require any additional configuration.
- The table or view from which data is loaded must have an increasing ID column. For any row, the value in this column (which may be an actual column in the table or any SQL column expression) must not decrease as new rows are inserted into the database. If this constraint is violated, data is lost. For adequate performance, an index is required on this column or column expression.

*Note: It is possible for multiple rows to have the same value in the Increasing ID column. One possibility is a timestamp column with less than perfect precision.*

- The data workbench server cannot load columns with long data (data above a certain length as determined by the specific database application in use).
- Retrieving data from a database is slower than reading it from a disk file. Datasets that load data from an ODBC source take much longer to process (particularly when reprocessing) than equivalently sized datasets whose data comes from Sensors or other disk files.

For information about reprocessing your data, see Reprocessing and Retransformation.

### To configure Insight Server for ODBC event data
Configuring the data workbench server to load data from an SQL database requires that you first perform the following steps in order:

1. Install the appropriate database client software, including an ODBC driver, on the data workbench server machine on which the dataset is processed.

   **Note:** If you are loading ODBC event data for processing on a data workbench server cluster, you must install the database client software on all of the processing servers in the cluster. For information about specifying processing servers in a cluster, see the Server Products Installation and Administration Guide.

2. Configure a Data Source using the ODBC Data Source Administrator for Windows.

   It is important to note that the data workbench server (InsightServer64.exe) runs as a Windows service. Therefore, the Data Source ordinarily must be configured as a System DSN rather than a User DSN for the data workbench server to be able to use it. You can find more information about this configuration step in the documentation for your database software.

After installing the database client software on the appropriate data workbench server machine, you can configure the dataset to use the ODBC data source by editing the appropriate parameters in the Log Processing configuration file for the desired profile.

**Parameters**

For data from databases using the Open Database Connectivity (ODBC) standard, the following parameters are available:

**Log Processing.cfg: ODBC Data Sources**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The identifier for the ODBC source.</td>
</tr>
<tr>
<td>Data Source Name</td>
<td>A DSN, as provided by an administrator of the data workbench server machine on which the dataset is processed, that refers to the database from which data is to be loaded.</td>
</tr>
<tr>
<td>Database Password</td>
<td>The password to be used when connecting to the database. If a password has been configured for the DSN in the Data Source Administrator, this may be left blank. Any password supplied here overrides the password configured for the DSN in the Data Source Administrator.</td>
</tr>
<tr>
<td>Database User ID</td>
<td>The user ID to be used when connecting to the database. If a user ID has been configured for the DSN in the Data Source Administrator, this may be left blank. Any user ID supplied here overrides the user ID configured for the DSN in the Data Source Administrator.</td>
</tr>
<tr>
<td>Fields</td>
<td>A vector of column objects that specifies a mapping from data columns in the database to data fields in the data workbench server execution engine. Each column has entries <strong>Column Name</strong> and <strong>Field Name. Column Name</strong> is an SQL column expression that must be valid in the context of the table identified by <strong>Table Identifier</strong> described above. It may be a column name or any SQL expression based on any number of columns in the table. A formatting function may be necessary to convert values of certain data types to strings in a way that does not lose precision. All data is implicitly converted to strings using the database's default formatting method, which may cause data loss for some column data types (such as date/time data types) if explicit formatting expressions are not used.</td>
</tr>
<tr>
<td>Increasing ID Column</td>
<td>A column name or SQL column expression that meets the criterion that it increases (or at least does not decrease) as new rows are added. That is, if Row B is added to the table at a later time than</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
Row A, the value of this column (or column expression) in Row B must be greater (according to the database's native sorting order) than the corresponding value in Row A.  
- The **Increasing ID Column** name may be the same as the name of an existing column, but is not required to be.  
- This expression is assumed to have an SQL character data type. If the actual increasing ID column is of some other data type, this value must be a column expression to convert it to a string. Because this usually means comparisons are lexicographic (character by character), it is important to format the value carefully.  
- The expression is used in SQL ORDER BY clauses and compared to in SQL WHERE clauses. It is critically important to have an index built on the exact column expression that is used.

**Log Source ID**  
This parameter's value can be any string. If a value is specified, this parameter enables you to differentiate log entries from different log sources for source identification or targeted processing. The x-log-source-id field is populated with a value identifying the log source for each log entry. For example, if you want to identify log entries from an ODBC source named ODBCSource01, you could type `from ODBCSource01` and that string would be passed to the x-log-source-id field for every log entry from that source.  
For information about the x-log-source-id field, see *Event Data Record Fields*.

**Run On Server**  
Index value in the `profile.cfg` file of the processing server that makes the ODBC queries to get data from the database. (The Processing Servers parameter in the `profile.cfg` file lists all of the processing servers for the dataset, and each server has an index value, the first being 0.) The default value is 0.

**Table Identifier**  
An SQL expression that names the table or view from which data is to be loaded. A typical table identifier is of the form SCHEMA.TABLE.

This example shows the **Log Processing** configuration window in data workbench with an ODBC data source. This Data Source takes data from a table called `VISUAL.VSL` in a database with **Data Source Name** "VSTestO." Five (5) column objects (**Fields**) map data from the data columns in the database to the data workbench server.
Considerations for the Log Processing Configuration File

Conceptual information to consider when editing the Log Processing.cfg file.

- Data files should not be moved between directories after the sources for a dataset have been defined. The only additional files a directory should receive are newly created ones that result from the data workbench server receiving data from Sensor(s).
- Changing any of the parameters in this file requires reprocessing of all the data. The Pause parameter in the Log Processing Mode.cfg file must be set to false for reprocessing to occur. (Note that this parameter's default value is false, so changing the parameter may not be required.) For information about the Log Processing Mode.cfg file, see Additional Configuration Files.
- If you reprocess the data, you can check the Log Processing Progress parameter in data workbench's Processing Legend.

For information about reprocessing your data, see Reprocessing and Retransformation. For information about the Processing Legend, see the Data Workbench User Guide.

- The Log Processing.cfg file can be shared by multiple dataset profiles. Transformations defined in the Log Processing.cfg file are applied to all dataset profiles that share this configuration file.
Note: Adobe recommends defining transformations for the log processing in one or more log processing dataset include files. For information, see Log Processing Dataset Include Files.

- You can add any of the parameters described above to the Log Processing.cfg file by opening and editing the file in Notepad. Any changes you make and save appear when you reopen the file in data workbench. When adding a new parameter, use the Space key (not the Tab key) to indent two (2) spaces to the right of the previous heading level.

Any errors that occur during the log processing phase of the dataset construction process for a dataset profile are shown in the Profiles node of the Detailed Status interface in data workbench. For information about the Detailed Status interface, see the Data Workbench User Guide.
Transformation Configuration File

Information about the Transformation.cfg file and how its parameters affect the transformation phase of the dataset construction process.

About the Transformation Configuration File

The Transformation.cfg file controls the transformation phase of dataset construction during which additional data transformations are applied to data already processed during log processing to create extended dimensions for use in analysis.

You must edit the Transformation.cfg file to perform any of the following dataset configuration tasks:

• Filtering data from log processing by defining the log entry condition for transformation.
• Setting the time zone to be used for creating time dimensions and making time conversions. See Time Zones.

Note: Transformation Dataset Include files can contain additional instructions for the transformation phase of dataset construction. These files exist within the Dataset\Transformation directory for any inherited profile, and they typically define application-specific parameters (such as web-specific configuration parameters for the Site application). For information about Transformation Dataset Include files, see Dataset Include Files. For information about web-specific configuration parameters for Site, see Configuration Settings for Web Data.

Editing the Transformation Configuration File

Steps for editing the Transformation.cfg file for a dataset profile.

1. While working in your dataset profile, open the Profile Manager and click Dataset to show its contents.

   For information about opening and working with the Profile Manager, see the Data Workbench User Guide.

   Note: A Transformation subdirectory may exist within the Dataset directory. This subdirectory contains the Transformation Dataset Include files that have been created for one or more inherited profiles. For information about Transformation Dataset Include files, see Dataset Include Files.

2. Right-click the check mark next to Transformation.cfg and click Make Local. A check mark for this file appears in the User column.

3. Right-click the newly created check mark and click Open > in Workstation. The Transformation.cfg window appears.

   You also can open the Transformation.cfg file from a Transformation Dependency Map. For information about transformation dependency maps, see Dataset Configuration Tools.

4. Edit the parameters in the configuration file using the following table as a guide.

   When editing the Transformation.cfg file within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z), select section (click+drag), and select all (Ctrl+a). In addition, you can use the shortcuts to copy and paste text from one configuration file (.cfg) to another.
**Note:** A Transformation Dataset Include files for an inherited profile contains a subset of the parameters described in the following table as well as some additional parameters. For information about Transformation Dataset Include files, see Dataset Include Files.

### Transformation.cfg

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>End Time</strong></td>
<td>Optional. Filter data to include log entries with timestamps up to, but not including, this time. Adobe recommends using one of the following formats for the time:</td>
</tr>
<tr>
<td></td>
<td>• January 1 2013 HH:MM:SS EDT</td>
</tr>
<tr>
<td></td>
<td>• Jan 1 2013 HH:MM:SS GMT</td>
</tr>
<tr>
<td></td>
<td>For example, specifying &quot;July 29 2013 00:00:00 EDT&quot; as the End Time includes data through July 28, 2013, at 11:59:59 PM EDT.</td>
</tr>
<tr>
<td></td>
<td>You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by the data workbench server, see Time Zone Codes.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you specify a value for End Time, a parameter named End Time is set and applied throughout the transformation phase of dataset construction. For information about parameters, see Defining Parameters in Dataset Include Files.</td>
</tr>
<tr>
<td><strong>Extended Dimensions</strong></td>
<td>Optional. Adobe recommends defining extended dimensions in one or more Transformation Dataset Include files. For information, see Transformation Dataset Include Files.</td>
</tr>
<tr>
<td><strong>Hash Threshold</strong></td>
<td>Optional. A sampling factor for random sub-sampling of rows. If set to a number n, then only one out of each n tracking IDs enters the dataset, reducing the total number of rows in the dataset by a factor of n. To create a dataset that requires 100 percent accuracy (that is, to include all rows), you would set Hash Threshold to 1.</td>
</tr>
<tr>
<td></td>
<td>If Hash Threshold is specified in both the Log Processing.cfg and Transformation.cfg files, it is not applied in sequence; the maximum of the values set in either configuration file applies.</td>
</tr>
<tr>
<td><strong>Log Entry Condition</strong></td>
<td>Optional. Defines the rules by which log entries output from log processing are considered for inclusion in the dataset profile. See Log Entry Condition.</td>
</tr>
<tr>
<td><strong>New Visitor Condition</strong></td>
<td>Optional. For use with web data. Defines the rules by which visitors are considered for inclusion in the data. The New Visitor Condition defines the first log entry for a visitor (ordered by time) that is to be used in the dataset. All subsequent log entries for this visitor are included in the dataset regardless of whether they meet this condition. See New Visitor Condition.</td>
</tr>
<tr>
<td><strong>Reprocess</strong></td>
<td>Optional. Any character or combination of characters can be entered here. Changing this parameter and saving the file initiates data retransformation.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>For information about reprocessing your data, see <em>Reprocessing and Retransformation</em>.</td>
<td></td>
</tr>
<tr>
<td>Schema Checking</td>
<td>True or false. If true, then the data workbench server identifies dataset corruption problems and records information about the problems in log files in the data workbench server’s Trace directory. The default value is true. Adobe recommends leaving this parameter set to true at all times.</td>
</tr>
</tbody>
</table>
| Stages              | Optional. The names of the processing stages that can be used in Transformation Dataset Include files. Processing stages provide a way to order the transformations that are defined in Transformation Dataset Include files. This parameter is very helpful if you have defined one or more transformations within multiple Transformation Dataset Include files and you want specific transformations to be performed at specific points during transformation. The order in which you list the stages here determines the order in which the transformations in the Transformation Dataset Include files are executed during transformation. **Preprocessing** and **Postprocessing** are built-in stages; **Preprocessing** is always the first stage, and **Postprocessing** is always the last stage. By default, there is one named stage called **Default**. **To add a new processing stage**

  - In the Transformation.cfg window, right-click *Stages* and click *Add New > Stage*.  
  - Enter a name for the new stage.  

  **To delete an existing processing stage**

  - Right-click the number corresponding to the stage that you want to delete and click *Remove* <#stage_number>.  

  *Note:* When you specify a Stage in a Transformation Dataset Include files the name of the stage must match exactly the name that you enter here. For more information about dataset include files, see *Dataset Include Files*.                                                                                                                                                                                                                                                                                                                                                     |
| Start Time          | Optional. Filter data to include log entries with timestamps at or after this time. Adobe recommends using one of the following formats for the time:

  - January 1 2013 HH:MM:SS EDT
  - Jan 1 2013 HH:MM:SS GMT

  For example, specifying July 29 2013 00:00:00 EDT as the **Start Time** includes data starting from July 29, 2013, at 12:00:00 AM EDT.

  You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by data workbench Server, see *Time Zone Codes*.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
### Information About Specific Transformation Parameters

Information about specific parameters that you can define in the Transformation.cfg file.

#### New Visitor Condition

The New Visitor Condition is a Condition Operation that is used with website data to determine which visitors are considered for inclusion in the dataset.

The New Visitor Condition defines the first log entry (ordered by time) for a visitor that is to be used in the dataset, and all subsequent log entries for this visitor are included in the dataset regardless of whether they meet this condition. Because the New Visitor Condition requires that data is ordered by visitor and time, it is applied only during the transformation phase of dataset construction.

The New Visitor Condition shown in this example creates a dataset that includes only those log entries for visitors who respond to email campaigns. This is accomplished by using the NotEmptyCondition test (see Not Empty) and the x-campaign-email field as input to the regular expression. After the new visitors meeting the condition are identified, all log entries for those visitors are captured.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformations</td>
<td>Optional. Adobe recommends defining transformations for the transformation phase of dataset construction in one or more Transformation Dataset Include files. For information, see Transformation Dataset Include Files.</td>
</tr>
<tr>
<td>Time Zone</td>
<td>Time zone of the dataset profile. Time zones are used for time conversions and for creating time dimensions. See Time Zones.</td>
</tr>
</tbody>
</table>

**Note:** If you specify a value for Start Time, a parameter named Start Time is set and applied throughout the transformation phase of dataset construction. For information about parameters, see Defining Parameters in Dataset Include Files.

**Note:** When defined in the Log Processing.cfg file, the Time Zone parameter is used for time conversions only.

5. Right-click (modified) at the top of the window and click Save.
6. In the Profile Manager, right-click the check mark for Transformation.cfg in the User column, then click Save to > <dataset profile name> to make the locally made changes take effect. Retransformation of the data begins after synchronization of the dataset profile.

**Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

For information about reprocessing or retransforming your data, see Reprocessing and Retransformation.
Time Zones

The Time Zone parameter in the `Transformation.cfg` file controls time dimensions, time conversions (for example, defining the `x-local-timestring` field), and formatting of all local times in the dataset profile.

💡 **Note:** The Time Zone parameter does not affect system-level functionality such as timestamps in status and event logs, which are expressed in system local time.

The Time Zone parameter supports a system-independent time zone format ("Coordinated Universal Time") of the following format:

```
Time Zone = string: UTC +hhmm dstrules
```

The sign (+) can be either a plus (+) or a minus (-) sign, and `hhmm` is the offset from UTC in hours and minutes. The optional variable `dstrules` specifies a set of rules to implement Daylight Saving Time or a similar clock-shifting policy.

If you specify `dstrules`, a tab-delimited file named `dstrules.dst` must be present within the dataset profile’s `Dataset\TimeZone` subdirectory. The file specifies a time-zone independent set of rules for Daylight Saving Time. You can have different sets of rules for different years. The `DST.dst` file provided by Adobe in the Base profile specifies the standard U.S. rules established by the Energy Policy Act of 2005 (in effect starting 2007) and the U.S. rules for prior years.

Sample Time Zone entries are listed below:

- U.S. Eastern Daylight Time: `Time Zone = string: UTC -0500 DST`
- UTC time with no offset and no `dstrules`: `Time Zone = string: UTC -0000`

When this format is used, the system time zone of data workbench server, data workbench, and Report machines need not be the same as the specified time zone. In addition, all active dataset profiles on a data workbench server machine need not have the same time zone setting.

Adobe does not recommend running more than one dataset profile on a single data workbench server machine or data workbench server cluster.

Data workbench users will see data in the dataset profile's time zone instead of their system time zone. Adobe recommends that the system time zone for a data workbench server machine be the same as the time zone used in its datasets.

💡 **Note:** You can specify a Time Zone parameter in the `Log Processing.cfg` file, where it is used for time conversions during log processing. For information about the Time Zone parameter in the `Log Processing.cfg` file, see Log Processing Configuration File.

Considerations for the Transformation Configuration File

Important information to consider when editing the `Transformation.cfg` file.
• Changing any of the parameters in this file requires retransformation of the data.
• If you reprocess the data, you can check the Transformation Progress parameter in data workbench’s Processing Legend.

For information about reprocessing your data or the Processing Legend, see Reprocessing and Retransformation.

• CrossRows, ODBCLookup, Sessionize, and AppendURI transformations work only when defined in a Transformation Dataset Configuration file. For information about these transformations, see Data Transformations.

   Note: Adobe recommends defining transformations for the transformation phase of dataset construction in one or more Transformation Dataset Include files. For information, see Transformation Dataset Include Files.

• You can add any of the parameters described above to the Transformation.cfg file by opening and editing the file in Notepad. Any changes you make and save appear when you reopen the file in data workbench. When adding a new parameter, use the Space key (not the Tab key) to indent two (2) spaces to the right of the previous heading level.

   Any errors that occur during the transformation phase of the dataset construction process for a dataset profile are shown in the Profiles node of the Detailed Status interface in data workbench. For information about the Detailed Status interface, see the Data Workbench User Guide.

DeviceAtlas Distribution

The DeviceAtlas JSON file will now be distributed in a .bundle file (a renamed .tar.gz) along with DeviceAtlas.dll and DeviceAtlas64.dll files.

When the administrator upgrades the Insight Server to version 6.0, the DeviceAtlas.bundle file is included with the upgrade package in the Software and Docs profile (softdocs profile) located at:

Server Packages > v6.00 > Server_6.00.zip

The DeviceAtlas.bundle file is extracted to Server\Lookups\DeviceAtlas.

The DeviceAtlas.bundle file should be placed in a directory that is synchronized to the DPUs, and the DeviceAtlas.cfg file corresponding to the new DeviceAtlasComponent should be placed in the “Components for Processing Servers” directory on the synchronization master. When the DeviceAtlas.bundle file is changed, the very next DeviceAtlas lookup call will get results based on the updated API and/or JSON file.

Modify the Transformation.cfg file

The DeviceAtlas Transformations will no longer need to specify the path to the JSON file. Any previous DeviceAtlasTransformation that is defined in the transformation.cfg file should no longer include the File parameter that points to the obfuscated JSON file.

This example Transformation.cfg file shows the File argument that should be deleted to avoid confusion. (Leaving it there will not cause harm, but only potential confusion because it will be ignored.)

```plaintext
6 = DeviceAtlasTransformation:
   Comments = Comment: 0 items
   Condition = AndCondition: 0 items

File = string: Lookups\DeviceAtlas\20110106_private.json.obfuscated
   ^^ DELETE THE ABOVE LINE FROM ALL PREVIOUS TRANSFORMATIONS ^^
```
Modify the DeviceAtlas.cfg file

This is an example of the component argument required in the DeviceAtlas.cfg file.

```
component = DeviceAtlasComponent:
    DeviceAtlas Bundle File = string:Lookups\DeviceAtlas\DeviceAtlas.bundle
    Unsynchronized Bundle Extraction Path = string: Temp\DeviceAtlas
```

This DeviceAtlas.bundle file will be treated just like a configuration file from the perspective of the Profile Synchronization feature. In addition, the JSON data and DLL will be used at the Component level rather than at the individual Transformation level.

A new DeviceAtlasComponent, upon startup, finds the .bundle conglomeration, de-obfuscates the JSON file into memory, extracts the files into a temporary directory, and loads the appropriate DLL for the running platform. This component also monitors changes to the bundle file, and reloads the DLL and .cfg file automatically if it changes.

Running DeviceAtlas

Proper configuration makes a big difference in the time required for transformation. The transformation can be configured to run only once per visitor per session to allow DeviceAtlas to speed up the process.

If deployed using Log Processing.cfg:

Run the transformations twice.

1. Look up only the mobile id field, then
2. Create conditions to ignore the mobile id and then look up the rest of the fields.

If deployed using Transformation.cfg:

Deploy as in Step 1 in Log Processing above, or use cross-rows to support a conditional setting.

• Cross-Rows—Grab the previous session key. Then identify if the current session key is different from the one found with cross-rows. If so, then the DeviceAtlas transformation will only run on one record per session.

Modifying In-Memory Cache

Overview

The DeviceAtlas.bundle file uses an in-memory cache to greatly improve the performance of lookups. By default, it will cache up to 100000 user-agents and their properties. The LRU cache is entirely self-contained inside the DeviceAtlas.bundle file, so any version of the server capable of using a bundle file will automatically benefit from the increased performance as soon as the new file is loaded.

Modifying the DeviceAtlas.cfg file
The maximum size of the LRU cache can be configured by modifying the *Cached Elements Limit* parameter in the `DeviceAtlas.cfg` file.

💡 **Note:** This feature was added in the Data Workbench 6.4 release.

You can override the default *Cached Elements Limit* of "100000" by changing its value and saving the file. The default value has been selected to be sufficient for most needs.

```plaintext
component = DeviceAtlasComponent:
DeviceAtlas Bundle File = string:
Lookups\DeviceAtlas\DeviceAtlas.bundle
Unsynchronized Bundle Extraction Path = string:
Temp\DeviceAtlas\nCached Elements Limit = unsigned int: 100000
```
Dataset Include Files

Information about dataset include files and how their parameters affect the log processing and transformation phases of the dataset construction process.

About Dataset Include Files

Many of the internal profiles that you received with your Adobe application come with their own dataset configuration files.

Because the internal profiles are sub-profiles of the dataset profile, their dataset configuration files contain rules that provide additional parameters for the log processing or transformation phases of dataset construction. The dataset configuration files for internal profiles and for any inherited profiles that you create are called dataset include files.

A dataset include file contains a subset of the parameters contained in the main dataset configuration files (Log Processing.cfg or Transformation.cfg) for the dataset profile. Dataset include files containing parameters associated with log processing are called Log Processing Dataset Include files (see Log Processing Dataset Include Files), while dataset include files associated with transformation are called Transformation Dataset Include Files. See Transformation Dataset Include Files. You can create multiple dataset include files for use in the dataset construction process. The complete dataset includes all of the fields, transformations, and extended dimensions defined in all of the dataset configuration files for the dataset profile and any inherited profiles.

Working with Dataset Include Files

Dataset include files provide a flexible way to configure your dataset.

Within each file, you can define as few or as many fields, transformations, or dimensions as you desire, and you can organize the include files based on the inherited profile to which they belong. When configuring your dataset, you have the option of editing the dataset include files provided with the internal profiles for your Adobe application or creating new dataset include files for any inherited profiles that you create.

When you edit the parameters of a dataset include file for an internal profile and save the updated file to your dataset profile or an inherited profile that you create, you are, in effect, overriding the file's original settings. Adobe recommends editing a dataset include file for an internal profile whenever you need to make minor changes to the dataset's contents, such as changing a Condition parameter or a parameter's default setting. See Editing Existing Dataset Include Files. However, when you want to specify a new field to be passed from log processing to transformation, update or create new fields using transformations, or define extended dimensions, it is best to create a new dataset include file. See Creating New Dataset Include Files. You can edit the file that you create whenever or however you see fit.

Editing Existing Dataset Include Files

Steps to editing existing dataset include files.

You open an existing dataset include file using the Profile Manager in data workbench.

For information about opening and working with the Profile Manager, see the Data Workbench User Guide.

1. While working in your dataset profile, open the Profile Manager and click Dataset to show the contents of the directory.

   • To open a Log Processing Dataset Include file, click Log Processing to show the contents of the directory.
• To open a Transformation Dataset Include file, click Transformation to show the contents of the directory.

2. Right-click the check mark next to the desired dataset include file and click Make Local. A check mark for this file appears in the User column.

3. Right-click the newly created check mark and click Open > from the workbench. The configuration window appears.

You can also open a dataset include file from a Transformation Dependency Maps. For information about Transformation Dependency Maps, see Reprocessing and Retransformation.

4. Edit the parameters in the configuration file as appropriate. See Log Processing Dataset Include Files or Transformation Dataset Include Files for descriptions of the parameters.

When editing a dataset include file within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z), select section (click+drag), and select all (Ctrl+a). In addition, you can use the shortcuts to copy and paste text from one configuration file (.cfg) to another.

5. To save your changes, right-click (modified) at the top of the window and click Save.

6. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs. Reprocessing or retransformation of the data begins after synchronization of the dataset profile.

Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

Creating New Dataset Include Files

Steps to create a new dataset include file.

You should create a new dataset include file to perform any of the following dataset configuration tasks:

• Specifying new fields of data to be passed from log processing to transformation.
• Defining transformations that do either of the following:
  • Update existing log fields.
  • Produce new fields that are to be passed from log processing to transformation or that are used to define extended dimensions.

For information about the available transformation types, see Data Transformations.

Note: If you are defining transformations in a new dataset include file, be sure to keep the order of the inputs and outputs in mind. For information about the ordering of transformations, see Conventions for Constructing Transformations.

• Creating extended dimensions. For information about the available dimension types, see Extended Dimensions.

1. While working in your dataset profile, open the Profile Manager and click Dataset to view the existing dataset include files.

• To view the Log Processing Dataset Include files, click Log Processing.
• To view the Transformation Dataset Include files, click Transformation.
2. Create a new Log Processing or Transformation Dataset Include files by performing one of the following steps:
   • In the User column for the Log Processing directory, click Create > New Log Processing. A file named New Log Processing.cfg appears in the directory.
   • In the User column for the Transformation directory, click Create > New Transformation. A file named New Transformation.cfg appears in the directory.

3. Rename the new file by right-clicking its check mark in the User column and typing the new name in the File parameter.

4. Right-click the check mark for the renamed file and click Open > from the workbench. The configuration window appears.

5. Edit the parameters in the configuration file as appropriate. See Log Processing Dataset Include Files or Transformation Dataset Include Files for descriptions of the available parameters.

6. To save your changes, right-click (modified) at the top of the window and click Save.

7. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs. Reprocessing or retransformation of the data begins after synchronization of the dataset profile.

   Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

To edit a dataset include file that you created, see Editing Existing Dataset Include Files.

Types of Dataset Include Files

Information about Log Processing and Transformation dataset include files.

• Log Processing Dataset Include Files
• Transformation Dataset Include Files

Log Processing Dataset Include Files

The Log Processing Dataset Include file for an inherited profile contains parameters associated with the log processing phase of dataset construction.

The first line of a Log Processing Dataset Include file defines a type LogProcessingInclude that supports the Decoder Groups, Fields, Log Entry Condition, Parameters, Reprocess, Stage, and Transformations parameters. All other parameters for log processing must be defined in the Log Processing.cfg file in the dataset profile's
Dataset directory. You can name a **Log Processing Dataset Include** file anything you want, but its file extension must be `.cfg`. The file must be stored within the `inherited profile name\Dataset\Log Processing` directory. Because the files are loaded recursively during the log processing phase of dataset construction, you can store the **Log Processing Dataset Include** files at any level within the directory (for example, `inherited profile name\Dataset\Log Processing\folder name\include file name.cfg`).

**Note:** Many web-specific configuration parameters for Site are defined in **Log Processing Dataset Include** files. For information about these parameters, see *Configuration Settings for Web Data*.

### Log Processing Dataset Include File Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Decoder Groups** | Required if you have defined log file or XML file log sources in the `Log Processing.cfg` file. The text file or XML decoders that you define to extract fields of data from log file and XML log sources.  

**To add a new decoder group**  
- Right-click **Decoder Group** and click **Add new > TextFileDecoderGroup** or **XMLDecoderGroup**.  
- In the Name parameter for the new group, enter the desired name of the decoder group.  

**Note:** When you specify a Decoder Group in the `Log Processing.cfg` file for the dataset profile, the name must match exactly the name that you enter here. For more information, see *Log Files or XML Log Sources*.  

For information about the decoders that you can define for each group, see *Text File Decoder Groups* or *XML Decoder Groups*. |
| **Fields**         | Lists fields that are defined in **Log Sources** or **Transformations** in a **Log Processing Dataset Configuration** file but used in transformations, conditions, or extended dimensions in a **Transformation Dataset Configuration** file must be listed here.  

Each field below must be listed in some **Log Processing Dataset Include** file:  
- x-trackingid  
- x-timestamp |
| **Log Entry Condition** | Optional. Defines the rules by which log entries are considered for inclusion in the dataset. See *Log Entry Condition*.  

**Note:** To be included in the dataset, a log entry must satisfy the **Log Entry Condition** in the `Log Processing.cfg` file and in every **Log Processing Dataset Include** file. |
<p>| <strong>Parameters</strong>     | Optional. A variable that you can reference in other configuration parameters. For more information, see <em>Defining Parameters in Dataset Include Files</em>. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reprocess</td>
<td>Optional. Any character or combination of characters can be entered here. Changing this parameter and saving the file to the data workbench server initiates data reprocessing. For information about reprocessing your data, see <em>Reprocessing and Retransformation</em>.</td>
</tr>
<tr>
<td>Stage</td>
<td>Optional. The name of the processing stage that applies to this <strong>Log Processing Dataset Include</strong> file. The processing stages are defined in the Stages parameter in the <strong>Log Processing.cfg</strong> file. <strong>Note:</strong> When you specify a Stage, the name of the Stage must match exactly the name that is listed in the Stages parameter in the <strong>Log Processing.cfg</strong> file for the dataset profile.</td>
</tr>
<tr>
<td>Transformations</td>
<td>Optional. Defines the data transformations that need to be applied during log processing. For information about the available transformation types, see <em>Data Transformations</em>.</td>
</tr>
</tbody>
</table>

**Note:** For descriptions of the parameters in the **Log Processing.cfg** file, see *Log Processing Configuration File*.

You should keep the following points in mind whenever you are working with **Log Processing Dataset Include** files:

- Changing any of the parameters in this file requires reprocessing of all of the data.
- You can add any of the parameters described above to the **Log Processing Dataset Include** file by opening and editing the file in Notepad. Any changes you make and save appear when you reopen the file in data workbench. When adding a new parameter, use the Space key (not the Tab key) to indent two (2) spaces to the right of the previous heading level.

**Text File Decoder Groups**

The processing of log files as log sources requires the definition of a decoder within the **Log Processing Dataset Include** file to extract fields of data from the log entries.

Defining text file decoder groups for log file log sources requires knowledge of the log file’s structure and contents, the data to be extracted, and the fields in which that data is stored. This section provides basic descriptions of the parameters that you can specify for decoders, but the manner in which you use any decoder depends on the log file that contains your source data.

For information about format requirements for log file log sources, see *Log Files*. For assistance with defining text file decoders, contact Adobe.

A text file decoder group can include:

- **Regular Expression Decoders**
- **Delimited Decoders**
Regular Expression Decoders

A regular expression decoder identifies complex string patterns within the log entries in a log file and extracts these patterns as fields of data. For each decoder, the number of fields must equal the number of capturing sub-patterns in the regular expression. The portion of the line matching the nth capturing sub-pattern is assigned to the nth field for that line.

To add a regular expression decoder to a text file decoder group

1. Open the Log Processing Dataset Include file as described in Editing Existing Dataset Include Files and add a text file decoder group. See the table entry Decoder Groups.
2. Right click Decoders under the newly created decoder group, then click Add new > Regular Expression.
3. Specify the following information:
   - **Fields**: List of the fields in the log file. If any of the fields defined here are to be passed to the transformation phase of dataset construction, those fields must be listed in the Fields parameter of one of the Log Processing Dataset Include files for the dataset. Custom field names must begin with "x-".
   - **Name**: Optional identifier for the decoder.
   - **Regular Expression**: Used to extract the desired fields from each line in the file.
4. Repeat steps 4 and 5 for any other decoders that you want to add to the group.
5. To save the Log Processing Dataset Include file, right-click (modified) at the top of the window and click Save.
6. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column. Click Save to <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

💡 **Note**: A given log file can have multiple regular expression decoders. The order in which you define the decoders is important: the first decoder to match a line in the log file is the one used to decode that line.

This example illustrates the use of a regular expression decoder to extract fields of data from a tab-delimited text file. You can achieve the same result by defining a delimited decoder with a tab delimiter.

```
-Traffic.cfg
| Log Processing Include LogProcessingInclude
|   -Decoder Groups
|     -0
|       Name Traffic Decoder Group
|       -Decoders
|         -0 REDecoder
|           -Fields
|             0 x-trackingid
|             1 x-query
|             2 x-time
|           Regular Expression (^[TAB]+)[TAB][^[TAB]+][TAB][^[TAB]+]
```

For more information about regular expression decoders, including terminology and syntax, see Regular Expressions.
Delimited Decoders

A delimited decoder decodes a log file whose fields are delimited by a single character. The number of fields must correspond to the number of columns in the delimited file; however, not all fields need to be named. If a field is left blank, the column is still required in the log file, but the decoder ignores it.

To add a delimited decoder to a text file decoder group
1. Open the Log Processing Dataset Include file as described in Editing Existing Dataset Include Files and add a text file decoder group. See the table entry Decoder Groups.
2. Right click Decoders under the newly created decoder group, then click Add new > Delimited.
3. Specify the following information:
   - **Fields**: List of the fields in the log file. If any of the fields defined here are to be passed to the transformation phase of dataset construction, those fields must be listed in the Fields parameter of one of the Log Processing Dataset Include files for the dataset. Custom field names must begin with "x-".
   - **Delimiter**: Character that is used to separate fields in the output file.
4. Repeat steps 4 and 5 for any other decoders that you want to add to the group.
5. To save the Log Processing Dataset Include file, right-click (modified) at the top of the window and click Save.
6. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

**Note**: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

This example illustrates the use of a delimited decoder to extract fields of data from a comma-delimited text file containing data about movies.

```
- Log Processing Include LogProcessingInclude
  - Decoder Groups
    - 0 Name TextFileDecoderGroup
    - Decoders
      - 0 DelimitedDecoder
        - Fields
          - 0 x-movieid
          - 1 x-trackingid
          - 2 x-rating
          - 3 x-date
        Delimiter ,
```

XML Decoder Groups

The processing of XML files as log sources the definition of a decoder within a Log Processing Dataset Include file to define decoders for extracting data from the XML file.

**Note**: Defining XML decoder groups for XML log sources requires knowledge of the XML file’s structure and contents, the data to be extracted, and the fields in which that data is stored. This section provides basic
descriptions of the parameters that you can specify for decoders. The manner in which you use any decoder depends on the XML file that contains your source data.

For information about format requirements for XML log sources, see XML Log Sources. For assistance with defining XML decoders, contact Adobe.

The top level of an XML decoder is a decoder group (XMLDecoderGroup), which is a set of decoder tables that you use to extract data from an XML file of a particular format. If you have XML files of different formats, then you must define a decoder group for each format. Each decoder group consists of one or more decoder tables.

The following table describes the Tables parameter and all of the subparameters that you must specify to define an XML decoder group.

**XMLDecoderGroup**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Tables    | Each table in a decoder group represents one level of data to be extracted from the XML file. For example, if you want to extract data about visitors, then you would create a decoder table that consists of the information you want to extract for each visitor. You also can create decoder tables within decoder tables (see Children). **To add a table to a decoder group**
  - Right-click **Tables** and click **Add new > XMLDecoderTable**. |
| Fields    | The extended fields (for example, x-trackingid, x-email) in which the data is stored. The data to be stored in the field is determined by the Path and/or Operation subfields. The Path is the field's level within the structured XML file. A field's path is relative to the path of the table in which it is defined. Examples include `tag.tag.tag` or `tag.tag.tag.@attribute`. Note that paths are case-sensitive. An Operation is applied to each line in the specified path to produce an output. The following operations are available:
  - **LAST**: The field takes the value of the path's last occurrence in the XML file.
  - **RANDOM**: Assigns a random value to the field. This operation is useful if you need to generate a unique id, such as for the x-trackingid field.
  - **INHERIT**: The defined field inherits its value from the parent table's corresponding field.
  - **"constant "**: The constant must be enclosed in quotation marks. You can use a constant operation to check for the existence of a particular path; if the path exists, then the field is assigned the constant's value. **To add a field to a decoder table**
  - Right-click **Fields**, then click **Add new > XMLDecoderField**. Define Field, Operation and Path as appropriate. |
| Path      | The level within the structured XML file for which the decoder table contains information. For a child XML decoder table, the path is relative to the parent table's path. Note that paths are case-sensitive. |
### Parameter | Description
--- | ---

**For example,** if your XML file contains the structure:

```xml
<logdata>
  <visitor>

  ...

  </visitor>

</logdata>
```

then the path would be `logdata.visitor`.

| Table | The value of this parameter should always be “Log Entry.”

> **Note:** *Do not change this value without consulting Adobe.* |

| Children | Optional. One or more embedded decoder tables. Each child includes the Fields, Path, and Table parameters described above.

**To add a child to a decoder table**

- Right-click *Children* and click *Add new > XMLDecoderTable*. Define Field, Operation and Path as appropriate.

---

To use an XML file as a log source for a dataset, XML decoder groups and tables must be defined to extract the information that is to be processed into the dataset. In this example, you can see how to define decoder groups and tables for a sample XML log source for a web dataset.

The following XML file contains information about a website visitor, including a visitor ID, email address, physical address, and information about the visitor’s page views.
Since we have a single XML file, we need only one decoder group, which we name "Sample XML Format." This decoder group applies to any other XML files of the same format as this file. To begin constructing XML decoder tables within this decoder group, we must first determine what information we want to extract and the fields in which the data will be stored.

In this example, we extract information about the visitor and the page views associated with that visitor. To do this, we create a top-level (parent) XML decoder table with information about the visitor and an embedded (child) XML decoder table with information about that visitor's page views.

**Information for the parent (visitor) table is as follows**

- A data type identifier for each row of data in the XML file. We use VISITOR as our identifier so that we can quickly identify rows of data pertaining to the visitor and not to the page views. We can store this value in the x-rowtype field.
- The visitor's ID, which we store in the x-trackingid field.
- The visitor's email address (contact.email), which we store in the x-email field.
- The visitor's registration status. If the visitor is a registered user, then we can store the value "1" in the x-is-registered field.
- The Path value is logdata.visitor, and the Table value is Log Entry. For information about these parameters, see the XMLDecoderGroup table above.

**Information for the child (page views) table is as follows:**

- A data type identifier for each row of data in the XML file. We use "PAGEVIEW" as our identifier so that we can quickly identify rows of data pertaining to the visitor's page views and not to the visitor only. We store this value in the x-rowtype field.
- The visitor's ID. This value is inherited from the parent table and is stored in the x-trackingid field.
- The timestamp of each page view, which is stored in the x-event-time field.
- The URI of each page view, which is stored in the cs-uri-stem field.
• The Path value is pageview, and the Table value is "Log Entry." For information about these parameters, see the XMLDecoderGroup table above.

The following screen capture shows a portion of Log Processing Dataset Include file with the resulting XML decoder group for the sample XML file based on the discussed structure of the parent and child XML decoder tables.

A table showing the output of this decoder for our sample XML file looks something like the following:

<table>
<thead>
<tr>
<th>x-rowtype</th>
<th>cs-uri-stem</th>
<th>x-email</th>
<th>x-is-registered</th>
<th>x-event-time</th>
<th>x-tracking-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISITOR</td>
<td><a href="mailto:foo@bar.com">foo@bar.com</a></td>
<td>1</td>
<td>registered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGEVIEW</td>
<td>/index.html</td>
<td>2006-01-01 08:00:00</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGEVIEW</td>
<td>/</td>
<td>2006-01-01 08:00:30</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
You can create a table like the one above in data workbench by using a field viewer interface. For information about the field viewer interface, see Dataset Configuration Tools.

Transformation Dataset Include Files

The Transformation Dataset Include file for an inherited profile contains parameters associated with the transformation phase of dataset construction.

The first line of the file defines a type TransformationInclude that supports the Extended Dimensions, Parameters, Reprocess, Stage, and Transformations parameters. All other parameters must be defined in the Transformation.cfg file in the dataset profile's Dataset directory.

Including parameters other than Extended Dimensions, Parameters, Reprocess, Stage, and Transformations in a Transformation Dataset Include file generates errors.

You can name a Transformation Dataset Include file anything you want, but its file extension must be .cfg. The file must be stored within the inherited profile name\Dataset\Transformation directory. Because the files are loaded recursively during the transformation phase of dataset construction, you can store the Transformation Dataset Include files at any level within the directory (for example, inherited profile name\Dataset\Transformation\folder name\include file name.cfg).

Note: Many web-specific configuration parameters for Site are defined in Transformation Dataset Include files. For information about these parameters, see Configuration Settings for Web Data.

The following table describes the parameters that are available in a Transformation Dataset Include file:

### Transformation Dataset Include File Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Dimensions</td>
<td>Optional. Defines the extended dimensions. See Extended Dimensions.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Optional. A variable that you can reference in other configuration parameters. For more information, see Defining Parameters in Dataset Include Files.</td>
</tr>
<tr>
<td>Reprocess</td>
<td>Optional. Any character or combination of characters can be entered here. Changing this parameter and saving the file initiates data retransformation. For information about reprocessing your data, see Reprocessing and Retransformation.</td>
</tr>
<tr>
<td>Stage</td>
<td>Optional. The name of the processing stage that applies to this Transformation Dataset Include file. The processing stages are defined in the Stages parameter in the Transformation.cfg file. Note: When you specify a Stage, the name of the Stage must match exactly the name that is listed in the Stages parameter in the Transformation.cfg file for the dataset profile.</td>
</tr>
<tr>
<td>Transformations</td>
<td>Optional. Defines the data transformations that need to be applied during transformation. For information about the available transformation types, see Data Transformations.</td>
</tr>
</tbody>
</table>
You should keep the following points in mind whenever you are working with Transformation Dataset Include files:

- Changing any of the parameters in this file requires retransformation of the data.
- **CrossRows**, **ODBCLookup**, **Sessionize**, and **AppendURI** transformations work only when defined in a Transformation Dataset Configuration file. For information about these transformations, see Data Transformations.
- You can add any of the parameters described above to the Transformation Dataset Include file by opening and editing the file in Notepad. Any changes you make and save appear when you reopen the file in data workbench. When adding a new parameter, use the Space key (not the Tab key) to indent two (2) spaces to the right of the previous heading level.

If you subscribe to Adobe's IP Geo-location or IP Geo-intelligence data service, Adobe provides you with an internal profile consisting of a set of data transformations and extended dimensions that are created specifically for the data service. The transformations and dimensions are defined in Transformation Dataset Include files that are included in the Dataset directory of the internal profile. For instructions to install the internal profile for the IP Geo-location or IP Geo-intelligence data service, see the Data Workbench User Guide.

### Defining Parameters in Dataset Include Files

When configuring your dataset, you can define variables, referred as parameters, to represent meaningful values.

To assign a value to a parameter (that is, to define the parameter), you add the parameter's name and value to the Parameters vector in a log processing or Transformation Dataset Include file. After you define parameters, you can reference them in your dataset profile's configuration files. Defining and referencing such parameters is referred to as parameter substitution. Using parameter substitution when configuring your dataset enables you to create a centralized location for your parameter definitions. When you need to update a parameter that is referenced multiple times or in multiple files, you have to make the change only once.

**Note:** In this guide, the term parameter has been used to refer to the name of any setting in a configuration file (such as Log Entry Condition, Reprocess, or Transformations). However, as used in this section, parameter refers specifically to a member of the Parameters vector in a dataset include file and not to the name of a setting in a configuration file.

You should consider the following points when defining a parameter:

- A parameter must be defined exactly once. Therefore, you cannot define the same variable in multiple dataset include files.
- Any parameter that you define is local to either the log processing or the transformation phases, but it is global across multiple dataset configuration files for that phase. For example, if you define a parameter in a Transformation Dataset Include file, the parameter is defined for the entire transformation phase, and you can reference it in the Transformation.cfg file and all other Transformation Dataset Include files for the inherited profiles. The parameter would not be defined for log processing; therefore, any references to the parameter in the Log Processing.cfg file or a Log Processing Dataset Include file would generate a processing error.

**To define a parameter**

You can define string, numeric, and vector parameters in Log Processing and Transformation Include files.
1. In the data workbench window for the Log Processing or Transformation Dataset Include file, right-click Parameters, then click Add new > Parameter.

2. Select String Parameter, Numeric Parameter, or Vector Parameter, and complete the Name and Value parameters as described in the following sections.

3. To save the dataset include file in which you have defined the parameter, right-click (modified) at the top of the window and click Save.

4. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

**Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

To reference a parameter

- When you reference a defined parameter in another dataset configuration file, you must type its name as $(parameter name).

The following sections describe the types of parameters that you can define.

- **String and Numeric Parameters**
- **Vector Parameters**

### String and Numeric Parameters

String and numeric parameters take as their values strings and numbers, respectively.

You can use them interchangeably, but numeric parameters must be defined to have a numerical value. You can reference string and numeric parameters when defining transformations, conditions, and extended dimensions, and you can reference more than one parameter in the same line.

You cannot reference string and numeric parameters in Input or Output fields, but you can use a string parameter to define a constant input field. In addition, you cannot reference string and numeric parameters in decoders or decoder groups.

This example shows a Log Processing Dataset Include file that defines a string parameter and a numeric parameter. Note that the string parameter, named “Value Lookups,” defines a file location (Lookups\Values) relative to the data workbench server installation directory.

```
-Log Processing Parameters.cfg (modified)
-Log Processing Include LogProcessingInclude
-Parameters
  -0  StringParameterDef
    Name: Value Lookups
    Value: Lookups\Values
  -1  NumericParameterDef
    Name: Maximum Count
    Value: 32768
```
Vector Parameters

Vector parameters contain multiple values for a single variable.

You can reference vector parameters only as the sole item of a vector. This example shows a Transformation Dataset Include file that defines a vector parameter. The vector parameter, "Internal Domains," consists of three values.

```
-Internal Domains.cfg
  | -Transformation Include TransformationInclude
  |   -Parameters
  |     -0 VectorParameterDef
  |       Name Internal Domains
  |       -Value
  |       0 abcsite.com
  |       1 123site.com
  |       2 xyzsite.com
```

Note that the vector parameter is the only item listed for the Matches vector in the String Match condition.

```
-Condition AndCondition
  | -0 StringCondition
  |   -Matches
  |     0 $Internal Domains
  |     Input cs(referrer-domain)
```

For more information about internal domains, see Configuration Settings for Web Data. For information about the String Match condition, see Conditions.
Additional Configuration Files

Information about some of the additional configuration files contained in the Dataset directory for the dataset profile or its inherited profiles.

Although these files are not always required to define a dataset profile, you can edit them to specify additional parameters for processing the data.

Log Processing Mode.cfg

The configuration file Log Processing Mode.cfg enables you to pause processing of data into a dataset, specify offline sources, or specify the frequency at which data workbench server saves its state files.

Making changes to the Log Processing Mode.cfg file, including adding or removing sources, does not cause reprocessing of the data.

To edit the Log Processing Mode.cfg file for a dataset profile

1. While working in your dataset profile, open the Profile Manager and click Dataset to show its contents.

   **Note:** If the Log Processing Mode.cfg file is not located in the directory for the desired profile, you need to copy this file from the Base directory on the data workbench server machine into the profile’s directory.

   For information about opening and working with the Profile Manager, see the Data Workbench User Guide.

2. Right-click the check mark next to the configuration file’s name and click Make Local. A check mark for this file appears in the User column.

3. Right-click the newly created check mark and click Open > from the workbench. The configuration window appears.

4. Edit the parameters in the configuration file using the following table as a guide.

   **Note:** Some of the parameters in the Log Processing Mode.cfg file have names that include Fast Input or Fast Merge. Fast Input refers to the log processing phase of dataset construction and is responsible for approximately half of the total dataset processing time. Fast Merge refers to the transformation phase of dataset construction only when preceded by log processing. Fast Merge does not occur during retransformation that results from modifying a Transformation Dataset Configuration file. Like Fast Input, Fast Merge is also responsible for approximately half of the dataset processing time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Cloud Bytes | A tuning parameter that affects the efficiency of data transformation. The default value is 128000000.  
  **Note:** You should not change this value without consulting Adobe. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Input Decision Ratio</td>
<td>A tuning parameter that specifies the ratio of total to unread log bytes at which the system enters <strong>Fast Input</strong> mode (and subsequently <strong>Fast Merge</strong>) instead of processing data in real time. The default value is 200, meaning that the system enters <strong>Fast Input</strong> mode from real-time mode when the unread log data is at 1/200th of the total data. A higher decision ratio makes the system enter <strong>Fast Input</strong> mode more readily, while a lower ratio makes it less likely to enter <strong>Fast Input</strong> mode. <strong>Note:</strong> Setting the parameter to 0 prevents the system from entering <strong>Fast Input</strong> mode at all, even for initial processing. Setting the parameter to 1.1 enables the system to enter <strong>Fast Input</strong> during initial processing but not for subsequent processing. Adobe does not recommend using values between 0 and 1.1. For more information about setting this parameter, contact Adobe.</td>
</tr>
<tr>
<td>Fast Input FIFO Bytes</td>
<td>A tuning parameter that balances memory usage and system performance during data processing. The default value is 120000000. <strong>Note:</strong> You should not change this value without consulting Adobe.</td>
</tr>
<tr>
<td>Fast Merge Buffer Bytes</td>
<td>A tuning parameter that balances memory usage and system performance during data processing. The default value is 128000000. <strong>Note:</strong> You should not change this value without consulting Adobe.</td>
</tr>
<tr>
<td>Offline Sources</td>
<td>Mask of the offline log source. <strong>To specify an offline source</strong>  - Right-click <strong>Offline Sources</strong>, then click <strong>Add new &gt; Source</strong>.  - In the parameter for the new source, enter the mask of the log sequence. For Sensor log sources with file names of the format YYYYMMDD-SENSORID.vsl, the mask is SENSORIDSENSORID is case-sensitive. For log file log sources, the mask is the string extracted by the <strong>Mask Pattern</strong>. See Log Files. Adding or removing sources from Offline Sources does not cause reprocessing of the dataset. <strong>As Of time measurements</strong> are maintained for the processing of the profile's online sources. When the offline source is again online, the processing of incoming log files for that source resumes. Whenever a source comes back online, you should remove it from Offline Sources. If you do not do so, data workbench server treats the source as an online source and updates the <strong>As Of time</strong> as long as the source is sending data. If the source goes offline again, the <strong>As Of time</strong> measurements stop.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Paused</td>
<td>True or false. If true, new data is not processed into the dataset. The default value is false.</td>
</tr>
<tr>
<td>Real Time Delay</td>
<td>The amount of time in seconds that data workbench Server waits between intervals of processing data into the dataset. When this value is set to zero, the system attempts to keep up with incoming data in real time. The default value is zero (0), but you can increase this value to reduce CPU load.</td>
</tr>
</tbody>
</table>
| Real Time FIFO Bytes      | The amount of memory in bytes used to store data that is waiting to be processed into the dataset. You may need to change this value based on the number of seconds that you specify for Real Time Delay. The default value is 16000000.  
  
  **Note:** You should not change this value without consulting Adobe. |
| Save Interval (sec)       | Frequency at which the data workbench server saves its state files. The default value is 3600.                                               |
|                           |  
  
  **Note:** You should not change this value without consulting Adobe. |

When editing the Log Processing Mode.cfg file within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z), select section (click+drag), and select all (Ctrl+a). In addition, you can use the shortcuts to copy and paste text from one configuration file (.cfg) to another.

5. Right-click (modified) at the top of the window and click Save.

6. In the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <datasetprofile name>.

  **Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

### Server.cfg

The Sample Bytes parameter in the Server.cfg file specifies the data cache size (in bytes) for data workbench.

The default value is 250e6. Instructions for opening and saving the Server.cfg file are the same as those for Log Processing Mode.cfg. See Log Processing Mode.cfg.

  **Note:** Because this file’s parameter affects system performance, please contact Adobe before making any changes.

You can further limit the size of the data cache for data workbench machine that connect to the data workbench server by setting the Maximum Sample Size parameter in the Insight.cfg file. For more information, see the Data Workbench User Guide.
Other Files

The Dataset directory includes additional files that are either required for the operation of the software or provide additional functionality for your Adobe implementation.

- **Client.cfg**: The Client.cfg file within the Dataset directory for the Base profile is required for operation of the software. Do not delete or modify any of the parameters in the Client.cfg file.

- **Cluster.cfg**: The Cluster.cfg file within the Dataset directory for the Base profile is required for operation of the software. In the Cluster.cfg file, you should modify only the Normalize Server parameter if you are configuring a dataset to be processed on an data workbench server cluster. For instructions to modify the Normalize Server parameter, see Creating a Centralized Normalization Server for a Cluster.

- **Insight Transform.cfg and Insight Transform Mode.cfg**: If you are using transformation functionality, you have two additional configuration files, data workbench Transform.cfg and data workbench TransformMode.cfg, in the Dataset directory for the Transform profile. For information about these files and their parameters, see Transform Functionality.

- **The PAServer.cfg file**: If you want to submit Predictive Analytics clustering jobs to Insight Servers, then you will need to configure the PAServer.cfg file for handling server-side clustering submissions.

   The custom profile should inherit the PAServer.cfg from the Predictive Analytics profile (Server\Profiles\Predictive Analytics\Dataset).

   ! **Important:** Set a Master Server in this file and save the PAServer.cfg to the implementation site.

   ```
   PAServer = PAServerConfig:
   Master Server = serverInfo:
   Address = string:
   Port = int: 80
   Use SSL = bool: false
   ```
Data Transformations

Overview of data transformations, describes each of the available transformation types, and describes how to use the transformations.

About Transformations

Transformations enable you to extract information available in your data files and manipulate it into a more useful form.

Transformations operate on the log entries (you can think of log entries as rows of data) in your log sources. For each row of data, the transformation takes the value of the specified input field, performs a set of processing steps, and records the result in the output field that you specify. You can define transformations to be executed during either the log processing or transformation phase of the dataset construction process:

- **During log processing**: Transformations executed during the log processing phase of dataset construction are applied to each event data record (log entry) to update existing log fields or produce new fields. The results of the transformations are used along with log entry conditions to evaluate which log entries are filtered out of the dataset during log processing.

- **During transformation**: Transformations executed during the transformation phase of dataset construction operate on the fields of data passed from log processing to create extended dimensions that you can use in your analyses. See Extended Dimensions.

💡 Note: The data input to transformation from log processing is ordered by time and grouped by the tracking ID in your source data. Several transformations require that the data is in this form and work only when defined in during transformation.

Changes to transformations must be made with care. Transformations do not affect which log entries flow into the dataset construction process, but they do affect the results presented. This permits changes to be made in what is being analyzed without changing the data upon which the analysis is based. However, changes in transformations can fundamentally alter the values produced in analyses.

Conventions for Constructing Transformations

Table showing what conventions apply when constructing transformations.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential execution</td>
<td>The transformations within a dataset configuration file are applied to the log entries sequentially (that is, in the order in which they are listed in the configuration file). Therefore, transformations must be listed in the order their outputs are used as inputs to other transformations. More specifically, if the output of one transformation is used as the input to another transformation, it is important for that former transformation to be listed prior to the latter transformation in the dataset configuration files. Otherwise, the data workbench server generates an error. Processing stages provide a way to order the transformations that are defined within multiple dataset include files. For all of the dataset include files associated with a particular processing stage, transformations are ordered based on their inputs and outputs. In addition, if multiple</td>
</tr>
</tbody>
</table>
### Convention

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset include files within a stage output data to the same field as a result of a transformation, the data workbench server generates an error. For more information about stages, see Log Processing Configuration File, Transformation Configuration File, and Dataset Include Files. A Transformation Dependency Map can display how a field is modified by a series of transformations. See Dataset Configuration Tools.</td>
</tr>
</tbody>
</table>

### Output names

Most transformations specify an output field. If the output is a user-defined extended field, the name for this field must start with "x-". The output field names cannot contain spaces or special characters. The names of extended fields can be written with mixed-case, such as "x-NewCampaignName," or "x-New-Campaign-Name" for readability, but they are treated by the software as case-insensitive.

### Input fields

Input fields refer to one of the baseline fields or a user-created field resulting from the output of a previous transformation. If a constant string is needed, a quoted string can be used instead of a baseline or user-created field. For a list of some of the commonly defined fields of data that the data workbench server can process, see Event Data Record Fields.

### Simple strings and vectors of strings

All transformations operate on strings and/or vectors of strings. Simple strings are literal sequences of characters. String vectors contain zero or more simple strings in a specific order.

---

**Defining a Transformation**

You can define data transformations to be applied during either the log processing or transformation phase of dataset construction.

> **Note:** Adobe recommends defining transformations in either Log Processing or Transformation Dataset Include files instead of in Log Processing.cfg or Transformation.cfg.

The following transformations work only when defined in the Transformation.cfg file or in a Transformation Dataset Include file:

- AppendURI
- CrossRows
- LookupRows
- ODBC Data Sources
- Sessionize

To define a transformation

1. Use the Profile Manager to open the dataset configuration file in which you want to define the transformation.
   - (Recommended) To open a dataset include file, see Dataset Include Files.
   - To open the Log Processing.cfg file, see Editing the Log Processing Configuration File.
   - To open the Transformation.cfg file, see Editing the Transformation Configuration File.
2. Right-click **Transformations**, then click **Add new > Transformation type**.

3. Input the appropriate information for your transformation. For descriptions of the transformation types and information about their parameters, see the following sections:

   - **Standard Transformations**
   - **URI Transformations**
   - **Integrating Lookup Data**

4. After you have defined your transformation(s) in the configuration file, save the file locally and save it to your dataset profile on the data workbench server.

Tips for defining and editing transformations:

   - When editing the configuration of a transformation within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z), select section (click+drag), and select all (Ctrl+a). In addition, you can use the shortcuts to copy and paste text or entire transformation definitions from one configuration file (.cfg) to another.
   - For any transformation that you define, you can add one or more comment lines to the Comments parameter to further describe the transformation or add notes about its use. To add a comment using data workbench, right-click the Comments label, then click **Add new > Comment Line**.
   - You can open the configuration of any transformation from a Transformation Dependency Map. After you open the configuration, you can edit it and save your changes. For information about Transformation Dependency Maps, see **Dataset Configuration Tools**.
   - An empty string output from a transformation can overwrite a non-empty string in the output field.

### Types of Transformations

Insight Server provides standard and URI transformations.

- **Standard Transformations**
- **URI Transformations**

### Standard Transformations

A standard transformation takes a set of inputs, performs some operation, then provides that result in one or more output fields.

The following sections provide a description, a table of available parameters, and an example for each of the standard transformations. Every data workbench server DPU provides all of the standard transformations.

#### ChangeCase

The **ChangeCase** transformation changes the case of the string in the Input parameter as specified by the **Action** parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>Upper or lower. Specifies whether the case is to be changed to upper or lower.</td>
<td>lower</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
</tbody>
</table>
The conditions under which this transformation is applied.

Input  The name of the field from the log entry to use as input.

Output The name of the output field.

In this example, which uses fields of data collected from website traffic, the case of the string within the s-dns field is changed to lower case, and the new value is output in the new field, x-lowercase-dns.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the log entry to use as input.</td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
</tr>
</tbody>
</table>

In this example, which uses fields of data collected from website traffic, the case of the string within the s-dns field is changed to lower case, and the new value is output in the new field, x-lowercase-dns.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the log entry to use as input.</td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
</tr>
</tbody>
</table>

In this example, which uses fields of data collected from website traffic, the case of the string within the s-dns field is changed to lower case, and the new value is output in the new field, x-lowercase-dns.

### Copy

The **Copy** transformation simply copies the value in the input field to the given output field. If the input field could be a vector of strings, the output field must start with "x-:"

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Default</td>
<td>Used if the condition test is true and the input value is not available in the given log entry.</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from which to copy.</td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
</tr>
</tbody>
</table>

In this example, which uses fields of data collected from website traffic, the output field, x-purchase-success, is given the literal value of "1" each time cs-uri-stem matches /checkout/confirmed.php. If the **Condition** is not satisfied (that is, cs-uri-stem does not match /checkout/confirmed.php), x-purchase-success is not changed.
Like other transformations, the **CrossRows** transformation is applied to the rows of data (log entries) in your log sources.

For each row of data, the transformation takes the value of the specified input field, performs a set of processing steps, and records the result in the output field that you specify. However, when the **CrossRows** transformation works on one row of data (this row is called the output row), it takes into account that row plus one or more other rows of data (these rows are called input rows) that are associated with the same tracking ID. Therefore, for a given tracking ID, the value of the output field for each output row is based on the values of the input field for one or more input rows.

The transformation provides multiple conditions and constraints that enable you to limit the input rows for the transformation. You can express these limits in terms of the data workbench server's conditions (see *Conditions*), a range of input rows relative to the output row, or a range of times relative to the time of the output row. For those input rows that satisfy the transformation's conditions and constraints, you can apply an operation (such as SUM) that determines the value of output field.

**Note:** To work, the **CrossRows** transformation requires that the data is ordered in time and grouped by the tracking ID in your source data. Therefore, **CrossRows** works only when defined in the *Transformation.cfg* file or in a *Transformation Dataset Include* file.

As you review the descriptions of the parameters in the following table, remember the following:

- The output row is the row of data that the transformation is working on at a given point in time.
- Input rows are all of the other rows of data (before, after, or including the output row) whose values of the input field serve as inputs to the transformation. Input rows are subject to the Input Condition, Key, Row Begin, Row End, Time Begin, and Time End parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Condition</td>
<td>Limits the output of the transformation to certain log entries. If the condition is not met for a particular log entry, the field in Output parameter is left unchanged. The input still may be used to affect other log entries.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the input row to use as input.</td>
<td></td>
</tr>
<tr>
<td>Input Condition</td>
<td>Accepts input for the transformation from only certain input rows. If the Input Condition is not met for a particular input row, the input field from that row is ignored and does not affect other output rows. However, the output field from that row is still modified per the specified Condition.</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Optional. The name of the field to use as the key. If a key is specified, the input rows for a given output row are limited to the contiguous block of rows having the same Key value as the output row. This restriction is in addition to all other limitations placed on the input rows by other parameters of the CrossRows transformation. For example, if you are working with web data and you make the field x-session-key (which has a unique value for each session) the key, then the input rows for the transformation are limited to those rows having the same x-session-key value as the output row. Therefore, you are considering only those input rows representing page views that occur during the same session as the output row.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>An operation that, for each output row, is applied to all of the input rows satisfying all of the conditions defined by the Input Condition, Key, Row Begin, Row End, Time Begin, and Time End parameters to produce an output: • ALL takes all of the values of the input field from the input rows and outputs them as a vector. • SUM interprets the values of the input field from the input rows as numbers and sums them. • FIRST ROW outputs the value of the input field from the first input row. • LAST ROW outputs the value of the input field from the last input row.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
<td></td>
</tr>
<tr>
<td>Row Begin/Row End</td>
<td>Optional. Specifies a range of input rows relative to the output row. For example, a Row Begin value of &quot;0&quot; excludes all rows before the output row. A row begin value of &quot;1&quot; excludes the output row as well. Common ranges include: • Begin 0: This row and all subsequent ones. • Begin 1: All subsequent rows. • End 0: This row and all previous ones. • End -1: All previous rows. • Begin -1, End -1: The previous row. • Begin 1, End 1: The next row.</td>
<td>All rows</td>
</tr>
</tbody>
</table>
The **CrossRows** transformation in this example is applied to rows of web data to find for each page view the time of the next page view. Because we know that **CrossRows** is applied only during the transformation phase of the dataset construction process, the rows of data are ordered by visitor (each visitor has a unique tracking ID) and time.

The input field, x-timestamp, is considered for only those input rows in which the x-is-page-view field is populated (indicating the row of data represents a page view). The x-session-key field (which has a unique value for each session) is specified for the Key parameter. Therefore, the input rows (log entries) for the transformation are limited to the contiguous block of rows having the same value of x-session-key as the output row. In other words, to be considered for the transformation, an input row must represent a page view that occurs during the same session as the page view in the output row. The first row operation takes the value of the output field from the first input row satisfying the **Input** Condition and having the same x-session-key value as the output row.

### CrossRows (Time of next pageview)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Begin/Time End</td>
<td>Optional. Specifies a range of times relative to the time of the output row. For example, a Time End of 30 minutes includes all rows that take place within 30 minutes after the output row. A Time Begin of -30 minutes includes all rows that take place within 30 minutes before the output row. Available time units are days, weeks, hours, minutes, ms (milliseconds), ticks (100 nanoseconds), and ns (nanoseconds). All times</td>
</tr>
</tbody>
</table>

**CrossRows** executes in an amount of time proportional to the size of its inputs plus the size of its outputs. This means that for operations SUM, FIRST ROW, and LAST ROW, it is no less efficient than other transformations. For ALL, the situation is more complex because it is possible to configure **CrossRows** to output an amount of data for each row of data (log entry) that is proportional to the total number of rows (log entries) for a given tracking ID.

**ExtractValue**

If you are working with web data, you can use the **ExtractValue** transformation to extract a value from a query string, cookie, or similarly encoded field in your website data.
Note that the name(s) corresponding to the value to be extracted can be different in each log entry.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Input Name</td>
<td>The name(s) of the field(s) to be extracted from the Input Query.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If the Input Name is a vector (that is, there are multiple names present), only one value is extracted.</td>
<td></td>
</tr>
<tr>
<td>Input Query</td>
<td>The encoded mapping (query string, cookie, and so forth) from which the value is to be extracted.</td>
<td></td>
</tr>
<tr>
<td>Output Value</td>
<td>The name of the field used to capture the extracted decoded value.</td>
<td></td>
</tr>
</tbody>
</table>

If you want to extract a search phrase, you can extract the entire phrase and, if desired, split the phrase into search terms using a **Tokenize** transformation. For information about the **Tokenize** transformation, see **Tokenize**.

This example configures an **ExtractValue** transformation to extract values of the `x-v-search-querynames` field from `cs(referrer-query)` and store them in the `x-search-phrase` field.

<table>
<thead>
<tr>
<th>ExtractValue (Search Phrase)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Search Phrase</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input Name</td>
<td><code>x-v-search-querynames</code></td>
</tr>
<tr>
<td>Input Query</td>
<td><code>cs(referrer-query)</code></td>
</tr>
<tr>
<td>Output Value</td>
<td><code>x-search-phrase</code></td>
</tr>
</tbody>
</table>

**Flatten**

The **Flatten** transformation takes a vector of strings and maps each value into its own field.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available for the log entry.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>A vector of string values to map to the output field names.</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>A set of output field names.</td>
<td></td>
</tr>
</tbody>
</table>
**Note: Considerations for Flatten**

- If the input vector contains more values than there are defined output fields, the extra input values are simply dropped.
- If the input vector contains fewer values than there are defined output fields, the extra output fields are given the default value (if defined) or an empty string if no default value is defined.

Here, the Flatten transformation is used to take a vector of products (x-products) and separate them into four fields (x-product1, ..., x-product4).

<table>
<thead>
<tr>
<th>-2</th>
<th>Flatten</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Products as Simple Strings</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>x-product1</td>
</tr>
<tr>
<td>1</td>
<td>x-product2</td>
</tr>
<tr>
<td>2</td>
<td>x-product3</td>
</tr>
<tr>
<td>3</td>
<td>x-product4</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Default</td>
<td>x-products</td>
</tr>
</tbody>
</table>

If the input value contained the strings B57481, C46355, and Z97123, the output fields would have the values shown here:

- x-product1 = B57481
- x-product2 = C46355
- x-product3 = Z97123
- x-product4 = Empty (There are more inputs than outputs, and there is no default value specified.)

**Format**

The Format transformation takes a set of inputs and formats them to create an output matching the given structure. The transformation works on either simple strings or string vectors and produces output by applying the given format to each input value until all of the input values have been transformed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Format</td>
<td>A formatting string used to specify how the output will look. %1% refers to a value from the first input vector, %2% refers to a value from the second input vector, and so on.</td>
</tr>
</tbody>
</table>
Fields containing either simple strings or string vectors. In the case of string vectors as inputs, the output will also be a string vector resulting from the application of the **Format** parameter to each set of input values.

*Note: The numbering of inputs starts at 0, but the numbering of the format substitution values starts at %1%.*

**Output**

The name of the field created to contain the results of the transformation. If the inputs are string vectors, the length of the output string vector will be the length of the longest input vector. If some of the input string vectors are of shorter length, empty strings are used for their position in the format string until the length of the output vector is reached.

In this example, two vectors, one a vector of strings representing product categories and the other a corresponding string vector representing the quantity of each product purchased, are transformed into a single vector of equivalent length that takes the form: Product %1%, Quantity %2%.

If the input vectors contained product categories of (683, 918) and quantities of (10, 4), the result would be one final output vector containing the following two strings: (“Product 683, Quantity 10”, “Product 918, Quantity 4”).

### Hash

The **Hash** transformation creates a nearly unique string representing a 64-bit number from the input values.

This transformation provides the same hash value when given the same inputs.

*Note: The resulting value is nearly unique because the transformation uses a 64-bit number as the space of possible hash values. For one million unique inputs to the hash transformation, there is a 1 in 38,000,000 chance of getting a duplicate hash value.*
The set of inputs to use to create the hash value.

Output: The name of the field for output.

In this example, the values of the c-ip and cs(user-agent) fields are used to create a tracking ID, which is stored in the x-trackingid field.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>The set of inputs to use to create the hash value.</td>
</tr>
<tr>
<td>Output</td>
<td>The name of the field for output.</td>
</tr>
</tbody>
</table>

Note: This example does not represent an ideal solution for creating unique tracking IDs. However, in situations in which archival log information is used, it may be the best method.

Haversine

In mathematics, the haversine formula is an equation that gives circle distances between two points on a sphere identified from their longitudes and latitudes.

Like the formula, the Haversine transform requires two sets of Latitude and Longitude settings, using these four inputs to calculate the true distance across the Earth between two locations.

This distance can be represented as miles or kilometers by changing the "In Kilometers" flag from false to true.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
</tbody>
</table>
The latitude of the point 1. Field
The latitude of the point 2. Field
The longitude of the point 1. Field
The longitude of the point 2. Field

Once calculated, the Output field contains distances between the points designated as elements in a Dimension.

As an example, if you code in a latitude and longitude of their store as Lat1, Lon1 and use an IP lookup lat and long for their customers, then distances to a store most customers buy from or come from can be determined.

Note: If you want to identify distances for other locations, then each individual location must have its own set of lat and lon fields.

IPLookup

The IPLookup transformation takes IP geo-location or IP geo-intelligence data (provided by any vendor of such data and converted into a proprietary format by Adobe) and transforms the data into geographical information that can be used in analysis.

Note:
Two IPLookup transformations are listed in the Add new > Transformation type menu:
• IPLookup Quova for IP geo-location data
• IPLookup Digital Envoy for IP geo-intelligence data

When defining an IPLookup transformation, choose the appropriate transformation for your IP geo-location or IP geo-intelligence data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Path and file name of the lookup file. Relative paths are with respect to the installation directory for the data workbench server. This file is typically located in the Lookups directory within the data workbench server installation directory.</td>
<td></td>
</tr>
<tr>
<td>IP Address</td>
<td>The field from which to read the IP address.</td>
<td>c-ip</td>
</tr>
<tr>
<td>Outputs</td>
<td>The names of the output strings.</td>
<td></td>
</tr>
</tbody>
</table>
The **IPLookup** Quova and **IPLookup** Digital Envoy transformations have different output parameters. Be sure to use the appropriate transformation for your IP lookup data.

In this example, **IP geo-location** data (in the lookup file `Quova.bin`) is used to create the output fields listed. The outputs (AOL, ASN, Area Code, and so on) can be used to create dimensions for geographical analysis of visitor traffic.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPLookup</strong></td>
<td>Quova and <strong>IPLookup</strong> Digital Envoy transformations have different output parameters. Be sure to use the appropriate transformation for your IP lookup data.</td>
<td></td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOL</td>
<td>x-q-asn</td>
<td></td>
</tr>
<tr>
<td>ASN</td>
<td>x-q-area-code</td>
<td></td>
</tr>
<tr>
<td>Area Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carrier</td>
<td>x-q-carrier</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>x-q-city</td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td>x-q-connection</td>
<td></td>
</tr>
<tr>
<td>Continent</td>
<td>x-q-continent</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td>x-q-country</td>
<td></td>
</tr>
<tr>
<td>DMA</td>
<td>x-q-dma</td>
<td></td>
</tr>
<tr>
<td>IP Routing Method</td>
<td>x-q-ip-routing</td>
<td></td>
</tr>
<tr>
<td>IP Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latitude</td>
<td>x-q-latitude</td>
<td></td>
</tr>
<tr>
<td>Line Speed</td>
<td>x-q-line-speed</td>
<td></td>
</tr>
<tr>
<td>Longitude</td>
<td>x-q-longitude</td>
<td></td>
</tr>
<tr>
<td>MSA</td>
<td>x-q-msa</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metro City</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metro Distance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metro State</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMSA</td>
<td>x-q-pmsa</td>
<td></td>
</tr>
<tr>
<td>Region</td>
<td>x-q-region</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second Level Domain</td>
<td>x-q-domain-2</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>x-q-state</td>
<td></td>
</tr>
<tr>
<td>Timezone</td>
<td>x-q-timezone</td>
<td></td>
</tr>
<tr>
<td>Top Level Domain</td>
<td>x-q-domain-1</td>
<td></td>
</tr>
<tr>
<td>Zip</td>
<td>x-q-zip</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Lookups\IP Geo-location\Quova.bin</td>
<td></td>
</tr>
<tr>
<td>IP Address</td>
<td>c-ip</td>
<td></td>
</tr>
</tbody>
</table>

**LookupRows**

The **LookupRows** transformation looks at other log entries with the same tracking ID and sets the value of the output field to the value of a designated field in the input row.
Because the LookupRows transformation performs its lookup on log entries and not lookup files, it is very similar to the CrossRows transformation. See CrossRows.

To work, the LookupRows transformation requires that the data is ordered in time and grouped by the tracking ID in your source data. Therefore, LookupRows works only when defined in the Transformation.cfg file or in a Transformation Dataset Include file.

As you review the descriptions of the parameters in the following table, remember the following:

- The output row is the row of data that the transformation is working on at a given point in time.
- Input rows are all of the other rows of data (before, after, or including the output row) whose values of the input field serve as inputs to the transformation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>Limits the output of the transformation to certain log entries. If the condition is not met for a particular log entry, the field in Output Row Value Output parameter is left unchanged. The input still may be used to affect other log entries.</td>
<td></td>
</tr>
<tr>
<td>Input Condition</td>
<td>Accepts input for the transformation from only certain input rows. If the Input Condition is not met for a particular input row, the input field from that row is ignored and does not affect other output rows. However, the output field from that row is still modified per the specified Condition.</td>
<td></td>
</tr>
<tr>
<td>Input Row Key Input</td>
<td>The name of the field to use as the key for the input rows.</td>
<td></td>
</tr>
<tr>
<td>Input Row Value Input</td>
<td>The name of the field in the input row whose value is copied to the field in the Output Row Value Output parameter if all conditions are satisfied.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>An operation that, for each output row, is applied to all of the input rows satisfying all of the conditions defined by the Input Condition and Input Row Key Input parameters to produce an output:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• FIRST outputs the value of the field in the Input Row Value Input parameter from the first matching input row in the data (not the first matching row after the output row).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• LAST outputs the value of the field in the Input Row Value Input parameter from the last input row in the data (not the last matching row before the output row).</td>
<td></td>
</tr>
<tr>
<td>Output Row Key Input</td>
<td>The name of the field to use as the key for the output row.</td>
<td></td>
</tr>
<tr>
<td>Output Row Value Output</td>
<td>The name of the field in the output row whose value is copied from the field in the Input Row Value Input parameter if all conditions are satisfied. All output rows with the same x-trackingid and Output Row Key Input values have the same Output Row Value Output value.</td>
<td></td>
</tr>
</tbody>
</table>

The Input Row Key Input, Input Row Value Input, and Input Condition parameters together define the lookup file for each tracking ID, while the Output Row Key Input, Output Row Value Input, and Condition parameters control what is looked up in the file and what value is stored in the field specified by Output Row Value Output.
To better understand the operation of the transformation, consider the following outline:

• For each output row satisfying the Condition and having a nonempty Output Row Key Input:
  • Find the FIRST or LAST input row such that
    • the input row satisfies the Input Condition, and
    • the x-trackingid of the input row equals the x-trackingid of the output row, and
    • the Input Row Key Input of the input row equals the Output Row Key Input of the output row,
  • and set the Output Row Value Output of the output row to the Input Row Value Input of the input row.

**Note:** Considerations for LookupRows

• Blank key values never match anything. Even if there are input rows with blank keys and nonblank values that match the Input Condition, an Output Row Key Input of "" will always produce an Output Row Value Output of "".
• If not forbidden by the Input Condition, a row may look up itself if its Input Row Key Input and Output Row Key Input values are the same.

If you have multiple key values, you can combine them using a Format transformation (see Format) before applying a LookupRows transformation.

Suppose that you have a website that has a pet registration page, where the name and the breed are entered, and a later "buy toy" page where only the name of the pet is used. You would like to be able to link the pet name with the pet breed entered on the registration page. To do so, you could create the following LookupRows transformation:

```
-1
| Name | LookupRows |
| Input Condition | Look up pet breed |
| -0 | AndCondition |
| | NonEmptyCondition |
| Input | cs-uri-query(petbreed) |
| Comments | Comment |
| Condition | AndCondition |
| Input Row Key Input | cs-uri-query(petname) |
| Input Row Value Input | cs-uri-query(petbreed) |
| Operation | LAST |
| Output Row Key Input | cs-uri-query(petname) |
| Output Row Value Output | x-pet-breed |
```

Let’s analyze this example using the previous outline:

• For each output row satisfying having a nonempty value of cs-uri-query(petname) :
  • Find the LAST input row such that
    • the input row contains a nonempty value of cs-uri-query(petbreed), and
    • the x-trackingid of the input row equals the x-trackingid of the output row, and
    • the value of cs-uri-query(petname) of the input row equals the value of cs-uri-query(petname) of the output row,
  • and set the value of x-pet-breed of the output row to the value of cs-uri-query(petbreed) of the input row.
The **LookupRows** transformation uses the pet name (the key) to make sure that the pet breed is linked to both the pet registration and buy toy pages so that you can analyze the toys bought for each breed of pet, even for visitors with multiple pets.

**Math**

The **Math** transformation enables the use of arithmetic operations on fields within the log entries.

The operations can include decimal integers and floating point constants.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>An arithmetic expression that describes the computation to be performed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>You can use any of the operations and functions listed below, and you can incorporate field names in the expression:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Operations</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Addition (+)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Subtraction (-)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Multiplication (*)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Division (/)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Remainder (%)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Exponentiation (^)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Functions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• sgn(x). Returns 1 if x is positive, 0 if x is zero, or -1 if x is negative.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• abs(x). Returns the absolute value of x.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• floor(x). Returns the greatest integer less than or equal to x.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• round(x). Returns the nearest integer to x.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• log(b,x). Returns the logarithm of x base b.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• min(x,y,...). Returns the smallest of all its arguments.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• max(x,y,...). Returns the largest of all its arguments.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the field containing the result of the arithmetic operation.</td>
<td></td>
</tr>
</tbody>
</table>

In this example, which uses fields of data collected from website traffic, a new field named x-page-duration is calculated by subtracting x-last-pv-timestamp from x-timestamp, then adding 1. The output is calculated only if the user-defined field x-last-pv-timestamp (which represents the timestamp of a visitor's last page view), is populated, or "not empty."
For information about the Not Empty condition, see Conditions.

Merge

The Merge transformation takes the values from the input field (typically a vector of strings), combines them into a single string separated by the given delimiter, and places the resulting string in the given output field.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>String that is used to separate the individual elements of the input string vector in the single output string.</td>
</tr>
<tr>
<td></td>
<td>If you hold down the Ctrl key and right-click within the Delimiter parameter, an Insert menu appears. This menu contains a list of special characters that often are used as delimiters.</td>
</tr>
<tr>
<td>Input</td>
<td>A vector of string values that are combined to form the output string.</td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output string.</td>
</tr>
</tbody>
</table>

In this example, an input vector of strings is assumed to contain a set of products that were selected for purchase. These products are placed into a single output string and are separated by "::" (two colons).
So if the input field x-products contained the string values B57481, C46355, and Z97123, the resulting output string x-show-products would be B57481::C46355::Z97123.

**PullNameValues**

The PullNameValues transformation is a special operation that takes the values in the cs-uri-query field and separates each of the name-value pairs into a separate string.

The entire collection of name-value pair strings is output in the specified output field as a vector of strings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available in the given log entry.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output string.</td>
<td></td>
</tr>
</tbody>
</table>

The PullNameValues transformation is used in this example to capture visitors' use of the search form: which buttons were selected, what values were typed in the form, and so on. The example uses a String Match condition (see Conditions) to isolate the use of this transformation to only the page /search.php. The vector of name-value pairs is output into the field x-search-namevalues.

```
-7
  Name  PullNameValues  Search Features and Values
  -Comments  Comment
  0
  -Condition  AndCondition
  -0
    -Matches  StringCondition
    0
    Case Sensitive  true
    Comments  cs-uri-stem
    Input  /search.php
    Default  x-search-namevalues
```

Using the transformation as defined above, if the cs-uri-stem field matched the page /search.php and cs-uri-query contained the following:

* Searchfor=Bob&State=Virginia&isMale=true

then x-search-namevalues would contain a vector containing the following three strings:

* Searchfor=Bob
* State=Virginia
* isMale=true
REMatch

The REMatch transformation is a pattern-matching transformation that uses regular expressions to specify one or more patterns to look for and capture in the input.

The transformation constructs an output field for each capturing sub-pattern in the regular expression. If the regular expression does not match the input field, the outputs are blank, and if the output field already exists, the values are replaced by the blank values. For a brief guide to using regular expressions, see Regular Expressions.

Note: The REMatch transformation operates similarly to the RETransform transformation (see RETransform), which uses regular expressions to capture a string and stores that string in a single output field.

REMatch parses a string more efficiently than multiple RETransform transformations or a single RETransform transformation followed by a Flatten transformation. See Flatten.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>True or false. Specifies whether the match is case-sensitive.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
</tr>
<tr>
<td>Expression</td>
<td>The regular expression used for matching.</td>
</tr>
<tr>
<td>Input</td>
<td>The field against which the regular expression is evaluated.</td>
</tr>
<tr>
<td>Outputs</td>
<td>The name of the output string or vector. In the case of string vectors as input, the outputs are also string vectors. An output field must exist for each capturing sub-pattern in the expression.</td>
</tr>
</tbody>
</table>

Note: REMatch transformations can be very slow and may account for much of the data processing time.

In this example, a REMatch transformation parses a date of the format YYYY-MM-DD into the fields x-year, x-month, and x-day. For the date 2007-01-02, the values of x-year, x-month, and x-day would be 2007, 01, and 02, respectively.

```plaintext
-2
<table>
<thead>
<tr>
<th>Name</th>
<th>REMatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>Parse yyyy-mm-dd date</td>
</tr>
<tr>
<td>0</td>
<td>x-year</td>
</tr>
<tr>
<td>1</td>
<td>x-month</td>
</tr>
<tr>
<td>2</td>
<td>x-day</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>false</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Expression</td>
<td>((\d+)-(\d+)-(\d+).*</td>
</tr>
<tr>
<td>Input</td>
<td>x-date</td>
</tr>
</tbody>
</table>
```
RETransform

The **RETransform** (regular expression) transformation is a pattern-matching transformation that uses regular expressions to specify a pattern to look for and capture in the input and stores the captured string in a designated output field.

Regular expressions are evaluated against the entire input string. If the input does not match the pattern specified in the regular expression, no data is captured. For a brief guide to using regular expressions, see *Regular Expressions*.

**Note:** The **RETransform** transformation operates similarly to the **REMatch** transformation (see **REMatch**), which constructs an output field for each capturing sub-pattern in the regular expression. You can think of **RETransform** as a combination of **REMatch** and **Format** transformations. If the Action parameter (see **Action** in the following table) is set to "RESULTS," then **RETransform** operates like a combination of **REMatch** and **Union** transformations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is either not available or the regular expression does not match the input value.</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>Specifies how the result is treated. The default setting of RESULTS simply takes the patterns matched and creates a vector of strings from the patterns being extracted. Alternatively, the action may be a formatting string to create a simple string output of a particular format. With this technique, you specify the number corresponding to the location of each matched pattern between % signs. For example, the 1st matched pattern would be %1%, and the 3rd matched pattern would be %3%. You would specify other characters in the formatting string literally.</td>
<td>RESULTS</td>
</tr>
<tr>
<td>Expression</td>
<td>The regular expression used for matching.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The field against which the regular expression is evaluated.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output string.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** **RETransform** transformations can be very slow and may account for much of the data processing time.

This example isolates the version of the Windows operating system that a website visitor is using and creates a field x-windows-version from that value. The output value in this case would simply be the version number.
If you wanted to include the string "Version" in front of the version number for readability, you would change the Action parameter from "RESULTS" to "Version %1%." To include a literal percent sign (%) in your output, escape it with a second percent sign, as in "%%."

Sessionize

If you are working with data collected from website traffic, you can use the Sessionize transformation to determine how sessions are defined.

The transformation takes as its input a timestamp and a tracking ID and outputs a session number for each log entry. The session number is "1" for the first session with a given tracking ID, "2" for the second session with the same tracking ID, and so on. The output can be used directly as a session key because it has a unique value for each session.

Note: To work, the Sessionize transformation requires that the data is ordered in time and grouped by the tracking ID in your source data. Therefore, Sessionize works only when defined in the Transformation.cfg file or in a Transformation Dataset Include file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Input Timestamp</td>
<td>The field containing the values of the timestamp to be used.</td>
<td>x-timestamp</td>
</tr>
<tr>
<td>Input Tracking ID</td>
<td>The field containing the values of the tracking ID to be used. The value must be a 64-bit (16 digit) or smaller hexadecimal number or a decimal integers of 16 digits or less.</td>
<td>x-trackingid</td>
</tr>
<tr>
<td>Maximum Session Duration</td>
<td>The longest length of session before a new session is started. (This keeps web pages that have auto content refreshing from creating sessions that are arbitrarily long.) If the Timeout Condition is satisfied and the referrer of a click is set to one of the entries in the Internal Domains parameter, Maximum Session Duration is</td>
<td>48 hours</td>
</tr>
</tbody>
</table>
Parameter | Description | Default
---|---|---
 | used to define the end of a session. No session may be longer than the specified Maximum Session Duration regardless of how many clicks it contains. The recommended value is 48 hours. For more information about the Maximum Session Duration and Internal Domains parameters, see Configuration Settings for Web Data. |  
Output Session Number | The field in which the session number is stored. This field has a unique value for each session for each visitor. |  
Session Timeout | The amount of time that needs to pass between log entries of a given visitor to determine the end of one session and the start of a new session (that is, the typical Session Timeout timeout used to define a user session). The recommended value of this parameter is 30 minutes. If the Timeout Condition is not satisfied and the referrer of a click is not set to one of the referrers in the Internal Domains parameter, Session Timeout is used to define the session.  
If the Timeout Condition is satisfied and cs(referrer-domain) for a log entry is in the list of internal domains, then Maximum Session Duration determines whether the current log entry is part of an existing session or the start of a new session.  
For more information about the Session Timeout parameter, see Configuration Settings for Web Data. | 30 minutes
Timeout Condition | The condition that must be satisfied for a log entry to be considered the start of a new session. Note that the amount of time that passes between the log entry and the previous log entry must be at least the value of the Session Timeout parameter. |  

A new session begins when any one of the following situations occurs:

- The tracking ID changes.
- The time since the last log entry is at least equal to the value of the Session Timeout parameter and the Timeout Condition is satisfied.
- The time since the first log entry of the last session exceeds the value of the Maximum Session Duration parameter.

💡 *Note:* If you have already defined Maximum Session Duration and Session Timeout as parameters in the `Session Parameters.cfg` file, do not enter values for them in the configuration. You can reference the parameters by typing `${parameter name}` as shown in the following example. For more information about these parameters, see Configuration Settings for Web Data.

The `Sessionize` transformation in this example takes as its input the x-timestamp and x-trackingid fields and records the session number for each log entry in the x-session-key field. The transformation's **Timeout Condition** is based on a **Neither** condition: If the cs(referrer-domain) field for a log entry matches a member of the Internal Domains parameter, the condition evaluates to false. Note the references to the Internal Domains and Session Timeout parameters.

For information about the **NeitherCondition**, see Conditions. For information about the Internal Domains and Session Timeout parameters, see Configuration Settings for Web Data.
The `Split` transformation splits a string into a vector of substrings based upon a given delimiter character. `Split` is particularly useful for extracting individual values from a collection of values associated with a single URI query name value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>String that is used to separate the input string into substrings. Must be a single character in length. If you hold down the Ctrl key and right-click within the Delimiter parameter, an Insert menu appears. This menu contains a list of special characters that often are used as delimiters.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field whose value is split to create the output string vector.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
<td></td>
</tr>
</tbody>
</table>

Consider a website in which the products purchased by a customer are listed as part of the cs-uri-query value when the confirmation page associated with a successful purchase is accessed. The following is an example of such a string:

```
* /checkout/confirmed.asp?prod_selected=B57481,C46355,Z97123
```

The cs-uri-stem field is used to determine whether the page being requested by the log entry is the confirmation page. The codes for the products that the customer purchased are listed as the comma-separated values of the `prod_selected` name in the cs-uri-query. The `Split` transformation can be used to extract this information by splitting the product codes at the comma if the value of cs-uri-stem matches the value specified in the `String Match` condition. See `String Match`. The following transformation details the solution to this problem.
Here, the output field is x-products, which would be used to create the desired extended dimension that maps the products purchased to the sessions during which the purchase was made.

**Tokenize**

The **Tokenize** transformation iteratively applies a regular expression against the input string.

However, unlike **RETransform**, **Tokenize** does not have to match the entire string: the regular expression used for the **Tokenize** transformation can match a subset of the input. After a match is found, **Tokenize** applies the regular expression again, starting at the character after the end of the last match.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>True or false. Specifies whether the match is case-sensitive.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is either not available or the regular expression does not match the input value.</td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>The regular expression used for matching.</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>The names of the output strings. You can have multiple outputs for a given input string. The number of outputs must correspond to the number of capturing sub-patterns in the regular expression.</td>
<td></td>
</tr>
</tbody>
</table>

In the following example, the **Tokenize** transformation uses a regular expression to capture the names of the query strings (in cs-uri-query) and output the captured sub-pattern (the query name) to x-pull-query-name.
For the query string "a=b&c=d," the output would be a vector containing "a" and "c."

For information about regular expressions, see *Regular Expressions*.

**Union**

The **Union** transformation takes a set of inputs and creates a vector of strings as the output.

If one of the inputs is itself a vector, each element in the input vector is associated with one element in the output vector (that is, the transformation does not create a vector of vectors).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>One or more input values.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the output field.</td>
<td></td>
</tr>
</tbody>
</table>

This example uses fields of data from website traffic to create a list of the zip codes associated with the website's visitors (that is, within each log entry). The data provides two possible sources for this information: one in cs-uri-query and the other in a `zipcode` field of the cookie. If neither of these fields contains a zip code, the default value of 00000 is used.

<table>
<thead>
<tr>
<th>-11</th>
<th>Union</th>
<th>Available Zip Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td>00000</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td>x-zipcode</td>
</tr>
</tbody>
</table>

While it is possible for both of these values to be available in a single log entry, you can select which value to use when you create a dimension based on the transformation's output. In a typical use case, you would create a simple
dimension that takes either the first or the last of the encountered values. For information about creating simple dimensions, see *Extended Dimensions*.

**URI Transformations**

The URI transformations modify the internal fields used to create the URI dimension provided with the Site application. These transformations enable you to modify the URI elements to create values that are more useful and meaningful to your site analysts. In general, these transformations have the same basic operational principles as the standard transformations. See *Standard Transformations*.

**AppendURI**

The AppendURI transformation provides a way to add information to the default value that comes from the log entries used to build the dataset.

The transformation places a name-value pair at the end of the internal field used to create the URI dimension. The name-value pair is built using the Query String Key parameter as the name and the value of the identified Input parameter as the value of the pair. The AppendURI command adds any appropriate ? and & symbols necessary to separate the name-value pairs from the URI stem and from any previous AppendURI operations that may have been applied to the URI.

The AppendURI transformation works only when defined in the Transformation.cfg file or a Transformation Dataset Include file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field whose value is appended to the URI.</td>
<td></td>
</tr>
<tr>
<td>Query String Key</td>
<td>The name to use in the creation of the name-value pair being appended.</td>
<td></td>
</tr>
</tbody>
</table>

Consider a website that was constructed using a traditional Model-View-Controller approach. In such systems, it is common to have a single web page be the point of access into the system. For such a site, visualizations of traffic patterns in the system would be very uninteresting and would provide no insights into visitor utilization and traffic flow. For example, consider a website that funnels all web requests through a URI of the following form:

• http://www.example.com/modelview.asp?id=login&name=bob

The modelview ASP page receives all traffic and determines its actions based on the value of the id field in the query. By default, the URI dimension would contain a single entry:

• modelview.asp

This would result in a rather uninteresting mapping of the traffic through the site, as all traffic is being funneled through a single URI. To address this particular scenario and provide a more informative view into the underlying architecture of the website, **AppendURI** can be used to move some of the unique name-value pairs from the
cs-uri-query field to the URI dimension used for visualizations. The transformation shown below gives the details of such a transformation:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-12</td>
<td>AppendURI</td>
<td>Unique Pages</td>
</tr>
<tr>
<td>Name</td>
<td>Comment</td>
<td>Append ID to make pages unique</td>
</tr>
<tr>
<td>Comments</td>
<td>AndCondition</td>
<td>StringCondition</td>
</tr>
<tr>
<td>Condition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Matches</td>
<td>/modelview.asp</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>/xmlmodelview.asp</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-stem</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-query(id)</td>
<td></td>
</tr>
<tr>
<td>Query String Key</td>
<td>id</td>
<td></td>
</tr>
</tbody>
</table>

In this example, there are two pages used by the system to handle all requests: modelview.asp and xmlmodelview.asp. One page is used for browser traffic, and the other is used for system-to-system XML communications. The application server process uses the id name of the cs-uri-query to determine which action to take. Therefore, you can extract the value from the id field and append it to the URI. The result is a collection of URIs with a range of variation that reflects visitor traffic through the website. Here, a String Match condition determines the log entries to which the transformation is applied by searching the cs-uri-stem field for the two web pages of interest and ignoring all others. The input (the value of our name-value pair) is the result of cs-uri-query(id), which is "login." As specified by the Query String Key parameter, the name being appended is "id." Thus, for the incoming cs-uri value of our example, the resulting URI used by the URI dimension is /modelview.asp&id=login.

PrependURI

Similar to the AppendURI transformation, the PrependURI transformation affects the internal field used by the data workbench server to construct the URI dimension.

The PrependURI transformation works by adding the value in the identified input field to the front of the value currently in the URI.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field whose value is prepended to the URI.</td>
<td></td>
</tr>
</tbody>
</table>

The following example simply prepends the s-dns field onto the URI, extending the representation of the URI dimension to include the domain requested by the client device.
In this example, prepending the s-dns field to the URI

*/modelview.asp?id=login

results in the following URL:

* www.adobe.com/modelview.asp?id=login

Now the URI is extended to include the domain requested.

ReplaceURI

The **ReplaceURI** transformation changes the value in the internal URI dimension to a new value.

If **URI Prefix** is specified, the resulting value is simply the URI prefix concatenated with the provided input value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The value to replace the URI.</td>
<td></td>
</tr>
<tr>
<td>URI Prefix</td>
<td>The value (string) to be prepended to the value in the <strong>Input</strong> field.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Before applying **ReplaceURI** transformations, you should create a new simple dimension with a parent of **Page View** from a copy of cs-uri-stem or cs-uri. For assistance with this, contact Adobe.

This example demonstrates the use of **ReplaceURI** to replace the "page=pageid" query strings with "homepage.html" whenever pageid indicates that the website's homepage was viewed. The end result is a more user-friendly view of the URI.

For the transformation shown, the page

* www.examplesite.com/info.html?page=1550

would be changed to

* www.examplesite.com/homepage.html

UnescapeURI

The **Unescape URI** transformation unescapes any characters in a string that have been escaped.
**Note:** Escaped characters replace the unsafe characters in a URI string. They are represented by a triplet consisting of a percent sign followed by two hexadecimal digits (for example, %20).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and the input value is not available.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The URI string to be unescaped.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the field in which the unescaped string is to be stored.</td>
<td></td>
</tr>
</tbody>
</table>

The following transformation unescapes the docname value in a HTTP header field and stores the output in the field x-docname-unescape:

```plaintext
<table>
<thead>
<tr>
<th>-0</th>
<th>URIUnescape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>cs(docname)</td>
</tr>
<tr>
<td>Output</td>
<td>x-docname-unescape</td>
</tr>
</tbody>
</table>
```

If the docname value were

- `mysite.net/lending%20and%20leasing%20forms/document%20library/credit%20application.doc`

then the value of x-docname-unescape would be

- `mysite.net/lending and leasing forms/document library/credit application.doc`

**DeviceAtlas Distribution**

The DeviceAtlas JSON file will now be distributed in a .bundle file (a renamed .tar.gz) along with DeviceAtlas.dll and DeviceAtlas64.dll files.

When the administrator upgrades the Insight Server to version 6.0, the DeviceAtlas.bundle file is included with the upgrade package in the Software and Docs profile (softdocs profile) located at:

Server Packages > v6.00 > Server_6.00.zip

The DeviceAtlas.bundle file is extracted to `Server\Lookups\DeviceAtlas`.

The DeviceAtlas.bundle file should be placed in a directory that is synchronized to the DPUs, and the DeviceAtlas.cfg file corresponding to the new DeviceAtlasComponent should be placed in the "Components for Processing Servers" directory on the synchronization master. When the DeviceAtlas.bundle file is changed, the very next DeviceAtlas lookup call will get results based on the updated API and/or JSON file.

**Modify the Transformation.cfg file**

The DeviceAtlas Transformations will no longer need to specify the path to the JSON file. Any previous DeviceAtlasTransformation that is defined in the transformation.cfg file should no longer include the File parameter that points to the obfuscated JSON file.
This example Transformation.cfg file shows the File argument that should be deleted to avoid confusion. (Leaving it there will not cause harm, but only potential confusion because it will be ignored.)

```csharp
6 = DeviceAtlasTransformation:
   Comments = Comment: 0 items
   Condition = AndCondition: 0 items
   File = string: Lookups\DeviceAtlas\20110106_private.json.obfuscated
   ^^^ DELETE THE ABOVE LINE FROM ALL PREVIOUS TRANSFORMATIONS ^^^
   Name = string: DeviceAtlas Lookup
   Outputs = vector: 4 items
     0 = Column:
        Column Name = string: vendor
        Field Name = string: x-vendor
     1 = Column:
        Column Name = string: model
        Field Name = string: x-model
     2 = Column:
        Column Name = string: isBrowser
        Field Name = string: x-isbrowser
     3 = Column:
        Column Name = string: usableDisplayHeight
        Field Name = string: x-usable-display-height
   User Agent = string: x-ua
```

Modify the DeviceAtlas.cfg file

This is an example of the component argument required in the DeviceAtlas.cfg file.

```csharp
component = DeviceAtlasComponent:
   DeviceAtlas Bundle File = string: Lookups\DeviceAtlas\DeviceAtlas.bundle
   Unsynchronized Bundle Extraction Path = string: Temp\DeviceAtlas\n```

This DeviceAtlas.bundle file will be treated just like a configuration file from the perspective of the Profile Synchronization feature. In addition, the JSON data and DLL will be used at the Component level rather than at the individual Transformation level.

A new DeviceAtlasComponent, upon startup, finds the .bundle conglomeration, de-obfuscates the JSON file into memory, extracts the files into a temporary directory, and loads the appropriate DLL for the running platform. This component also monitors changes to the bundle file, and reloads the DLL and .cfg file automatically if it changes.

Running DeviceAtlas

Proper configuration makes a big difference in the time required for transformation. The transformation can be configured to run only once per visitor per session to allow DeviceAtlas to speed up the process.

If deployed using Log Processing.cfg:

Run the transformations twice.

1. Look up only the mobile id field, then
2. Create conditions to ignore the mobile id and then look up the rest of the fields.

If deployed using Transformation.cfg:

Deploy as in Step 1 in Log Processing above, or use cross-rows to support a conditional setting.

• Cross-Rows—Grab the previous session key. Then identify if the current session key is different from the one found with cross-rows. If so, then the DeviceAtlas transformation will only run on one record per session.

Modifying In-Memory Cache

Overview
The `DeviceAtlas.bundle` file uses an in-memory cache to greatly improve the performance of lookups. By default, it will cache up to 100000 user-agents and their properties. The LRU cache is entirely self-contained inside the `DeviceAtlas.bundle` file, so any version of the server capable of using a bundle file will automatically benefit from the increased performance as soon as the new file is loaded.

**Modifying the `DeviceAtlas.cfg` file**

The maximum size of the LRU cache can be configured by modifying the `Cached Elements Limit` parameter in the `DeviceAtlas.cfg` file.

⚠️ *Note: This feature was added in the Data Workbench 6.4 release.*

You can override the default `Cached Elements Limit` of "100000" by changing its value and saving the file. The default value has been selected to be sufficient for most needs.

```plaintext
component = DeviceAtlasComponent:
DeviceAtlas Bundle File = string:
Lookups\DeviceAtlas\DeviceAtlas.bundle
Unsynchronized Bundle Extraction Path = string:
Temp\DeviceAtlas\Cached Elements Limit = unsigned int: 100000
```

**Integrating Lookup Data**

Data workbench provides a set of transformations that enables the data workbench server to incorporate lookup data into the dataset.

Lookup data is external data from corporate databases or lookup files that you can combine with event data to create the dataset. In general, you use lookup data to augment the event data from your log sources. Conceptually, you can think of using lookup data to populate event data records with additional columns of information.

When you use lookup data, you load the data into a memory-resident lookup table. A column in the table must contain a common key that also exists in the event data records. The data in the lookup table itself can be loaded from a flat file or from an ODBC data source. Lookup data can be incorporated into the dataset during the log processing or transformation phase of the dataset construction process.

To incorporate lookup data, you must first generate a lookup file or have the information needed to access an SQL database, then define one or more of the following transformations in the dataset configuration files for log processing and transformation.

To integrate lookup data into the dataset

1. Generate your lookup file. See *Populating the Lookup Table.*
2. Define one of the following types of transformations in the Transformations parameter in the appropriate dataset configuration file:
   - Categorize
   - FlatFileLookup
   - ODBC Lookup

⚠️ *Note: Note that the ODBC Lookup transformation works only when defined in the Transformation.cfg file or in a Transformation Dataset Include file.*
Populating the Lookup Table

If you use the Categorize or FlatFileLookup transformations, the lookup table is loaded in memory and populated from a flat file whose location you specify when you define the transformation.

The flat file you specify must meet the following requirements:

- Each line in the file must represent a row in the lookup table.
- Columns in the file must be separated by an ASCII delimiter. You may use any character that is not a line-ending character and does not appear anywhere within the event data itself. When you define the transformation, you specify which character has been used to delimit the columns in the flat file.

If you use an ODBC Lookup transformation, the lookup table is loaded into memory and populated from a table or view in an ODBC database that you specify. When you define the transformation, you must also specify the data source, user name, and password that the data workbench server must use to establish a connection to the database.

⚠️ Note: Lookup tables are loaded when the data workbench server initially begins constructing the dataset. Once established, lookup files are not meant to be changed. If you change the flat file or ODBC table that is used for the transformation phase, you are required to retransform the entire dataset. If you change a flat file that is used during the log processing phase, the new lookup data is applied to all new records that enter the dataset, but the changes are not applied retroactively.

Defining Lookup Transformations

Information about the transformations that you can use to incorporate lookup data into the dataset.

Note that not all types can be used during both phases of the dataset construction process.

- Categorize
- FlatFileLookup
- ODBC Lookup

Categorize

The Categorize transformation uses a two-column lookup table composed of pattern-string/value pairs. During this transformation, the data workbench server reads each event data record in turn and compares the contents of a designated field in the record to each of the pattern strings listed in the first column of the lookup table. If the designated field matches one of the pattern strings, the data workbench server writes the value (found in the second column) that is associated with that pattern string to a designated output field in the record.

The strings in the first column of the lookup table optionally can start with the ^ character and/or end in the $ character to force matching at the beginning and/or end. This transformation does not accept regular expressions for defining match conditions in the first column. If the input value is a vector of strings, each string is run through the transformation and the result(s) are appended to an output string vector.

A Categorize transformation is generally easier and faster than using a Regular Expression transformation to accomplish the same thing.

⚠️ Note: The substring test used in Categorize is case-sensitive unless otherwise specified using the Case Sensitive parameter.
Categorize

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>True or false. Specifies whether the substring test is case-sensitive.</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition test passes and no entry in the categorization file matches the input, or the input field is not defined in the given log entry.</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>String that is used to separate the columns in the lookup file. Must be a single character in length. If you hold down the Ctrl key and right-click within the Delimiter parameter, an <strong>Insert</strong> menu appears. This menu contains a list of special characters that often are used as delimiters.</td>
<td></td>
</tr>
<tr>
<td>Multiple Values</td>
<td>True or false. If true, when multiple rows in the file match the input, each match results in a value being appended to the output vector of strings. If false, only the first matching row in the file is used in the output. In the latter case, if the input is a vector, the output is also a vector of equivalent length. If the input is a simple string, the output is also a simple string.</td>
<td>false</td>
</tr>
<tr>
<td>File</td>
<td>Path and file name of the categorization file. Relative paths are with respect to the installation directory for the data workbench server. This file is typically located in the Lookups directory within the data workbench server installation directory.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>The categorization file matches its substrings against the value in this field to identify the matching row in the file.</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>The name of the field associated with the result.</td>
<td></td>
</tr>
</tbody>
</table>

**Note: Considerations for Categorize**

- Changes to lookup files in **Categorize** transformations defined in the **Transformation.cfg** file or in a **Transformation Dataset Include** file require retransformation of the dataset. Lookup files for **Categorize** transformations defined in the **Log Processing.cfg** file or a **Log Processing Dataset Include** file are not subject to this limitation. For information about reprocessing your data, see Reprocessing and Retransformation.

- **Categorize** transformations defined in the **Log Processing.cfg** file or a **Log Processing Dataset Include** file reload their lookup files whenever the lookup files change. Changes are not applied retroactively, but they apply to all log data read after the change takes place.

This example illustrates the use of the **Categorize** transformation to integrate lookup data with event data collected from website traffic. Suppose that a particular website has business sections, and there is a requirement to be able to look at and make comparisons based on traffic flow and value generated by the different sections. You can create a lookup file that lists the substrings used to identify these different sections.
The lookup file `Lookups/custommap.txt` contains the following table:

<table>
<thead>
<tr>
<th>Products/products/</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>^/sports/</td>
<td>Sports</td>
</tr>
<tr>
<td>^/news/</td>
<td>News</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

This categorization file maps anything containing the string "/products/" to the value "Products," anything starting with "/sports/" to the value "Sports," and anything starting with "/news/" to the value "News." The following categorization transformation uses the value in the cs-uri-stem field as the string within which we are looking for a matching substring. The result of the transformation is placed into the x-custommap field.

<table>
<thead>
<tr>
<th>-14</th>
<th>Categorize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Custom Site Mapping</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>,</td>
</tr>
<tr>
<td>File</td>
<td>lookups/custommap.txt</td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-stem</td>
</tr>
<tr>
<td>Multiple Values</td>
<td>false</td>
</tr>
<tr>
<td>Output</td>
<td>x-custommap</td>
</tr>
</tbody>
</table>

Assuming that the Multiple Values parameter is set to false, the example would produce the following values for x-custommap given the listed values for cs-uri-stem.

<table>
<thead>
<tr>
<th>cs-uri-stem</th>
<th>x-custommap</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sports/news/today.php</td>
<td>Sports</td>
</tr>
<tr>
<td>/sports/products/buy.php</td>
<td>Products</td>
</tr>
<tr>
<td>/news/headlines.php</td>
<td>News</td>
</tr>
<tr>
<td>/news/products/subscribe.php</td>
<td>Products</td>
</tr>
</tbody>
</table>

The output is based on the order of the substrings in the lookup file. For example, the cs-uri-stem `/sports/products/buy.php` returns "Products." Although the URI stem starts with "/sports/," the string "/products/" is listed before "/sports/" in the lookup file. If the Multiple Values parameter were set to true, there would be an additional value for x-custommap, as the last example would match two rows in the lookup table: Products and News.

**FlatFileLookup**

The FlatFileLookup transformation uses a lookup table composed of any number of columns and rows (although, recall that it resides in memory). During this type of transformation, the data workbench server reads each event data record in turn and compares the contents of a designated field in the record to each of the values in a designated column of the lookup table. If there is a match, the data workbench server writes one or more values from the matching row in the lookup table to one or more designated output fields in the event data record.
The lookup table used during this transformation is populated from a flat file whose location you specify when you define the transformation.

**FlatFileLookup**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and if no entry in the lookup file matches the input.</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>String that is used to separate the columns in the lookup file. Must be a single character in length. If you hold down the Ctrl key and right-click within the Delimiter parameter, an Insert menu appears. This menu contains a list of special characters that often are used as delimiters.</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Path and file name of the lookup file. Relative paths are with respect to the installation directory for the data workbench server. This file is typically located in the Lookups directory within the data workbench Server installation directory.</td>
<td></td>
</tr>
<tr>
<td>Header Row</td>
<td>True or false. Indicates that the first row in the table is a header row to be ignored in processing.</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td><strong>Column Name</strong> is the name of the column used for matching the input to the row(s) in the file. If Header Row is true, this can be the name of a column in the lookup file. Otherwise, this must be the zero-based column number to match against. <strong>Field Name</strong> is the name of the field used to locate the row in the lookup file.</td>
<td></td>
</tr>
<tr>
<td>Multiple Values</td>
<td>True or false. Determines whether a single value (a matching row) or multiple values should be returned (one for each matching row). <strong>Note:</strong> If Multiple Values is set to false, you must ensure that there are not multiple matches. When multiple matches occur, there is no guarantee which match will be returned.</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>A vector of column objects (results) in which each object is defined by column and field names. <strong>Column Name</strong> is the column from which the output value is obtained. If Header Row is true, this can be the name of a column in the lookup file. Otherwise, this must be the zero-based column number to match against. <strong>Field Name</strong> is the name of the field used to capture the output. Note that this can be a vector of results, one for each row identified in the case where the Multiple Values parameter is true.</td>
<td></td>
</tr>
</tbody>
</table>
Note: Considerations for FlatFileLookup

- Matching the input field to the lookup file is always case-sensitive.
- Changes to lookup files in FlatFileLookup transformations defined in the Transformation.cfg file or Transformation Dataset Include files require retransformation of the dataset. Lookup files for FlatFileLookup transformations defined in the Log Processing.cfg file or Log Processing Dataset Include files are not subject to this limitation. For information about reprocessing your data, see Reprocessing and Retransformation.
- FlatFileLookup transformations in the Log Processing.cfg file or Log Processing Dataset Include files reload their lookup files whenever the lookup files change. Changes are not applied retroactively, but they apply to all log data read after the change takes place.

This example illustrates the use of the FlatFileLookup transformation to integrate lookup data with event data collected from website traffic. Suppose that you want to isolate website partners that are routing traffic to the website and transform their partner IDs into more user-friendly names. You then can use the user-friendly names to create extended dimensions and visualizations that map more clearly to the business relationship than the site-to-site relationship used for routing traffic.

The example transformation searches the cs(referrer-query) field for the PartnerID name-value pair, and, if located, the lookup file Lookups\partners.txt is used to compare the PartnerID value against the values in the Partner column of the table. If a row is located, the output field x-partner-name is given the name from the PrintName column of the identified row.

<table>
<thead>
<tr>
<th>-15</th>
<th>Name</th>
<th>FlatFileLookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Comments</td>
<td>Comment</td>
<td>Renamed Referrer Partner</td>
</tr>
<tr>
<td>0</td>
<td>Input</td>
<td>Gets nice names for referrer partners</td>
</tr>
<tr>
<td></td>
<td>Column Name</td>
<td>Partner</td>
</tr>
<tr>
<td></td>
<td>Field Name</td>
<td>cs(referrer-query)(PartnerID)</td>
</tr>
<tr>
<td>Outputs</td>
<td>-0</td>
<td>Column</td>
</tr>
<tr>
<td></td>
<td>Column Name</td>
<td>PrintName</td>
</tr>
<tr>
<td></td>
<td>Field Name</td>
<td>x-partner-name</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>No Partner</td>
<td></td>
</tr>
<tr>
<td>Delimiter</td>
<td>[TAB]</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Lookups\partners.txt</td>
<td></td>
</tr>
<tr>
<td>Header Row</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>Multiple Values</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

If the lookup table contained the following information:

<table>
<thead>
<tr>
<th>ID</th>
<th>Partner</th>
<th>Started</th>
<th>PrintName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P154</td>
<td>Aug 21, 1999</td>
<td>Yahoo</td>
</tr>
<tr>
<td>2</td>
<td>P232</td>
<td>July 10, 2000</td>
<td>Microsoft</td>
</tr>
<tr>
<td>3</td>
<td>P945</td>
<td>Jan 12, 2001</td>
<td>Amazon</td>
</tr>
</tbody>
</table>
the following examples would transform as follows:

• If cs(referrer)(PartnerID) returned P232, the field x-partner-name would be given the value "Microsoft."
• If cs(referrer)(PartnerID) returned P100, the field x-partner-name would be given the value "No Partner."
• If cs(referrer)(PartnerID) returned nothing, the field x-partner-name would be given the value "No Partner" as specified by the Default parameter.

**ODBCLookup**

The **ODBCLookup** transformation operates like a **FlatFileLookup** transformation. The only difference is that the lookup table used during this transformation is populated from an ODBC database and not a flat file.

💡 **Note:** **ODBCLookup** transformations can be executed only during the transformation phase of the dataset construction process. When possible, Adobe recommends that you use the **FlatFileLookup** transformation instead of the **ODBCLookup** transformation. **FlatFileLookup** transformations are inherently more reliable because they do not depend on the availability of an outside system. Additionally, there is less risk that the lookup table is modified if it resides in a flat file that you control locally.

### ODBC Lookup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the transformation. You can enter any name here.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the transformation.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which this transformation is applied.</td>
<td></td>
</tr>
<tr>
<td>Data Source Name</td>
<td>A DSN, as provided by an administrator of the data workbench server machine on which the dataset is processed, that refers to the database from which data is to be loaded.</td>
<td></td>
</tr>
<tr>
<td>Database Password</td>
<td>The password to be used when connecting to the database. If a password has been configured for the DSN in the <strong>Data Source Administrator</strong>, this may be left blank. Any password supplied here overrides the password configured for the DSN in the <strong>Data Source Administrator</strong>.</td>
<td></td>
</tr>
<tr>
<td>Database User ID</td>
<td>The user ID to be used when connecting to the database. If a user ID has been configured for the DSN in the <strong>Data Source Administrator</strong>, this may be left blank. Any user ID supplied here overrides the user ID configured for the DSN in the <strong>Data Source Administrator</strong>.</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>The default value to use if the condition is met and no entry in the lookup file matches the input.</td>
<td></td>
</tr>
<tr>
<td>Input Column</td>
<td><strong>Column Name</strong> is the column name or SQL expression for the data that is matched against the input. <strong>Field Name</strong> is the name of the field containing the data to be looked up.</td>
<td></td>
</tr>
<tr>
<td>Multiple Values</td>
<td>True or false. Determines whether a single value (a matching row) or multiple values should be returned (one for each matching row).</td>
<td></td>
</tr>
</tbody>
</table>
### Parameter | Description | Default
--- | --- | ---
|  | **Note:** If Multiple Values is set to false, you must ensure that there are not multiple matches. When multiple matches occur, there is no guarantee which match will be returned.  
| Output Columns | A vector of column objects (results) where each object is defined by column and field names.  
| | **Column Name** is the name of or SQL expression for the column from which the output value is obtained. **Field Name** is the name of the field used to capture the output.  
| Table Identifier | An SQL expression that names the table or view from which data is to be loaded. A typical table identifier is of the form SCHEMA.TABLE.  

**Note:**

- The Data Source Name, Database User ID, Database Password, and Table Identifier parameters are the same as the parameters of the same names that are described for ODBC data sources. See ODBC Data Sources.
- Unlike ODBC data sources, ODBC Lookup transformations do not require an increasing ID column. See ODBC Data Sources. That is because the contents of the lookup table must not change in any way while the dataset is active. Changes in a lookup table or view can not be detected until retransformation occurs. For information about reprocessing your data, see Reprocessing and Retransformation.

Suppose that you want to convert outdated DNS records to the updated records. Both sets of records are stored in an SQL database. To perform this task, you would reference a lookup table that is generated from the database and replace the outdated DNS records.

Our example transformation searches the log entries for the s-dns field, and, if located, the lookup table VISUAL.LOOKUP is used to compare the s-dns entry against the entries in the **OLDDNS** column of the table. If a row is located in the table, the output field s-dns is given the updated DNS record entry from the **NEWDNS** column of the identified row.
<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Domain Name</td>
</tr>
<tr>
<td>-Input Column</td>
<td></td>
</tr>
<tr>
<td>Column Name</td>
<td>OLDDNS</td>
</tr>
<tr>
<td>Field Name</td>
<td>s-dns</td>
</tr>
<tr>
<td>-Output Columns</td>
<td></td>
</tr>
<tr>
<td>Column Name</td>
<td>NEWDNS</td>
</tr>
<tr>
<td>Field Name</td>
<td>s-dns</td>
</tr>
<tr>
<td>Comments</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Data Source Name</td>
<td>VSTest0</td>
</tr>
<tr>
<td>Database Password</td>
<td>visual</td>
</tr>
<tr>
<td>Database User ID</td>
<td>visual</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Multiple Values</td>
<td>false</td>
</tr>
<tr>
<td>Table Identifier</td>
<td>VISUAL_LOOKUP</td>
</tr>
</tbody>
</table>
## Extended Dimensions

Instructions to create extended dimensions and describe the types of extended dimensions that you can define for creation during the transformation phase of data set construction.

Derived dimensions make up another category of dimensions used by the Insight Server. As the name suggests, derived dimensions are created from existing extended dimensions or metrics. You do not define derived dimensions within a **Transformation Dataset Configuration** file as you do extended dimensions. Instead, you define them as individual `.dim` files within an inherited profile or a data set profile.

See *Extended Dimensions* for steps to create a derived dimension.

### About Extended Dimensions

The Insight Server (InsightServer64.exe) enables you to define custom dimensions from event data or lookup data. Any custom dimensions that you define are referred to as extended dimensions. You can use them to create visualizations, build extended metrics, or perform analysis to understand the operations and issues associated with your business channel. You can define several types of extended dimensions in the `Transformation.cfg` file or in `Transformation Dataset Include` files.

An extended dimension represents a relationship between log field values and a parent dimension. A parent dimension can be any user-defined countable dimension. See *Countable Dimensions*. You specify the parent when defining the dimension in the `Transformation.cfg` file or a `Transformation Dataset Include` file. A dimension's parent is the same as its level. For example, if you define a dimension with a parent of Session, then that dimension is a session-level dimension.

*Note:* The log field values can come from the inherent values available in the log (.vsl) files or other event data sources or from extended log fields created through the use of transformations.

To add an extended dimension to a visualization, you access it from the Extended list within the **Select Dimension** menu. For example, to add an extended dimension to a graph visualization, you would right-click within the workspace and click **Add Visualization** > **Graph** > **Extended** > `<dimension name>`. If you would like to organize the list of your extended dimensions within the data workbench interface, you can move the extended dimensions into subfolders that you create. See the Administrative Interfaces chapter of the *Data Workbench User Guide*. If you do this, the subfolders’ names also appear in the menu, as in Add Visualization > Graph > Extended > `<subfolder name>` > `<dimension name>`.

To see all the dimensions that have been defined for your dataset profile and the buffer size for each, open the **Detailed Status** interface in data workbench and click **Performance**, then **Dimensions** to expand the nodes. The buffer size, which controls query times, is expressed in MB. For more information about the **Detailed Status** interface, see the Server Administration and Installation guide.

### Defining Extended Dimensions

Steps to define extended dimensions.

1. While working in your dataset profile, open the **Profile Manager** and click **Dataset** to show its contents.
2. Open the `Transformation.cfg` file or the `Transformation Dataset Include` file in which you want to define the extended dimension.
(Recommended) To open a dataset include file, see Dataset Include Files.

Note: Adobe recommends defining extended dimensions in one or more new Transformation Dataset Include files. For more information, see Creating New Dataset Include Files.

To open the Transformation.cfg file, see Editing the Transformation Configuration File.

3. Right-click Transformations and click Add new > <Extended dimension type>.

4. Input the appropriate information for your extended dimension. For descriptions of the transformation types and information about their parameters, see the following sections:

   - Countable Dimensions
   - Simple Dimensions
   - Many-to-Many Dimensions
   - Numeric Dimensions
   - Denormal Dimensions
   - Time Dimensions

   For any extended dimension that you define, you can add one or more comment lines to the Comments parameter to further describe the dimension or add notes about its use. To add a comment, right-click the Comments label and click Add new > Comment Line.

5. After you have defined your extended dimension(s) in the configuration file, save the file locally and save it to your dataset profile on the data workbench server.

Types of Extended Dimensions

The Insight Server enables you to define countable, simple, many-to-many, numeric, denormal, and time dimensions for inclusion in your data set.

Each dimension type has a set of parameters whose values you edit to provide specific instructions for the Insight Server to create the dimensions during the transformation phase of data set construction.

While some of the parameters differ among the dimension types, all require the specification of a parent dimension (the Parent parameter). The parent dimension determines which log entries from the log sources are provided as input to the new dimension. In other words, the log entries that are associated with the elements of the parent dimension are the ones that are associated with the new dimension before any filtering is applied. The parent dimension also determines the new dimension's position within the data set's hierarchy, referred to as the data set schema. For information about the interface that shows the data set schema, see Dataset Configuration Tools.

After the Insight Server uses the parent dimension to determine which log entries should be considered in the creation of the dimension, it applies the specified condition(s) (the Condition parameter) to blank the log entries that do not satisfy the condition. The server then identifies the value of the specified input field (the Input parameter) for each log entry and applies the specified operation (the Operation parameter), if applicable.

Note: If a log entry does not satisfy an extended dimension's Condition, the Insight Server substitutes blank values for all of the fields in the log entry. The actual log entry still exists, and the specified Operation determines whether the blank value of the Input field is used.
Countable Dimensions

A countable dimension’s elements can be counted by the system.

Countable dimensions are typically used to create sum metrics, which return the count, or sum, of all the elements of the dimension. You can define countable dimensions to count instances such as reservation bookings or product orders. For example, you could define the countable dimension Orders whose elements (log entries corresponding to orders from your online store) could be counted. If you want to show a count of orders within a visualization, you would define the Orders sum metric, which can be evaluated over a dimension or have filters applied to it.

Countable dimensions can be parents of other dimensions or children of other countable dimensions.

💡 **Note:** If you need a dimension that only provides a count of something, you should use a numeric dimension with an operation of COUNT. See Numeric Dimensions.

Countable dimensions are defined by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as appears to the user in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which the input field contributes to the creation of the countable dimension. If specified, a condition restricts the set of log entries visible to the dimension and all of its children in the dataset schema.</td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>false</td>
</tr>
<tr>
<td>Key</td>
<td>Optional. The name of the field to use as the key. If you define this parameter, an element of the countable dimension exists for every combination of an element of the countable dimension’s parent and a distinct value of the field specified as the key. Each element of the countable dimension is required to relate to a contiguous set of log entries. Therefore, if the log entries are not ordered by the key, an element of the countable dimension is created each time the key field changes. To prevent this situation, Adobe recommends that you use a unique key which is contiguous in time order.</td>
<td></td>
</tr>
<tr>
<td>Parent</td>
<td>The name of the parent dimension. Any countable dimension can be a parent dimension. To make a dimension the top-level dimension in the dataset’s schema, set the parameter to “root.” The defined dimension becomes the root countable dimension for the dataset. For example, if you are working with Site, the Visitor dimension is the root countable dimension for your dataset.</td>
<td></td>
</tr>
</tbody>
</table>

💡 **Note:** Although your root countable dimension does not have to be associated with the tracking IDs in the data, Adobe recommends that you configure your dataset’s root countable dimension to use the tracking ID field (x-trackingid) as its Key. As a
result, each element of the root countable is associated with a unique value of x-trackingid, and all of the data about each element is grouped together. If you would like to configure your dataset differently, contact Adobe.

This example illustrates the definition of a countable dimension using event data collected from website traffic. The countable dimension counts the web campaign events within a given session. The assumption is that all email campaign resources are requested from the web server with "email=" as part of cs-uri-query. In the example, the number of times that the visitor responds to an email campaign during a given session is of interest, not the actual value of the cs-uri-query(email) field.

```plaintext
| -0 |
| Name       | Countable  
| Condition  | Campaign Event |
| -Matches   | AndCondition |
| 0          | RECondition  
| Case Sensitive | true         
| Comments   | Comment      
| Input      | cs-uri-query(email) |
| Comments   | Comment      
| Hidden     | false        
| Key        | Session      
| Parent     |              |
```

This example also illustrates the definition of a countable dimension using event data collected from website traffic, but it has a defined Key parameter. The Session countable dimension uses the x-session-key field as its key. (The x-session-key field is the output of the `Sessionize` transformation and has a unique value for each session.) Every unique combination of an element of the Visitor dimension (the parent) and the x-session-key field is an element of the Session dimension.

```plaintext
| -0 |
| Name       | Countable  
| Comments   | Session    
| Condition  | Comment    
| Hidden     | false      
| Key        | x-session-key |
| Parent     | Visitor    |
```

**Simple Dimensions**

A simple dimension has a one-to-many relationship with its parent countable dimension.

A simple dimension is always a child of a countable dimension. You can think of a simple dimension as a representation of a property of the elements in its parent dimension. For example, if you are working with web data, you could define the Visitor Referrer dimension, which is a simple dimension with a parent dimension of Visitor. It represents the first HTTP referrer for each visitor in the Visitor dimension. Each visitor in the Visitor dimension has
only one visitor referrer, but many visitors can have the same visitor referrer. Therefore, the Visitor Referrer dimension has a one-to-many relationship with the Visitor dimension.

Simple dimensions are defined by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as it appears in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which the relationship between the Parent and the input field's value should be created.</td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>The field of values that is related to the parent dimension (Parent).</td>
<td></td>
</tr>
<tr>
<td>Load File</td>
<td>Optional. A file of available values for the relationship. You use a load file when either of the following applies:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The values have a specific sort order that you want to preserve in the data workbench display. For example, you might want to create a Quarter dimension whose elements (the quarters of the year) always display in chronological order.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• You want to create place holders for values that may not be found in the data but need to appear in the data workbench display.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If a value is encountered that is not present in the file, it is added to the end of the values when viewed in data workbench.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Available operations are as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• FIRST NONBLANK: The first non-blank input value is used, regardless of whether it comes from the first log entry. If Input is a vector field, the first row in the vector for the relevant log entry is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• FIRST ROW: The value for the first log entry related to the parent dimension element is used, even if the input is blank. If Input is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• LAST NONBLANK: The last non-blank input value is used, regardless of whether it comes from the last log entry. If Input is a vector field, the first row in the vector for the relevant log entry is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• LAST ROW: The value for the last log entry related to the parent dimension element is used, even if the input is blank. If Input is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.</td>
<td></td>
</tr>
</tbody>
</table>
Note: If Operation yields no value or a blank value for a particular log entry, the corresponding element of the parent dimension will relate to the "None" element of the simple dimension.

You should specify an operation to ensure that the dimension is defined as intended.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>The name of the parent dimension. Any countable dimension can be a parent dimension.</td>
</tr>
</tbody>
</table>

This example illustrates the definition of a simple dimension using event data collected from website traffic and a load file.

Consider the example of a poll of site visitors’ favorite Girl Scout cookies. A web page captures this vote and returns it to the web server in the name-value pair favoritecookie. Only one vote per visitor is counted, but visitors can change their minds and vote again if desired. This is a one-to-many relationship: one visitor can have many votes, but each vote is associated with only one visitor. Therefore, the parent of the dimension is visitors (only one vote per visitor) and the operation is LAST ROW (so that they can change their minds and vote again).

Placeholders must exist for all types of cookies so that cookie types receiving no votes appears in the data workbench display. For these reasons, a load file has been defined that contains the list of cookie types that may be selected. This file's contents, saved in a file named `cookietypes.txt`, looks something like the following:

Animal Treasures
Caramel Delights
Lemon Pastry Creams
Peanut Butter Patties
Shortbreads
Thin Mints

The final dimension is defined as shown here:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Favorite Cookie</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Hidden</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-query(favoritecookie)</td>
</tr>
<tr>
<td>Load File</td>
<td>Lookups\cookietypes.txt</td>
</tr>
<tr>
<td>Maximum Elements</td>
<td>32768</td>
</tr>
<tr>
<td>Operation</td>
<td>LAST ROW</td>
</tr>
<tr>
<td>Parent</td>
<td>Visitor</td>
</tr>
</tbody>
</table>

Many-to-Many Dimensions

A many-to-many dimension has a many-to-many relationship with its parent countable dimension.
You can think of a many-to-many dimension as a representation of a set of values for each element in its parent dimension. For example, the many-to-many dimension Search Phrase is a Session-level dimension (it has a parent of Session). It represents the set of search phrases associated with each session in the Session dimension. A single search phrase can be used in any number of sessions, and a single session can include zero or more search phrases. Therefore, the Search Phrase dimension has a many-to-many relationship with the Session dimension.

Many-to-many dimensions are defined by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as it appears to the user in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which the relationship between the parent and the input field's value should be created.</td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>The value that is related to the parent dimension (Parent). If this field is a vector of strings, then each element of the vector has its own relationship with the parent.</td>
<td></td>
</tr>
<tr>
<td>Parent</td>
<td>The name of the parent dimension. Any countable dimension can be a parent dimension.</td>
<td></td>
</tr>
</tbody>
</table>

This example illustrates the definition of a many-to-many dimension using event data collected from website traffic. This many-to-many dimension, named "Selected Product," relates sessions to the products purchased by the visitor during that session. The x-products field contains a vector of values, each of which is associated with a page view, which, in turn, is associated with a session.

<table>
<thead>
<tr>
<th>-3</th>
<th>ManyToMany</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Selected Product</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Hidden</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>x-products</td>
</tr>
<tr>
<td>Maximum Elements</td>
<td>32768</td>
</tr>
<tr>
<td>Parent</td>
<td>Session</td>
</tr>
</tbody>
</table>

By creating such a transformation, you can create a visualization in data workbench that depicts the relationship between the selected product dimension and the number of sessions that involve each of the products.
**Numeric Dimensions**

A numeric dimension consists of ordered, numerical elements and has a one-to-many relationship with its parent countable dimension.

You can think of a numeric dimension as a representation of the numeric properties of the parent dimension's elements. For example, if you are working with web data, you could define the numeric dimension Session Revenue, which defines an amount of revenue, in dollars, for each session in the Session dimension. Each session has a single amount of associated revenue, but several sessions can have the same amount of associated revenue. Therefore, the Session Revenue dimension has a one-to-many relationship with the Session dimension.

Numeric dimensions are often used to define metrics that sum values, count occurrences of a condition, or locate a minimum or maximum value. For example, a metric named “Revenue” might be defined using the Session Revenue dimension: \( \text{sum}(	ext{Session Revenue}, \text{Session}) \). Defined this way, the Revenue metric would give the total amount of revenue for the selected sessions.

Numeric dimensions cannot be parents of other dimensions.

Numeric dimensions are defined by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as it appears in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Clip Values</td>
<td>True or false. Specifies whether the input value (after Operation) is to be clipped to be between the values of Min and Max. If Clip Values is true, the value is clipped to that range. If Clip Values is false, no value is returned for the element of the parent dimension.</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which the input field contributes to the creation of the numeric dimension.</td>
<td></td>
</tr>
<tr>
<td>Fixed Size</td>
<td>True or false. Controls the number of elements in a dimension (cardinality). If true, all elements from Min to Max are included in the dimension. If false, the dimension’s size grows as values are added.</td>
<td>false</td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>The value to use with the specified Operation or the input value for which you want to count occurrences. If this field is a vector of strings, the evaluation occurs for each element in the vector. So for example, a vector with length 3 and an Operation of COUNT adds 3 to the count.</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>Lower limit on the final dimension result.</td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>Upper limit on the final dimension result.</td>
<td>1e6</td>
</tr>
</tbody>
</table>
Available operations are as follows:

- **COUNT**: The total number of nonblank values in the input field across all log entries that meet the dimension's Condition is used. If the input field is a vector field, the total number of nonblank values in each log entry is counted.

- **FIRST NONBLANK**: The first non-blank input value is used, regardless of whether it comes from the first log entry. If input is a vector field, the first row in the vector for the relevant log entry is used. If the value is not a number, no value is used.

- **FIRST ROW**: The value for the first log entry related to the parent dimension element is used, even if the input is blank. If input is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.

- **LAST NONBLANK**: The last non-blank input value is used, regardless of whether it comes from the last log entry. If input is a vector field, the first row in the vector for the relevant log entry is used. If the value is not a number, no value is used.

- **LAST ROW**: The value for the last log entry related to the parent dimension element is used, even if the input is blank. If input is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.

- **SUM**: The total of all the numeric values in the input field across all log entries that meet the dimension's Condition is used. If there are no such log entries or no numeric values found, the numeric value 0 is used.

- **MIN or MAX**: The minimum or maximum numeric value found in the input field across all log entries that meet the dimension's Condition is used. If there are no such log entries or no numeric values, no value is used.

*Note: You should specify an operation to ensure that the dimension is defined as intended.*

---

**Parameter** | **Description** | **Default**
--- | --- | ---
Offset | See Scale in this table. | 0

**Operation**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>The name of the parent dimension. Any countable dimension can be a parent dimension.</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>To yield the ordinal value of the dimension, the result of Operation is transformed as follows: $$(\text{scale} \times \text{input}) + \text{offset}$$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

*Note: If Operation yields no value, or Clip Values is false and the value is not between Min and Max, no element of the numeric dimension is related to the element of the parent dimension.*

This example illustrates the definition of a numeric dimension using event data collected from website traffic. This numeric dimension, named "Ad View Counter," counts the number of times that the visitor sees an advertisement during a given session. The assumption is that all advertisement resources are requested from the web server with...
ad= as part of the cs-uri-query. In the example, the number of times (COUNT) that the visitor is presented with an advertisement is the value of interest, not the actual value in the field.

<table>
<thead>
<tr>
<th>Name</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip Values</td>
<td>false</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Fixed Size</td>
<td>false</td>
</tr>
<tr>
<td>Hidden</td>
<td>false</td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-query(ad)</td>
</tr>
<tr>
<td>Max</td>
<td>1e6</td>
</tr>
<tr>
<td>Min</td>
<td>0.0</td>
</tr>
<tr>
<td>Offset</td>
<td>0.0</td>
</tr>
<tr>
<td>Operation</td>
<td>COUNT</td>
</tr>
<tr>
<td>Parent</td>
<td>Session</td>
</tr>
<tr>
<td>Scale</td>
<td>1.0</td>
</tr>
</tbody>
</table>

### Denormal Dimensions

A denormal dimension has a one-to-one relationship with its parent countable dimension.

You would define a denormal dimension whenever the desired dimension contains a unique element for each element of its parent. For example, **EMail Address** is a denormal dimension with a parent of Visitor. Each Visitor has an email address, and each element in the EMail Address dimension is the email address of a single visitor. Even if two visitors have the same email address, the addresses are distinct elements of the EMail Address dimension.

You can use denormal dimensions in any table visualization, in detail tables, or to create filters. In addition, you can use denormal dimensions with the data workbench server's segment export functionality to export values of fields (such as **Tracking ID** or **EMail Address**) that have lots of values. Because any segment data that you want to export must be defined as a dimension within the profile, you must create a denormal dimension that stores the raw strings of the field's data.

💡 **Note:** When using a denormal dimension in a table or other visualization that expects a normal dimension, a derived denormal dimension is created automatically. The derived denormal dimension has a one-to-many relationship with the parent dimension.

For information about the detail table visualization and filters, see the Analysis Visualizations chapter in the *Data Workbench User Guide*. For information about segment export, see the Configuring Interface and Analysis Features chapter in the *Data Workbench User Guide*.

💡 **Note:** Denormal dimensions can be very expensive in query time and disk space. A denormal dimension with parent **Page View** and a 50-byte average input string could add 25 GB of data to the buffers in a typical, large dataset, equivalent to about 13 simple or numeric page view dimensions, or about 125 session level dimensions. Never add a denormal dimension to a dataset without a careful evaluation of the performance impact.

Denormal dimensions are defined by the following parameters:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as it appears in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>The conditions under which the relationship between the Parent and the input field's value should be created.</td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>true</td>
</tr>
<tr>
<td>Input</td>
<td>The value that is related to the parent dimension (Parent).</td>
<td></td>
</tr>
<tr>
<td>Normalized Elements</td>
<td>A performance tuning parameter that specifies the number of dimension elements whose names are to be stored in system memory. Setting this parameter to a higher value causes a denormal dimension to use more RAM but results in faster queries. The default value is 16383.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Available operations are as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>FIRST NONBLANK</strong>: The first non-blank input value is used, regardless of whether it comes from the first log entry. If <strong>Input</strong> is a vector field, the first row in the vector for the relevant log entry is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>FIRST ROW</strong>: The value for the first log entry related to the parent dimension element is used, even if the input is blank. If <strong>Input</strong> is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>LAST NONBLANK</strong>: The last non-blank input value is used, regardless of whether it comes from the last log entry. If <strong>Input</strong> is a vector field, the first row in the vector for the relevant log entry is used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>LAST ROW</strong>: The value for the last log entry related to the parent dimension element is used, even if the input is blank. If <strong>Input</strong> is a vector field, the first row in the vector for the relevant log entry is used. If this value is blank or not a number, or if the relevant log entry does not meet the dimension's Condition, no value is used.</td>
<td></td>
</tr>
</tbody>
</table>

💡 **Note:** *If Operation yields no value, a blank value ("\"\") is used.*

You should specify an operation to ensure that the dimension is defined as intended.

| Parent        | The name of the parent dimension. Any countable dimension can be a parent dimension.                                                                                                                                                                                                                                                      |         |

The denormal dimension shown in this example takes all of the data in the field x-trackingid as input and includes it in a dimension named Visitor ID. For a segment of visitors that you have created, you can export the data in the Visitor ID dimension (as well as any other defined dimension).
Time Dimensions

A time dimension enables you to create a set of periodic or absolute local time dimensions (such as Day, Day of Week, Hour of Day, Reservation Time, and so on) based on any timestamp field that you specify for the Input Time (1970 epoch) parameter.

When defining time dimensions, you also can choose a day other than Monday to be used as the start of a week by specifying the Week Start Day parameter. You can define more than one set of time dimensions in your dataset as long as the dimensions have different names.

Time dimensions are defined by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Descriptive name of the dimension as it appears in data workbench. The dimension name cannot include a hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the extended dimension.</td>
<td></td>
</tr>
<tr>
<td>Dimensions</td>
<td>You can specify dimension names for any of the following periods:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Day</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Day of Week</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Hour</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Hour of Day</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Month</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Week</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The names that you enter here are the names that appear in dimension menus and in visualizations in data workbench. If you leave the name of a time dimension blank, the dimension is not created in the dataset.</td>
<td></td>
</tr>
<tr>
<td>Hidden</td>
<td>Determines whether the dimension appears in the data workbench interface. By default, this parameter is set to false. If, for example, the dimension is to be used only as the basis of a metric, you can set this parameter to true to hide the dimension from the data workbench display.</td>
<td>true</td>
</tr>
<tr>
<td>Input Time (1970 epoch)</td>
<td>The name of the timestamp field to use as input.</td>
<td></td>
</tr>
</tbody>
</table>

Note: The field's values must represent the number of seconds since January 1st, 1970, at 00:00:01. If the input time is not a valid time (1970 to 2037), the
**Parameter** | **Description** | **Default**
---|---|---

Transformation process will fail, and the data workbench server will generate an error.

Parent | The name of the parent dimension. Any countable dimension can be a parent dimension. For web data, the parent is Session.

Week Start Day | The day to use as the first day of a week. This parameter affects the Week dimension, the Day of Week dimension, and any reporting time dimensions defined in terms of weeks. | **Mon**

This example creates a set of time dimensions based on the user-defined input field x-time-1970. The set of time dimensions is named "Session Time." Because the parent of each dimension is the Session dimension, each element of the time dimensions corresponds to the time at which a session began. The Week Start Day parameter specifies that each week of the Week dimension starts on Monday.

```
Extended Dimensions
-0
   Name: Session Time
   Dimensions: map
     Day
     Day of Week
     Hour
     Hour of Day
     Month
     Week
   Comments: Comment
   Hidden: false
   Parent: Session
   Week Start Day: Mon
```
Transform Functionality

Information about transformation functionality and the steps to configure the server to export data to external targets.

About Transformation Functionality

Transform functionality (Transform) runs on a data workbench server machine to enable the export of log source data for use by other applications.

Transform can read .vsl files, log files, XML files, and ODBC data and export the data as .vsl files, text files, or delimited text files that can be used by DataWarehouse loading routines, auditing agencies, or other targets. The data extraction and transformation can be performed on a continuous or other scheduled basis.

Note: Typically, Transform is configured on a data workbench server FSU. However, your implementation may require configuration on an data workbench server DPU. For more information, contact Adobe.

You can view memory usage information for Transform in the Detailed Status interface. For more information, see the Administrative Interfaces chapter of the Data Workbench User Guide.

The segment export feature provides another means of exporting data from an Adobe application. Transform enables you to export unprocessed data to an external target, but the segment export feature enables you to output processed data from the dataset and requires that the exported data be defined as a dimension in the dataset. For more information about segment export, see the Data Workbench User Guide.

Configuration Files for Transform

Transform functionality includes three configuration files that contain the required parameters for reading and exporting data.

• The Transform.cfg File
• The Transform Mode.cfg File
• The Required Field Definitions.cfg File

Note: Before you begin to configure your implementation of transformation functionality, please confirm that you have properly installed the data workbench Transform.cfg and data workbench Transform Mode.cfg files and all of the Adobe software products and applications needed for your implementation. For installation instructions, see the Server Products Installation and Administration Guide.

The Transform.cfg File

The data workbench Transform.cfg file contains the parameters that define the log sources, data transformations, and exporters.

The transformations that you define manipulate raw data collected by Sensors (.vsl files) or contained in text files, XML files, or ODBC-compliant databases and output them either into existing fields, overwriting the current data, or into newly defined fields.

To configure transformation functionality, you edit the data workbench Transform.cfg file within the Dataset folder for the profile for which you want to export event data. Typically, this profile is dedicated to transformation functionality (that is, you perform no other data processing than what is defined in the data workbench Transform.cfg file). It
is important to note that any processing instructions specified in the Log Processing Dataset Include files for any inherited profiles are applied in addition to those specified in the data workbench Transform.cfg file.

For information about dataset include files, see Dataset Include Files.

If the data that you want to export is processed by a data workbench server cluster, each of the processing servers (DPUs) in the cluster processes the data, but only the first DPU (processing server #0 in the profile.cfg file) will write the output data to its local file system.

To edit the data workbench Transform.cfg file

1. While working in the profile for which you want to export data, open the Profile Manager and click Dataset to show the contents of the directory.
2. Right-click the check mark next to data workbench Transform.cfg, then click Make Local. A check mark for this file appears in the User column.
3. Right-click the newly created check mark and click Open > from the workbench. The data workbench Transform.cfg window appears.
4. Edit the parameters in the configuration file using the table below as a guide:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Time</td>
<td>Optional. Filter data to include log entries with timestamps up to, but not including, this time. Adobe recommends using one of the following formats for the time:</td>
</tr>
<tr>
<td></td>
<td>• January 1 2013HH:MM:SS EDT</td>
</tr>
<tr>
<td></td>
<td>• Jan 1 2013 HH:MM:SS GMT</td>
</tr>
<tr>
<td></td>
<td>For example, specifying July 29 2013 00:00:00 EDT as the End Time includes data through July 28, 2013, at 11:59:59 PM EDT.</td>
</tr>
<tr>
<td></td>
<td>You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by the data workbench server, see Time Zone Codes.</td>
</tr>
<tr>
<td></td>
<td>The Use Start/End Times parameter for Sensor and log file sources is related to this parameter.</td>
</tr>
<tr>
<td>Exporters</td>
<td>The subfields of an exporter specify how the output data is processed and/or formatted. You can define multiple exporters for a set of log sources. Each exporter type creates output independently.</td>
</tr>
<tr>
<td></td>
<td>Three types of exporters exist:</td>
</tr>
<tr>
<td></td>
<td>• ExportTextFile</td>
</tr>
<tr>
<td></td>
<td>• ExportDelimitedTextFile</td>
</tr>
<tr>
<td></td>
<td>• ExportVSLFile</td>
</tr>
<tr>
<td></td>
<td>For more information about exporter types, see Defining Exporters.</td>
</tr>
<tr>
<td>Hash Threshold</td>
<td>Optional. A sampling factor for random sub-sampling of rows. If set to a number n, then only one out of each n tracking IDs are selected for exporting, reducing the total number of rows exported by a factor of n. To export all rows, you would set Hash Threshold to 1.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Log Entry Condition</td>
<td>Optional. Defines the rules by which log entries are considered for export. For more information about the Log Entry Condition, see Log Processing Configuration File.</td>
</tr>
<tr>
<td>Log Sources</td>
<td>The sources of data. Log sources can be .vsl files, log files, or XML files or data from ODBC-compliant databases. For information about log sources, see Log Processing Configuration File. Transform expects all source data to be in chronological order within lexicographically ordered input files. If this requirement is not satisfied, As Of calculations are incorrect, and additional input data may be processed after the output files are closed.</td>
</tr>
<tr>
<td>Offline Mode</td>
<td>Optional. True or false. If true, Transform assumes that all of the input files are present when it starts processing the data. When all of the input data has been read, Transform closes all of the output files without waiting for additional data to be received. The default value is false.</td>
</tr>
<tr>
<td>Note: Offline Mode</td>
<td>If Offline Mode is set to true, Transform expects all source data to be present before processing starts. A warning message is generated in the VisualServer.log file if additional data is received after the output files are closed.</td>
</tr>
<tr>
<td>Reprocess</td>
<td>Optional. Any character or combination of characters can be entered here. Changing this parameter and saving the file to the Transform machine initiates data reprocessing. For information about reprocessing your data, see Reprocessing and Retransformation.</td>
</tr>
<tr>
<td>Stages</td>
<td>Optional. The names of the processing stages that can be used in Log Processing Dataset Include files that are executed in addition to the data workbench Transform.cfg file. Processing stages provide a way to order the transformations that are defined in Log Processing Dataset Include files. This parameter is very helpful if you have defined one or more transformations within multiple Log Processing Dataset Include files and you want specific transformations to be performed at specific points during the export process. The order in which you list the stages here determines the order in which the transformations in the Log Processing Dataset Include files are executed during data export. Preprocessing and Postprocessing are built-in stages; Preprocessing is always the first stage, and Postprocessing is always the last stage. By default, there is one named stage called Default. To add a new processing stage • In the data workbench Transform.cfg window, right-click Stages, then click Add New Stage. • Enter a name for the new stage. To delete an existing processing stage • Right-click the number corresponding to the stage that you want to delete and click Remove &lt;#stage_number&gt;.</td>
</tr>
</tbody>
</table>
### Parameter  | Description
---|---

**Note:** When you specify a Stage in a Log Processing Dataset Include file the name of the stage must match exactly the name that you enter here. For more information about dataset include files, see Dataset Include Files.

**Start Time**  
Optional. Filter data to include log entries with timestamps at or after this time. Adobe recommends using one of the following formats for the time:

- January 1 2013 HH:MM:SS EDT
- Jan 1 2013 HH:MM:SS GMT

For example, specifying July 29 2013 00:00:00 EDT as the Start Time includes data starting from July 29, 2013, at 12:00:00 AM EDT.

You must specify a time zone. The time zone does not default to GMT if not specified. For a list of time zone abbreviations supported by the data workbench server, see Time Zone Codes.

**Note:** The Use Start/End Times parameter for Sensor and log file sources is related to this parameter.

**Transformations**  
Optional. Defines the transformations that are to be applied to the data. For information about the available transformation types, see Data Transformations.

**Note:** The following transformation types do not work when defined in the data workbench Transform.cfg file:

- AppendURI
- CrossRows
- ODBCLookup
- Sessionize

**Note:**

If additional data is received after the output files are closed (see Log Sources and Offline Mode in the preceding table), Transform creates new output files with the additional data. The names of the new output files are generated from the original output file's name with the addition of a parenthesized version number just before the extension. For example, if the original output file is 20070701-ABC.vsl, subsequent versions of this file will be named 20070701-ABC(1).vsl, 20070701-ABC(2).vsl, and so on. Note that using the versioned files as input to the data workbench server may result in processing errors.

Adobe recommends avoiding the creation of versioned output files by making sure that all source data is in chronological order within lexicographically ordered input files and, if Offline Mode is set to true, that all source data is present before processing starts. For more information, see the Log Sources and Offline Mode entries in the preceding table.
5. Add transformations by right-clicking Transformations and clicking Add new > Transformation type. Complete the transformation fields.

See Data Transformations for descriptions and examples of the transformations that you can use with transformation functionality.

6. Right-click (modified) at the top of the window, then click Save.

7. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for data workbench Transform.cfg in the User column, then click Save to > profile name, where profile name is the name of the profile for which you are exporting data. Reprocessing of the data begins after synchronization of the profile.

   Note: For information about reprocessing your data for export, see Reprocessing and Retransformation.

Defining Exporters

Exporters provide the instructions for outputting the event data.
Transformation functionality provides three types of exporters for exporting vsl files, log files, XML files, and ODBC data as vsl files, text files, or delimited text files that can be used by DataWarehouse loading routines, auditing agencies, or other targets.

   Note: For an exporter to work properly, the log source must meet the appropriate requirements discussed in the Log Sources section of Log Processing Configuration File.

To define an exporter

1. Open Transform.cfg in data workbench. See To edit the Insight Transform.cfg file.
2. Right-click Exporters, then click Add New.
3. Select one of the following options:
   - ExportTextFile
   - ExportDelimitedTextFile
   - ExportVSLFile

   Note: For the ExportVSLFile option, all of the extended fields in the input file and all user-defined fields of the form cs(header) are always written to the VSL output file. If you overwrite an existing extended field, the new value is written to the output file, even if the field is blank.

4. Edit the Exporters parameters in the configuration file using the following table as a guide:

   Exporter Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Format</td>
<td>For ExportTextFile only. The format of each output line, consisting of field name escapes (expressed as %fieldname%) and any other desired fixed text. The format should include a line separator, typically [CR] [LF]. A literal percent sign (%) can be embedded in the format string by escaping the character as shown here: %%</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>An example of an entry for the Data Format parameter is <code>%x-timestring%</code> <code>%x-trackingid%[CR][LF]</code>.</td>
</tr>
<tr>
<td>Fields</td>
<td>For <strong>ExportDelimitedTextFile</strong> only. Names of the fields to be output.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>Optional. For <strong>ExportDelimitedTextFile</strong> only. Character that is used to separate the fields in the output file.</td>
</tr>
<tr>
<td></td>
<td>The software can not escape delimiters that are included in the data's values. As a result, Adobe does not recommend using commas as delimiters.</td>
</tr>
<tr>
<td></td>
<td>If you hold down the Ctrl key and right-click within the Delimiter parameter, an Insert menu appears. This menu contains a list of special characters that often are used as delimiters.</td>
</tr>
<tr>
<td>Line Separator</td>
<td>Optional. For <strong>ExportDelimitedTextFile</strong> only. The character(s) used to separate lines in the output files. The default value is [CR] [LF].</td>
</tr>
<tr>
<td>Name</td>
<td>Optional. Identifier for the exporter. This name appears in the Detailed Status interface.</td>
</tr>
<tr>
<td></td>
<td>For information about the Detailed Status interface, see the Data Workbench User Guide.</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the exporter.</td>
</tr>
<tr>
<td>Output Path</td>
<td>Path where output files are to be stored. The path is relative to the data workbench server installation folder.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The data workbench server that stores output data is processing server #0 in the <code>profile.cfg</code> file.</td>
</tr>
<tr>
<td>File Rotation Period</td>
<td>Optional. The frequency at which data is exported to the output file. Each output file contains data related to a specific time period called the rotation period. All time calculations are in GMT: One day starts at midnight GMT and ends the following day at midnight GMT, even if the data written to the file includes a field which has been transformed to local time. Available values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• YEAR. Each file contains data for one calendar year.</td>
</tr>
<tr>
<td></td>
<td>• MONTH. Each file contains data for one calendar month. Months are numbered 1 (January) through 12 (December).</td>
</tr>
<tr>
<td></td>
<td>• WEEK. Each file contains data for one week. A week starts on Monday. The week starting on one of the first seven days of the year is week 1, and the prior (partial) week, if any, is week 0.</td>
</tr>
<tr>
<td></td>
<td>• DAY. Each files contains data for one calendar day.</td>
</tr>
<tr>
<td></td>
<td>• HOUR. Each file contains data for one hour.</td>
</tr>
<tr>
<td></td>
<td>• NONE. No rotation is performed. All of the data is written to the same file (or a set of files as determined by other parameter settings). See the <strong>File Name Format</strong> parameter in this table.</td>
</tr>
<tr>
<td></td>
<td>The default file rotation period is DAY.</td>
</tr>
</tbody>
</table>
### Description

**Note:**
- Set the file rotation to NONE only when working in **Offline Mode**. See the **Offline Mode** parameter description.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| File Name Format              | Optional. The format of the output file name. Each log entry can be stored in a file whose name is derived from the start time of the rotation period, and optionally, from values of fields in the rows it contains. The fields to use in the file name are embedded as field name escapes (expressed as `%fieldname`). The file name components related to the rotation period are embedded in the format string using the following escape sequences:  
  - `%yyyy%` (four-digit year)  
  - `%yy%` (two-digit year)  
  - `%mm%` (two-digit month, 01 - 12)  
  - `%ww%` (two-digit week, 01 - 52)  
  - `%dd%` (two-digit day, 01 - 31)  
  - `%HH%` (two-digit hour, 00 - 23)  
  
  The default file name format is `%yyyy%mm%dd%-x-mask%.txt`

  **Note:**
  - The escape sequences are case-sensitive.
  - When File Rotation Period is set to NONE, an empty string is substituted for each of the escape sequences, if present.
  - An error is generated if **File Name Format** does not result in a unique file name for each rotation period (see the File Rotation Period parameter in this table). For example, when using the DAY rotation period, the `%dd%`, `%mm%`, and `%yy%` or `%yyyy%` escape sequences must be present in the pattern to avoid data loss.
  - If you are using field name escape sequences within the pattern and the given field has many distinct values, many output files are written for each rotation period. Note that this scenario may result in poor performance, so you should use this feature with caution.
  - All times calculations are in GMT.

| Execute at Rollover           | Optional. When each file is finalized, an external (Windows) command can be executed. The command line is derived from the final file name by substituting the following escape sequences into this parameter:  
  - `%dir%`. The directory part of the final file name, including the trailing backslash.  
  - `%file%`. The file name (excluding the directory and extension) of the final file.  
  - `%ext%`. The extension (including the leading ".") of the final file name.  
  - `%path%`. The full path name of the file, equivalent to `%dir%file%ext%`.  
  
  By default, this parameter is empty (no command is executed). |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Limit</td>
<td>Optional. The amount of memory in bytes used for buffering the exporter's output. The default value is 10,000,000 bytes.</td>
</tr>
<tr>
<td>Note: If you have many output files that are open at the same time, you may want to increase this value, but you may decrease the amount of memory available for use by other components of the system. Decreasing this value, however, may slow down the export process. For assistance, contact Adobe.</td>
<td></td>
</tr>
<tr>
<td>Open Files Limit</td>
<td>Optional. The maximum number of files that may be open at the same time for output from the exporter. If this number is exceeded, an error is recorded in the event log and the data workbench server stops running. The default value is 1000.</td>
</tr>
</tbody>
</table>

5. After you have defined your exporter (and made changes to other parameters) in the Transform.cfg file, save the file locally and save it to the appropriate profile on the data workbench server machine.

Sample Data Workbench Transform.cfg Files
Information about how to specify parameters in the Transform.cfg file based on the several scenarios.

- A Simple Data Workbench Transform.cfg File
- Output with Comma-Separated Values
- Sample Log Files
- Splitting Log Files by Web Site Section

In each sample, the file is displayed as a Transform.cfg window in data workbench.

A Simple Data Workbench Transform.cfg File
The following Transform.cfg window provides instructions to read .vsl files from the Logs directory and export the x-timestring and x-trackingid fields to a text file stored in the Logs\VT directory. Because no file rotation period or output file name format is specified, each file contains data for one calendar day and has a name in the default format %yyyy%mm%dd-%x-mask%.txt.

```
-Visual Transform.cfg
  -Visual Transform
    -Exporters
      -0
    Data Format
      Output Path
        Logs\VT
    -Log Sources
      -0
      -Log Paths
        0
```

```bash
  VisualTransform
    ExportTextFile
      %x-timestring%x-trackingid[CRLF]Logs\VT
    VisualSensor
      Logs\*.vsl
  ```
The following Transform.cfg window provides instructions to read .vsl files from the Logs directory and export fields 0 through 13 to a comma-delimited (.csv) file stored in the Logs\VT\CSV directory. Because no file rotation period is specified, each file contains data for one calendar day. The output files are .csv files named in the format %yyyy%mm%dd-%x-mask%.csv.

### Sample Log Files

You can configure transformation functionality to create and maintain an up-to-date, compact version of your full log files. Doing so enables you to test your dataset configurations quickly, with reprocessing times of seconds or minutes instead of hours needed to reprocess the entire dataset. The following example provides an example of how to configure transformation functionality to do this.

The following Transform.cfg window provides instructions to read .vsl files from the Logs directory and export the x-timestring and x-trackingid fields to a text file stored in the Logs\VT directory. The specified Hash Threshold filters certain tracking IDs from the dataset, thereby creating a dataset that is sampled by a factor of 100. Because no file rotation period is specified, each file contains data for one calendar day. The names of the output files are in the default format %yyyy%mm%dd-%x-mask%.txt.
Splitting Log Files by Web Site Section

The following **Transform.cfg** window provides instructions to read .vsl files from the Logs directory and export the x-timestring and x-trackingid fields to a text file stored in the Logs\VT directory. The regular expression transformation (**RETransform**) takes as its input the cs-uri-stem field and creates a new field (x-site) that defines a section of the site. The x-site field is included in the name of output text files, each of which contain data for one calendar day.

The **Transform Mode.cfg** File

The configuration file **Transform Mode.cfg** enables you to pause processing of data into a dataset, specify offline sources, or specify the frequency at which the data workbench server running transformation functionality saves its state files.
Making changes to the Transform Mode.cfg file, including adding or removing sources, does not cause reprocessing of the data.

**To edit the Transform Mode.cfg file for a dataset profile**

1. While working in the profile whose data you want to export, open the Profile Manager and click Dataset to show the contents of the directory.
2. Right-click the check mark next to Transform Mode.cfg and click Make Local. A check mark for this file appears in the User column.
3. Right-click the newly created check mark and click Open > from the workbench. The Transform Mode.cfg window appears.
4. Edit the parameters in the configuration file using the following table as a guide:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline Sources</td>
<td>Mask of the offline log source.</td>
</tr>
<tr>
<td></td>
<td>To specify an offline source:</td>
</tr>
<tr>
<td></td>
<td>• Right-click Offline Sources and click Add new &gt; Source.</td>
</tr>
<tr>
<td></td>
<td>• In the parameter for the new source, enter the mask of the log sequence. For Sensor log sources with file names of the format YYYYMMDD-SENSORID.vsl, the mask is SENSORID SENSORID is case-sensitive. For log file log sources, the mask is the string extracted by the Mask Pattern (see Log Files).</td>
</tr>
<tr>
<td></td>
<td>Adding or removing sources from Offline Sources does not cause reprocessing of the dataset.</td>
</tr>
<tr>
<td></td>
<td>As Of time measurements are maintained for the processing of the profile's online sources. When the offline source is again online, the processing of incoming log files for that source resumes.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Whenever a source comes back online, you should remove it from Offline Sources. If you do not do so, the data workbench server treats the source as an online source and updates the As Of time as long as the source is sending data. If the source goes offline again, the As Of time measurements stop.</td>
</tr>
<tr>
<td>Paused</td>
<td>True or false. If true, new data is not processed into the dataset. The default value is false.</td>
</tr>
<tr>
<td>Save Interval (sec)</td>
<td>Frequency at which the data workbench server on which transformation functionality is running saves its state files. The default value is 3600.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You should not change this value without consulting Adobe.</td>
</tr>
</tbody>
</table>

When editing the Transform Mode.cfg file within a data workbench window, you can use shortcut keys for basic editing features, including cut (Ctrl+x), copy (Ctrl+c), paste (Ctrl+v), undo (Ctrl+z), redo (Ctrl+Shift+z), select section (click+drag), and select all (Ctrl+a). In addition, you can use the shortcuts to copy and paste text from one configuration file (.cfg) to another.

5. Right-click (modified) at the top of the window and click Save.
6. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the data workbench Transform Mode.cfg in the User column, then click Save to > <profile name>, where profile name is the name of the profile for which you are exporting data. Reprocessing of the data begins after synchronization of the profile.

For information about reprocessing your data for export, see Reprocessing and Retransformation.

The Required Field Definitions.cfg File

The Required Field Definitions.cfg file is a Log Processing Dataset Include file that defines the required fields x-trackingid and x-timestamp.

The file is located in the Transform profile's Dataset\Log Processing folder. For more information about defining fields, see Log Processing Dataset Include Files.
Reprocessing and Retransformation

Information about dataset reprocessing and retransformation and how to prepare for either.

Understanding Reprocessing and Retransformation

During reprocessing, the data workbench server reconstructs your dataset as you have specified in the Log Processing and Transformation Dataset Configuration files.

To do so, the data workbench server (InsightServer64.exe) must complete both the log processing phase and the transformation phase of dataset construction. When log processing finishes, it triggers transformation to occur automatically, but transformation also can occur independently of log processing.

During the log processing phase, data workbench users do not have access to the data in the dataset. During the transformation phase, data workbench users do have access to up-to-date data, but the data is sampled instead of complete. Data analysis can continue during transformation, but queries will complete only as quickly as the transformation is occurring.

For more information about the log processing and transformation phases of dataset construction, see Dataset Configuration.

This section discusses the following topics:

• Reprocessing
• Retransformation

Reprocessing

Each time you complete one of the following tasks, log processing, and therefore transformation, occurs automatically to reconstruct your dataset as you have specified in the dataset configuration files:

• Add a new data source.
• Add a new data workbench server to your cluster in the Profile.cfg file.
• Change the Cluster.cfg file.
• Change the Log Processing.cfg file or a Log Processing Dataset Include file, including but not limited to the following:
  • Add a new parameter
  • Change a transformation
  • Change the Start Time or End Time parameters
• Upgrade your Insight Server.exe file.

You also can initiate reprocessing at any time by entering any character or combination of characters in the Reprocess parameter of the Log Processing.cfg file and saving the file.

💡 Note: For reprocessing to occur, the Pause parameter in the Log Processing Mode.cfg file must be set to false. This parameter's default value is false, so changing the parameter may not be required. For more information about Log Processing Mode.cfg, see Additional Configuration Files.
Retransformation

Each time you change any information in the Transformation.cfg file or in a Transformation Dataset Include file, such as change a transformation or define a new dimension, transformation occurs automatically.

Each time you change lookup files that are referenced in the Transformation.cfg file or in a Transformation Dataset Include file (including lookup files for Categorize, FlatFileLookup, and ODBCLookup transformations), you must initiate transformation by entering any character or combination of characters in the Reprocess parameter of the Transformation.cfg file and saving the file.

Preparing for Reprocessing or Retransformation

Steps to ensure that reprocessing or retransformation goes smoothly and finishes in time for data workbench users to get back to work

1. Determine the elapsed time of previous processing or transformation by checking the dataset profile’s Processing Mode History in the Detailed Status interface.
   1. While working in your dataset profile, open the Detailed Status interface.
   2. Click Processing Status > <dataset profile name> > Processing Mode History to view the elapsed times of previous processing or transformation.
      • Fast Input is the total time needed for log processing.
      • Fast Merge is the total time needed for transformation.
      • The sum of the two times (Fast Input + Fast Merge) is the total time needed for processing the dataset.

      The example below indicates that log processing took approximately 43 seconds, while transformation took less than 2 minutes.

<table>
<thead>
<tr>
<th>Processing Mode History</th>
<th>Real Time for 00:00:01 (hh:mm:ss)</th>
<th>Fast Input for 00:00:43 (hh:mm:ss)</th>
<th>Fast Merge for 00:01:40 (hh:mm:ss)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information about the Detailed Status interface, see the Data Workbench User Guide.

2. Schedule and plan the reprocessing. Because data workbench users do not have access to the data during the log processing phase, ensure that you schedule reprocessing to occur during an appropriate time, such as over the weekend.

3. Monitor the progress of the reprocessing and retransformation using the fields in the Processing Legend. For more information about the Processing Legend, see the Data Workbench User Guide.
Dataset Configuration Tools

Information about the data workbench interfaces that enable you to monitor or manage your dataset and the options for hiding dataset components.

Working With Dataset Configuration Interfaces

The dataset configuration interfaces show the relationships among the components (log sources, fields, transformations, and dimensions) of your dataset.

Some of the interfaces provide options for viewing information about, searching within, or editing dataset components.

To open the dataset configuration interfaces

You can open the interfaces for dataset configuration using one of the following methods:

- **Admin Menu**: Right-click within a workspace, click **Admin**, then click the appropriate menu option.
  - OR -
- **Admin Tab**: Click to open the appropriate workspace thumbnail on the **Admin > Dataset and Profile tab**.

Dataset Schema

The **Dataset Schema** interface displays the extended dimensions (countable, simple, many-to-many, numeric, denormal, and time dimensions) defined in any **Transformation Dataset Configuration** file and the relationships between those dimensions.

In addition, the **Dataset Schema** interface shows any derived dimensions that you have defined, as well as any extended dimensions that are configured to be hidden.

This section discusses the following topics:

- To Interpret Dimension Type Using the Dataset Schema Interface
- To Display the Default Visualization for a Dimension
- To Display a Specific Visualization for a Dimension
To Interpret Dimension Type Using the Dataset Schema Interface

The following table lists the dimension types and the colors in which their names appear in the Dataset Schema interface. Parents for the sample dimensions (from the example above) are noted as well.

**Dataset Schema Interface**

<table>
<thead>
<tr>
<th>Dimension Type</th>
<th>Color</th>
<th>Sample Dimension and Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Countable</td>
<td>Pink</td>
<td>Visitor - In this schema, Visitor is a root countable dimension.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Session - parent is Visitor.</td>
</tr>
<tr>
<td>Denormal</td>
<td>Yellow</td>
<td>DenormalPage - parent is Page View.</td>
</tr>
<tr>
<td>Derived</td>
<td>Blue</td>
<td>Next Page - parent is Page View.</td>
</tr>
<tr>
<td>Many-to-Many</td>
<td>Pink and Green</td>
<td>Search Term - parent is Session.</td>
</tr>
<tr>
<td></td>
<td>(The stem from the parent is pink, while the dimension name is green.)</td>
<td></td>
</tr>
<tr>
<td>Numeric</td>
<td>Green</td>
<td>Exact Page Duration - parent is Page View. In this example, Exact Page Duration is a hidden numeric dimension. See the Hidden dimension type in this table.</td>
</tr>
<tr>
<td>Simple</td>
<td>Green</td>
<td>Page - parent is Page View.</td>
</tr>
<tr>
<td>Time</td>
<td>Green</td>
<td>Hour - parent is Session.</td>
</tr>
<tr>
<td>Hidden</td>
<td>Hidden dimensions are a darker version of the appropriate dimension type color. For example, a hidden numeric dimension is a darker, less bright green.</td>
<td>Exact Page Duration - parent is Page View.</td>
</tr>
</tbody>
</table>

To Display the Default Visualization for a Dimension

- In the Dataset Schema interface, click the desired dimension. The default visualization displays. For example, if the default visualization is a table displaying Sessions and the selected dimension, and you click the URI dimension, data workbench displays a table with URI by Sessions.

  *Note: If you want to change the default visualization that displays, see the Configuring Interface and Analysis Features chapter in the Data Workbench User Guide.*

To Display a Specific Visualization for a Dimension

- In the Dataset Schema interface, right-click the desired dimension and click Add Visualization > `<visualization type>`.

Dependency Maps

Dependency maps enable you to visualize and manage the configuration of the components of your profile.
- **Dataset components**: Log sources, filters, fields, transformations, and extended dimensions defined in your dataset's Log Processing.cfg, Transformation.cfg, and dataset include files.
- **Query model components**: Metrics, dimensions, and filters defined in the Dimensions, Metrics, and Filters folders.
- **Workspaces and visualizations**: Workspaces, reports, menu options, and globe layers.

For more information about working with query model components, workspaces, and visualizations in dependency maps, see the *Data Workbench User Guide*.

Profile components are represented by colored dots (nodes) in the map. The lines connecting the nodes depict dependencies, that is, how the components relate to one another. A line between two nodes means that an output of the node on the left is an input of the node on the right, that is, the right node depends on the left node.

**Displaying Dataset Components**

**To display the profile's dataset components**

1. Right-click within the dependency map and click **Display**.
2. Choose **Dataset**. An X appears to the left of **Dataset**.

For more information about the other display options see the *Data Workbench User Guide*.

The following figure shows a dependency map whose nodes represent a dataset's log sources, fields, transformations, and extended dimensions.

- A yellow-green node represents one or more log sources or a filter defined in the dataset. A node for a log source always appears furthest to the left in the map.
- A gray node represents a field that is listed in the Fields parameter in a Log Processing.cfg or Log Processing Include file.
- A blue node represents a transformation.
- A green node represents an extended dimension.

💡 **Note**: If your dataset has a single log source, the map displays Log Source: log source name. If your dataset has multiple log sources, the map displays number Log Sources, where number is the count of log sources. For example, if you have three log sources in your dataset, your map displays 3 Log Sources.

If you cannot see all of the nodes on the map, you can move the map or zoom in or zoom out to display the entire map or to focus on a particular section. For more information about zooming, see the Working with Visualizations chapter of the *Data Workbench User Guide*. 
When you click a node, all of the nodes that depend on that node and all of the nodes on which that node depends are highlighted and their names display.

💡 **Note:** *A highlighted path in a dependency map does not constitute a selection.*

When you right-click a node, you can see identifying information about each component shown on the map and choose menu options that enable you to view more detail about the component or to edit the component. In addition, you can perform text searches and display performance information for transformations and extended dimensions.

For information about these functions for dependency maps, see the Administrative Interfaces chapter of the *Data Workbench User Guide*.

### Field Viewers

A field viewer is a table containing the values of one or more data fields.

The fields whose values display are inputs or outputs of a dataset's log sources, transformations, or extended dimensions. The field's name is shown in the column heading, and each row contains the field's value for a single row of source data. Because field viewers enable you to see a field's values, they are helpful in writing and testing regular expressions.

You can open a field viewer as a callout from a *Transformation Dependency* map or as a standalone visualization from the **Admin** menu:

- Creating a field viewer from a *Transformation Dependency* map. When you open a field viewer from a *Transformation Dependency* map, the viewer is populated automatically based on the log source, transformation, or dimension that you right-click. For a log source or a transformation, the fields in the viewer are inputs or outputs of the log source or transformation. For a dimension, the fields are inputs of the dimension. You can add and remove fields as desired.

- Creating a field viewer as standalone visualization. When you open a field viewer as a standalone visualization, you can create a *Log Processing Field Viewer* or a *Transformation Field Viewer*. The viewer is blank, and you must add the desired fields to the viewer. For a *Log Processing Field Viewer*, you can add fields from the *Log Processing.cfg* file or any *Log Processing Dataset Include* file. For a *Transformation Field Viewer*, you can add fields from the *Transformation.cfg* file or any *Transformation Dataset Include* file.

For information about adding and removing fields and filtering within a field viewer, see the Administrative Interfaces chapter of the *Data Workbench User Guide*. 

Hiding Dataset Components

As you configure your dataset, you might want to hide certain extended dimensions so they do not show on the dimension menu in data workbench.

You might want to override certain dataset configuration files in a profile that you cannot or do not want to modify. The following sections provide instructions to perform these tasks.

Hiding Extended Dimensions

You can use either the Hidden parameter or the Show parameter to hide extended dimensions so they do not show on the dimension menu in data workbench.

When you enter the appropriate setting for either parameter, the dimension name is not listed in the menu in data workbench, but it is still in the profile and available to be used. Any data workbench user can temporarily unhide hidden dimensions by setting the Unhide All parameter in the Insight.cfg file to true.

For more information about the Unhide All parameter, see the appendix on data workbench configuration parameters in the Data Workbench User Guide.

The following sections describe how to use the Hidden and Show parameters to hide extended dimensions.

• Hiding Extended Dimensions Using the Hidden Parameter
• Hiding Extended Dimensions Using the Show Parameter

Hiding Extended Dimensions Using the Hidden Parameter

The Hidden parameter is an optional parameter that you can use when defining extended dimensions in Transformation Dataset Configuration files.

1. Open Transformation Dataset Configuration files in which the extended dimension that you want to hide is defined. See Editing Existing Dataset Include Files.
2. Locate the Hidden parameter for the desired dimension in the configuration window and type true.
3. Save the file locally, then save it to the appropriate profile on the server. See Editing Existing Dataset Include Files.

The dataset retransforms, after which the extended dimension does not appear on the dimension menu in data workbench. For more information about the Hidden parameter, see Extended Dimensions.

If you change the Hidden parameter’s setting, you must retransform the dataset for the change to take effect.

Hiding Extended Dimensions Using the Show Parameter

The Show parameter is not one of the parameters available for defining extended dimensions in Transformation Dataset Configuration files. Instead, the parameter is defined in the .dim files for any derived dimensions that you create. Therefore, to use the Show parameter to hide an extended dimension, you first must create a derived dimension that is based on the extended dimension as described in the following procedure:

1. Use a text editor such as Notepad to create an empty file called <dimension name>.dim The file name should match the name of the dimension that you want to hide. For example, to hide the Next Page dimension, you would create a Next Pagedim file.
2. Add the following text to the file:
• entity = ref: wdata/model/dim/Parent/+name
• show = bool: false

3. Save the file to the profile’s Dimensions directory. You can save the file to a subdirectory if desired.

When you use the Show parameter to hide an extended dimension, you do not have to retransform your dataset to make the change take effect. You can choose to hide or show the dimension in your local version of the profile (that is, you can save the .dim file to your User folder), or you can save the .dim file to the server for use by other users of the profile.

You also can use the Show parameter to hide metrics and filters. For information, see the Configuring Interface and Analysis Features chapter in the Data Workbench User Guide.

### Hiding Dataset Configuration Files

If you do not want to inherit a configuration file from an internal or other inherited profile (that is, you want the instructions in the file to be ignored during dataset construction), but you do not want to modify the file, you can create an empty (zero-byte) file with the same name and store the file in another profile.

**To zero-byte a dataset configuration file**

1. In the Profile Manager, open the necessary folders and subfolders to locate the file that you want to zero-byte.
2. Right-click the check mark next to the name of the file and click Make Local.
3. Open the local file in a text editor such as Notepad and delete its contents.
4. Save and close the file.
5. In the Profile Manager, save the zero-byte file to a profile to the right of the profile in which the original file resides. (You want the zero-byte file to take precedence over the original file.)

In the Profile Manager, a hyphen (-), instead of a check mark, in a column identifies the zero-byte file as shown in the example below.

<table>
<thead>
<tr>
<th>File</th>
<th>Base</th>
<th>Movie Data</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile.cfg</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Context\</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset\</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cluster.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Log Processing.cfg</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Log Processing Mode.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Server.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Transformation.cfg</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Log Processing\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TimeZone\</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Transformation\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Movie Ratings.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Movie Transformation.cfg</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Dimension Legend\</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Dimensions\</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>FileEditor\</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Filters\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Images\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Menu\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Metrics\</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
When you reprocess your dataset, the dataset does not contain the dataset components that the original file defines.

💡 **Note:** If you zero-byte a configuration file that defines an extended dimension that is used in a visualization or a metric definition, data workbench produces an error for that visualization or metric, respectively.

You can also use zero-byte files to move a metric, dimension, or filter to another location in the profile or to hide menu items. For information, see the *Data Workbench User Guide*. 
Conditions

Information about the types of condition operations that you can use when configuring your dataset.

About Conditions

Transformations and dimensions use conditions to determine when certain instructions or actions apply to log fields. The Log Entry Condition parameter uses conditions to determine which log entries should be included in the dataset construction process. This appendix describes the different types of conditions available within data workbench server.

A condition falls into one of two categories:

- **Test Operations:** Compare, Not Empty, Range, Regular Expression, and String Match conditions are used to test for different states in the available log fields.
- **Boolean Operations:** The And, Or, and Neither conditions are used to combine the test operations described above. For example, if you have a Range condition and a String Match condition that must both be false to take the appropriate action, you would make these two operations children of the Neither condition. Note that the And condition is used as the root of all condition testing in the system.

Working With Conditions

Information about adding, removing, or copying a condition.

- **To Add a Condition to a Dataset Configuration File**
- **To Remove a Condition from a Dataset Configuration File**
- **To Copy a Condition**

**To Add a Condition to a Dataset Configuration File**

1. While working in your dataset profile, use the Profile Manager to open the dataset configuration file that you want to edit.
   - To open the Log Processing.cfg file, see *Editing the Log Processing Configuration File*.
   - To open the Transformation.cfg file, see *Editing the Transformation Configuration File*.
   - To open a Dataset Include file, see *Dataset Include Files*.

2. Within the dataset configuration file, locate the Log Entry Condition or Condition parameter that you would like to define.

3. Right-click the parameter and click Add new. Choose one of the following condition types:
   - Not Empty
   - String Match
   - Regular Expression
   - Range
   - And
   - Or
   - Neither
   - Compare
4. Edit the parameters of the condition as desired. For descriptions of the parameters of each condition, see the appropriate section of this appendix.

5. To save your changes, right-click (modified) at the top of the window and click Save.

6. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the file belongs.

   **Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

---

**To Remove a Condition from a Dataset Configuration File**

1. Right-click the name of the condition or the number corresponding to the condition that you want to remove.
2. Click Remove <#number>, where number is the number corresponding to the condition that you want to remove.

---

**To Copy a Condition**

You can copy a condition from one location to another location in the same file, or you can copy a condition from one dataset configuration file to another.

1. Right-click the name of the condition or the number corresponding to the condition that you want to copy and click Copy.
2. Right-click the name or number of the condition below which you want to place the copied condition and click Paste.

---

**Test Operations**

Test operations take input from the available log fields, perform a test based upon their values, and return true if the test succeeded or false if the test failed.

These tests are always the child of one of the Boolean operations described in the next section. See Boolean Operations.

   **Note:** If the input is a vector of strings, only the first value (string) in the vector is used for the test. This rule applies to all of the test operations.

---

**Test Operation Conditions**

Information about test operation conditions including compare, not empty, range, regular expression, and string match.

- **Compare**
- **Not Empty**
- **Range**
- **Regular Expression**
- **String Match**
**Compare**

The **Compare** condition compares string or numeric values. For comparisons of string values, you can specify whether case should be considered.

The parameters of the **Compare** condition are described in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Sensitive</td>
<td>True or false. Used only if the Type is <strong>LEXICAL</strong>. If set to false, upper and lower case letters are considered equal.</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the condition.</td>
<td>Comments</td>
</tr>
<tr>
<td>Input A</td>
<td>The first of the two values to compare. This value represents the left operand in the condition.</td>
<td></td>
</tr>
<tr>
<td>Input B</td>
<td>The second of the two values to compare. This value represents the right operand in the condition.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>The comparison operation. The available operations (and their meanings) are as follows:</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>• = or == (Input A equals Input B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &lt;&gt; or != (Input A is not equal to Input B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &lt; (Input A is less than Input B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &lt;= (Input A is less than or equal to Input B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &gt; (Input A is greater than Input B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• &gt;= (Input A is greater than or equal to Input B)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>The type of comparison to be made. Available types are <strong>LEXICAL</strong>, <strong>NUMERIC</strong>, and <strong>DATETIME</strong>. For descriptions of the types, see <em>Test Types for Test Operations</em>.</td>
<td><strong>LEXICAL</strong></td>
</tr>
</tbody>
</table>

This example uses a **Compare** condition to define the **Log Entry Condition**. As the data workbench server reads each event data record, it compares the numeric values x-age and 55. If for a given log entry, x-age is less than or equal to 55, the log entry is included in the dataset construction process.

<table>
<thead>
<tr>
<th>Log Entry Condition</th>
<th>AndCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CompareCondition</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input A</td>
<td>x-age</td>
</tr>
<tr>
<td>Input B</td>
<td>&quot;55&quot;</td>
</tr>
<tr>
<td>Operation</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Type</td>
<td>NUMERIC</td>
</tr>
</tbody>
</table>

**Not Empty**

The **Not Empty** condition checks a field to see if it contains a value or is empty. The condition is satisfied for any log entry whose value for the **Input** field is not empty.

The parameters of the **Not Empty** condition are described in the following table:
This example takes as its input x-some-field and tests whether the field is not empty. The condition is satisfied if the field is populated.

```
-1
<table>
<thead>
<tr>
<th>Comments</th>
<th>NotEmptyCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>x-some-field</td>
</tr>
</tbody>
</table>
```

### Range

The **Range** condition takes an input field and determines whether the value of that field falls, inclusively, within the given minimum (Min) and maximum (Max) parameter values.

The parameters of the **Range** condition are described in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Sensitive</td>
<td>True or false. Used only if the <strong>Type</strong> is <strong>LEXICAL</strong>. If set to false, upper and lower case letters are considered equal.</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the condition.</td>
<td>Comments</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the log entry to use as input.</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>Lower bound of range.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This parameter's value must be a literal value or a string - not a field name. If you use a date for this field, you must specify a time zone. For a list of supported time zone abbreviations, see <a href="#">Time Zone Codes</a>.</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>Upper bound of range.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This parameter's value must be a literal value or a string - not a field name. If you use a date for this field, you must specify a time zone. For a list of supported time zone abbreviations, see <a href="#">Time Zone Codes</a>.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>The type of comparison to be made. Available types are LEXICAL, NUMERIC, and DATETIME. For descriptions of the types, see <a href="#">Test Types for Test Operations</a>.</td>
<td></td>
</tr>
</tbody>
</table>

This example uses a **Range** condition to define the **Log Entry Condition**. As the data workbench server reads each event data record, it compares the numeric values x-age and 55. If for a given log entry, x-age is at least 55, the log entry is included in the dataset construction process. This example performs the same function as the **Compare** condition example. See **Compare**.

**Note:** If the Min or Max parameter is left blank, the data workbench server substitutes the minimum or maximum integer values available. The minimum value is zero (0), and the maximum value is infinity.
The Regular Expression condition test uses regular expressions pattern matching (see Regular Expressions) to determine whether the value of the specified input field contains a string that matches one of the patterns specified in the Matches parameter.

If the input is a vector of strings, only the first value in the vector is used for the test. The Regular Expression condition performs full string comparisons. If you want to identify substrings, you must prepend and append ".*" to the string.

The parameters of the Regular Expression condition are described in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Sensitive</td>
<td>True or false. If set to false, upper and lower case letters are considered equal.</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the condition.</td>
<td>Comments</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the log entry to use as input.</td>
<td></td>
</tr>
<tr>
<td>Matches</td>
<td>The regular expression pattern(s) to match against the value of the input field.</td>
<td></td>
</tr>
</tbody>
</table>

**To add a regular expression pattern**

1. Right-click **Matches**.
2. Click **Add new > Regular Expression**.
3. Enter the desired regular expression in the text box.

This example illustrates the use of the Regular Expression condition to match a field of data collected from website traffic. The condition returns true only if the cs(referrer-query) field contains a string matching the regular expression campaign=C\[1-9\][0-9]{4}. This regular expression matches any string containing "campaign=C12345." However, the pattern would not match the string "campaign=C0123&" because the first character after the "C" is not in the range 1-9.
String Match

The String Match condition tests for string equality. It takes a specified field as input and tests the value of that field in each log entry against the strings specified in the operation's Matches parameter. If any one of these case-sensitive match strings is the same as the value in the provided input field, the operation returns true. In the event that the StringCondition contains no match strings, the condition returns false. If the input is a vector of strings, only the first value (string) in the vector is used for the test.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Sensitive</td>
<td>True or false. If set to false, upper and lower case letters are considered equal.</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Optional. Notes about the condition.</td>
<td>Comments</td>
</tr>
<tr>
<td>Input</td>
<td>The name of the field from the log entry to use as input.</td>
<td></td>
</tr>
<tr>
<td>Matches</td>
<td>The string(s) to match against the value of the input field.</td>
<td></td>
</tr>
</tbody>
</table>

**To add a string**
1. Right-click Matches.
2. Click Add new > String.
3. Enter the desired string in the text box.

This example uses data collected from website traffic to illustrate the use of the String Match condition. The condition tests whether the input field (cs-uri-stem) matches either of the two strings specified in the Matches parameter, and it succeeds if the field cs-uri-stem is either the exact string /navigation/footer.asp or the exact string /navigation/header.asp.

<table>
<thead>
<tr>
<th>-0</th>
<th>StringCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Matches</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>/navigation/footer.asp</td>
</tr>
<tr>
<td>1</td>
<td>/navigation/header.asp</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input</td>
<td>cs-uri-stem</td>
</tr>
</tbody>
</table>

**Test Types for Test Operations**

The Compare condition and Range condition require that you specify the type of comparison to be made for the condition.

The following table describes the available types (LEXICAL, NUMERIC, and DATETIME).

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>First turns the input field into an integer. If this is not possible, a value of zero is used. The test returns true only if the resulting integer input value is greater</td>
<td>If either of the min or max fields is left blank, the system uses the appropriate min or max value available to 64-bit signed integers.</td>
</tr>
<tr>
<td>Test Type</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>DATETIME</td>
<td>First turns the input field into a date. If the input field cannot be turned into a valid date, the condition test returns false. If the field can be turned into a date, the test returns true only if the input date falls on or after the specified minimum date and on or before the specified maximum date.</td>
<td>If the min and max dates are not valid, the dataset are not constructed. If the min or max dates are not supplied, the system substitutes appropriately either the min date (Jan 1, 1600) or the max date (sometime in the 24th century). Adobe recommends using one of the following formats for DATETIME: • January 1 2013 HH:MM:SS EDT • Jan 1 2013 HH:MM:SS GMT The time zone defaults to GMT if not specified.</td>
</tr>
<tr>
<td>LEXICAL</td>
<td>Returns true only if the input field is lexically greater than or equal to the string specified as the minimum and less than or equal to the string specified in the maximum value.</td>
<td>Lexical comparison uses the ASCII value of characters in the strings moving from left to right comparing the characters. For the first character that does not match, the one with the larger ASCII value is considered to be the greater of the two. In the event that one string is shorter than the other, but up until that point all of the characters have been the same, the longer string is considered the greater of the two. If the strings are character for character equivalent and the exact same length, they are considered lexically equivalent.</td>
</tr>
</tbody>
</table>

### Boolean Operations

The **Boolean** operations combine the results of the test operations, which function as children of the **boolean** operations.

For information about the test operations, see **Test Operations**. When you define a **boolean** operation, you can define zero or more children for the operation.

**To add a child condition to a Boolean operation**

1. Right-click the name or the number corresponding to the **Boolean** operation.
2. Click **Add new child** and choose one of the available condition types to add.
3. Repeat steps 1 and 2 until you have added all of the desired child conditions for the **Boolean** operation.

**Note:** When you right-click the name or the number corresponding to a **Boolean** operation, you see the **Add new sibling** menu option. A sibling is another condition at the same relative position in the condition hierarchy as the **Boolean** operation that you right-clicked. Adding a new sibling for a **Boolean** operation is the same as adding a new condition by right-clicking the **Condition** or **Log Entry Condition** parameter.
To remove a child condition from a Boolean operation:

1. Right-click the name of the child condition or the number corresponding to the child condition that you want to remove from the Boolean operation.
2. Click Remove <#number>, where number is the number corresponding to the child condition that you want to remove.

This section discusses the following topics:

- **And**
- **Neither**
- **Or**

**And**

The And condition can have zero or more child conditions and returns true when none of its children nodes return false.

The And condition forms the root operation of all condition testing within the data workbench server. If the And condition contains no children, the condition evaluates to true and the associated operation proceeds. This is why actions that have only the And condition as the condition test always execute and why it is used as the root for all condition tests.

This example shows how an And condition is used to make sure that the Copy transformation occurs when only both the date of the log entry occurred in the year 2006 and that the page requested was /products/purchase.asp.

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Sample Copy</td>
</tr>
<tr>
<td>Condition</td>
<td>AndCondition</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input</td>
<td>x-timestamp</td>
</tr>
<tr>
<td>Max</td>
<td>December 31, 2006</td>
</tr>
<tr>
<td>Min</td>
<td>January 1, 2006</td>
</tr>
<tr>
<td>Type</td>
<td>DATETIME</td>
</tr>
<tr>
<td>-1</td>
<td>StringCondition</td>
</tr>
<tr>
<td>Matches</td>
<td>/products/purchase.asp</td>
</tr>
</tbody>
</table>

**Neither**

The Neither condition can have zero or more child conditions and returns false if any of its child conditions evaluate to true. If the Neither condition contains no children, none of its children can return true. As a result, the Neither condition evaluates to true.
The following example shows a Neith er condition with two Range conditions as its children. As defined, the Neith er condition excludes log entries that occurred between January 1, 2007 and January 10, 2007 or during the period January 12, 2007, to January 14, 2007. Such a condition might be used as the Log Entry Condition to eliminate transactions from a dataset during periods in which there was a known problem with the data collected.

<table>
<thead>
<tr>
<th>Condition</th>
<th>AndCondition</th>
<th>NeitherCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>RangeCondition</td>
</tr>
<tr>
<td>-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>x-timestamp</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>January 10, 2007</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>January 1, 2007</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>DATETIME</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
<th>OrCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>StringCondition</td>
</tr>
<tr>
<td>-0</td>
<td></td>
</tr>
<tr>
<td>Matches</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input</td>
<td>x-hasproblem</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>1</td>
<td>RangeCondition</td>
</tr>
<tr>
<td>Case Sensitive</td>
<td>true</td>
</tr>
<tr>
<td>Comments</td>
<td>Comment</td>
</tr>
<tr>
<td>Input</td>
<td>x-timestamp</td>
</tr>
<tr>
<td>Max</td>
<td>January 10, 2007</td>
</tr>
<tr>
<td>Min</td>
<td>January 1, 2007</td>
</tr>
<tr>
<td>Type</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>
Regular Expressions

Regular expressions are used across all data workbench search fields including the query entity panels.

- About Regular Expressions
- Regular Expression Terminology
- About Literal Matching
- Using Metacharacters
- Pattern Extraction

About Regular Expressions

A regular expression is a text pattern, consisting of a combination of alphanumeric characters and special characters known as metacharacters, that locates patterns and extract substrings from text. Regular expressions are widely used in computer programming and are an integral part of languages such as Perl.

To identify and extract complex string patterns, the data workbench server uses regular expressions in some of the transformations and conditions. What follows is a brief guide to regular expressions.

This appendix is not a comprehensive introduction to regular expressions. A particularly good reference is the O'Reilly publication *Mastering Regular Expressions, 2nd Edition* by Jeffrey E. F. Friedl.

Regular Expression Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal</td>
<td>A literal is a character we use in a regular expression to locate a specific sequence of characters. For example, to find product in shop/products.html, the string product is a literal, or what we are literally looking for in the string.</td>
</tr>
<tr>
<td>Metacharacter</td>
<td>A metacharacter is a special character that has a unique interpretation in the context of regular expressions. For example, the period (.) is a metacharacter that is used to match any character.</td>
</tr>
<tr>
<td>Escape Sequence</td>
<td>An escape sequence is simply a way to tell the regular expression engine that we would like to use one of the metacharacters as a literal. Escape sequences always start with the backslash character (). By placing the backslash (which is also a metacharacter) in front of a metacharacter, the regular expression engine interprets the escaped metacharacter as a literal. For example, if you want to match the metacharacter period (.), you need to use an escape sequence. However, to match one of the periods in the string 168.196.0.11, you could use the regular expression consisting of a backslash and a period (.).</td>
</tr>
<tr>
<td>Pattern</td>
<td>This is shorthand terminology for the regular expression. In essence, a regular expression is a pattern you are trying to match against the target string.</td>
</tr>
<tr>
<td>Target String</td>
<td>This term refers to the string we are searching within to locate the desired pattern.</td>
</tr>
</tbody>
</table>

About Literal Matching

Literal matching takes a literal string without any escape characters and looks in the target string to see if it is a substring of the target string.

In this example, you see how literal matching works. Consider a situation in which data is collected from website traffic, and the cs(referrer) field contains the following value:
To determine whether the referrer represents someone who clicked on one of the advertisements, you need to see whether the referrer contains the string ad. You could simply use the literal string ad to search the target string and determine if an advertisement was used to route the traffic to the site. While this would match the target string, it would match in two locations and is thus ambiguous and can lead to false positives.

The following URL contains the string ad in two different places:

http://www.abc.com/adventurenews/today.html?ad=123AZ45

Thus, if you are trying to determine which sessions started as a result of a particular advertisement campaign, simply using the literal ad as the regular expression is clearly not sufficient. Changing the literal to "ad=" would eliminate this ambiguity and result in the expression making only a single match. However, even this may not be sufficient to ensure that the referrer was part of the advertisement campaign. Consider the following referrer:

http://www.xyz.com/hello.html?pad=something

You do not have any control over the URLs that others may be using to create links to the site. Literal matching is too simple a mechanism to locate sessions that started as a result of the advertisement campaign. The following section discusses how you can use metacharacters for more flexible and powerful matching.

Using Metacharacters

A metacharacter is a special character in a program or data field that provides information about other characters.

<table>
<thead>
<tr>
<th>metacharacter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>. (dot)</td>
<td>Matches a single character, for example: re:x.z matches &quot;xyz&quot; or &quot;xxz&quot;.</td>
</tr>
<tr>
<td>* (star)</td>
<td>Matches one or more characters, for example: re:Z* matches &quot;ZZZ&quot;.</td>
</tr>
<tr>
<td>? (wildcard)</td>
<td>Matches 0 or 1 of previous expression to force minimal matching, for example: xy?z matches &quot;xy&quot; and &quot;xyz&quot;.</td>
</tr>
</tbody>
</table>

Additional common regular expressions can also be used to create more complex search strings.

Lists, Ranges, and OR

Literal matching lets you look for a single string, but brackets, dashes, and pipes let you define a list of things to look for in the target string.

<table>
<thead>
<tr>
<th>For this metacharacter...</th>
<th>The regular expression processor will...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square Brackets ([ ])</td>
<td>Match any of the characters inside of the bracket with a single character position. For example, [AB] is an instruction to match either the letter A or the letter B and [0123456789] says match to any character in the range 0 to 9.</td>
</tr>
<tr>
<td>Dash (-)</td>
<td>Match a range of characters. Thus, instead of writing [0123456789] we could simply write [0-9]. This can be extended to ranges of characters and multiple ranges within one set of brackets. For example, [0-9A-C] matches the characters 0 through 9 and A to C.</td>
</tr>
</tbody>
</table>
The regular expression processor will...  

<table>
<thead>
<tr>
<th>For this metacharacter...</th>
<th>The regular expression processor will...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe (</td>
<td>)</td>
</tr>
</tbody>
</table>

Note: To test for a dash (-) as a literal inside the brackets, it must come first or last. For example, [-0-9] tests for - and 0 to 9.

Consider the following examples:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>String</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win95[5]</td>
<td>OS=Win95</td>
<td>Win95</td>
</tr>
<tr>
<td>Win95</td>
<td>8</td>
<td>OS=Win98</td>
</tr>
<tr>
<td>[0-9]</td>
<td>Mozilla/3.0</td>
<td>3</td>
</tr>
<tr>
<td>Lesson[A-Z]</td>
<td>Lesson a</td>
<td>No match because lower-cased a is not in the range of upper-cased A through Z.</td>
</tr>
</tbody>
</table>

Negation

Negation is a way to say that you would like to match anything except the given characters. The negation metacharacter, the circumflex or caret (^), is used as the first character inside brackets to say that you would like the match to be anything but the remaining characters in the brackets. For example, to match any character but a semicolon (;), you would write

[^;]

This would match any character except the semicolon.

Positioning

To force a match to the beginning or end of a target string, one of two metacharacters are used.

<table>
<thead>
<tr>
<th>For this metacharacter...</th>
<th>The regular expression processor will...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circumflex or Caret (^)</td>
<td>Match against the beginning of the string. For example, ^[Tt]he would match the target string &quot;The Beginning&quot; but would not match &quot;This is the beginning.&quot;</td>
</tr>
<tr>
<td>Dollar sign ($)</td>
<td>Match against the end of the string. For example, [Ee]nd$ would match &quot;This is the end&quot; but would not match &quot;The end is a special time.&quot;</td>
</tr>
</tbody>
</table>

Note: When the regular expression contains ^ at the beginning and $ at the end, the entire target string must match the regular expression.

Matching Anything

The period (.) is a special metacharacter that matches any character in the target string. For example, the regular expression ^...$ matches any target string that is exactly three characters long. The regular expression "..." matches any target string that contains at least three characters.

Repeated Patterns
Iteration metacharacters let you match a pattern more than once.

<table>
<thead>
<tr>
<th>For this metacharacter...</th>
<th>The regular expression processor will...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question Mark (?)</td>
<td>Match no instances or one instance of the character immediately preceding the metacharacter (?). For example, the pattern rea?d matches red and read.</td>
</tr>
<tr>
<td>Asterisk (*)</td>
<td>Match zero or more occurrences of the character immediately preceding the metacharacter (<em>). For example, the pattern [0-9]</em> matches any number of the characters 0 through 9 (any integer).</td>
</tr>
<tr>
<td>Plus (+)</td>
<td>Match one or more occurrences of the preceding character or range. For example, the pattern thre+ would match three but not through.</td>
</tr>
<tr>
<td>{n}</td>
<td>Match the proceeding character or range exactly n times. The following pattern matches United States phone numbers: [0-9]{3}-[0-9]{3}-[0-9]{4}. While not an optimal pattern, it determines whether the target string is in the proper format.</td>
</tr>
<tr>
<td>{n,m}</td>
<td>Match the preceding character at least n times and at most m times. For example, fo{1,2}d would match fod and food but not foodo.</td>
</tr>
</tbody>
</table>

**Pattern Extraction**

Pattern matching is only part of the power of regular expressions. Regular expressions also provide a mechanism for extracting key portions of a target string. This is done through the use of the left and right parentheses. These extractions are typically used as input into another process and are accessed through the use of %position%, where position is an integer referring to the count of which set of parentheses was matched.

Consider the following examples of pattern extraction:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>String</th>
<th>Match</th>
<th>Extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win(9[58])</td>
<td>OS=Win95</td>
<td>Win95</td>
<td>%1% = 95</td>
</tr>
<tr>
<td>(Win)(95</td>
<td>8)</td>
<td>OS=Win98</td>
<td>Win98</td>
</tr>
<tr>
<td>Mozilla/([0-9]).([0-9])</td>
<td>Mozilla/3.0</td>
<td>Mozilla/3.03</td>
<td>%1% = 3</td>
</tr>
<tr>
<td>Lesson([A-Z])</td>
<td>Lesson a</td>
<td>No match because lower-cased a is not in the range of upper-cased A through Z</td>
<td>%2% = 0</td>
</tr>
</tbody>
</table>
Event Data Record Fields

Information about the fields of data that the data workbench server can process to construct a dataset.

- About Event Data
- Baseline Event Data Record Fields
- Derived Fields

About Event Data

The event data used to build a dataset resides in files referred to as log sources. The data available in the log sources is called event data because each data record represents a transaction record or a single instance of an event with an associated timestamp.

A log source's event data is collected in real-time by Sensors. Event data collected by Sensors from HTTP and application servers is transmitted to data workbench servers, which convert the data into compressed log (.vsl) files. Event data that resides in a flat file, XML file, or an ODBC data source is read by the data workbench server, which provides decoders that you define to extract a common set of data fields from these different formats.

The following sections provide information about the data fields (referred to as event data record fields or log entry fields) that are collected by Sensors or read and made available to the data workbench server.

💡 Note: The names of the fields generally follow the naming convention for the W3C extended log file format. Many of the fields have prefixes that indicate the source of the information contained in the field:

- cs indicates communication from the client to the server.
- sc indicates communication from the server to the client.
- s indicates information from the server.
- c indicates information from the client.
- x indicates information that is created by an Adobe software product.

Baseline Event Data Record Fields

Log (.vsl) files contain the fields of event data that are collected from servers by Sensors and used by the data workbench server in the dataset construction process. The following table lists the fields in a typical event data record as recorded by Sensor:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ip</td>
<td>The IP address of the client as included in the request made to the server.</td>
</tr>
<tr>
<td></td>
<td>Example: 207.68.146.68</td>
</tr>
<tr>
<td>cs(cookie)</td>
<td>The cookies sent by the client with the request.</td>
</tr>
<tr>
<td></td>
<td>Example: v1st=42DF66DE610CF36;</td>
</tr>
<tr>
<td></td>
<td>ASPSESSIONID=4C1=610CF36;</td>
</tr>
<tr>
<td></td>
<td>ASPSESSIONID=4C1=610CF36;</td>
</tr>
<tr>
<td></td>
<td>ASPSESSIONID=4C1=610CF36;</td>
</tr>
<tr>
<td></td>
<td>Example: <a href="http://www.mysite.net/cgi-bin/websearch?qry">http://www.mysite.net/cgi-bin/websearch?qry</a></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cs(user-agent)</td>
<td>The string sent by the client with its request to the server that indicates what type of user agent the client is.</td>
</tr>
<tr>
<td></td>
<td>Example: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040707 Firefox/0.9.2</td>
</tr>
<tr>
<td>cs-method</td>
<td>The method type of the HTTP request.</td>
</tr>
<tr>
<td></td>
<td>Example: GET</td>
</tr>
<tr>
<td></td>
<td>Reference:</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.w3.org/TR/2000/NOTE-shoplogfileformat-20001115/#field_method">http://www.w3.org/TR/2000/NOTE-shoplogfileformat-20001115/#field_method</a></td>
</tr>
</tbody>
</table>
| cs-uri-query | The query string portion of URI (stem + query string = URI). This is preceded by a question mark (?) and may contain one or more name-value pairs separated by ampersands (&). }
|            | Example: page=homepage                                                                                                                                                                                        |
| cs-uri-stem | The stem portion of URI (stem + query string = URI). The stem is the actual or logical path to the requested resource on the server.                                                                           |
|            | Example: /index.asp                                                                                                                                                                                           |
| sc(content-type) | The content type of the resource being requested by the client as reported by the server.                                                                                                               |
|            | Examples: text/html, image/png, image/gif, video/mpeg                                                                                                                                                        |
| sc-bytes   | The number of bytes of data sent from the server to the client in response to the request                                                                                                                     |
|            | Example: 4996                                                                                                                                                                                                |
| sc-status  | The status code returned to the client by the server.                                                                                                                                                         |
|            | Example: 200                                                                                                                                                                                                  |
|            | Reference: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html                                                                                                                                             |
| s-dns      | The fully qualified domain name or IP address of the host of the requested resource.                                                                                                                        |
|            | Example: www.adobe.com                                                                                                                                                                                         |
| x-experiment | The list of all the controlled experiment names and groups that the client is a member of at the time of the request.                                                                                          |
|            | Example: VSHome_Exp.Group_1, VSRegistration_Exp.Group_2                                                                                                                                                      |
| x-timestamp | The date and time (GMT) at which the request was received by the server. The time is expressed as the number of 100 nanoseconds since January 1, 1600.                                                             |
|            | Example: 127710989320000000 would be the x-timestamp value for 11:28:52.0000000 on Tuesday, September 13, 2005.                                                                                            |
The 64-bit, hexadecimal value of the unique browser identifier found in a persistent cookie as set by a Sensor and provided by the client with a request to a server.
Example: 42FDF66DE610CF36

### Derived Fields
The table below lists examples of fields that are derived by the data workbench server from the baseline event data record fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-trackingid</td>
<td>The 64-bit, hexadecimal value of the unique browser identifier found in a persistent cookie as set by a Sensor and provided by the client with a request to a server. Example: 42FDF66DE610CF36</td>
</tr>
<tr>
<td>cs(cookie)(name)</td>
<td>The value of a given name-value pair within a cookie.</td>
</tr>
<tr>
<td>cs(referrer-domain)</td>
<td>The domain name or IP address of the HTTP referring URI. Note: This field is read-only.</td>
</tr>
<tr>
<td>cs(referrer-host)</td>
<td>The entire hostname of the referrer. Example: If cs(referrer) is <a href="http://my.domain.com/my/page">http://my.domain.com/my/page</a>, cs(referrer-host) is my.domain.com.</td>
</tr>
<tr>
<td>cs(referrer-query)(name)</td>
<td>The value of a referrer query string. Note: You cannot access a referrer query string value using the cs(referrer)(name) field.</td>
</tr>
<tr>
<td>cs-uri</td>
<td>The complete URI (stem + query string = entire URI). Example: /shopping/checkout.html?product1=8Track&amp;product2=casette&amp;product3=cd</td>
</tr>
<tr>
<td>cs-uri-query(name)</td>
<td>The value associated with the given name. If multiple values exist for the given name, this field returns the last of those values. Examples: • For the URI /shopping/checkout.html?product1=8Track&amp;product2=casette&amp;product3=cd, cs-uri-query(product3) would return cd. • For the URI /shopping/checkout.html?product1=8Track&amp;product1=casette, cs-uri-query(product1) would return cassette.</td>
</tr>
<tr>
<td>ctime</td>
<td>x-timestamp expressed as seconds since January 1, 1970. This field is also called x-unixtime.</td>
</tr>
<tr>
<td>date</td>
<td>x-timestamp in the format YYYY-MM-DD.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>time</td>
<td>x-timestamp in the format HH:MM:SS.</td>
</tr>
<tr>
<td>x-local-timestring</td>
<td>x-timestamp converted to the local timezone that is specified in the Transformation.cfg file for the dataset. The format is YYYY-MM-DD HH:MM:SS.mmm.</td>
</tr>
</tbody>
</table>

**Note:** You also can define time conversions such as `x-local-timestring` in the `Log Processing.cfg` file. For information, see *Log Processing Configuration File*.

| x-log-source-id | The identifier corresponding to the log source for a particular log entry. For the identifier to be recorded, you must specify it in the Log Source ID field of the Log Processing.cfg file when defining Sensor, log file, or ODBC data sources. For more information, see *Log Processing Configuration File*.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x-mask</td>
<td>The mask pattern of the Sensor data sources (derived from the .vsl file names). For a file whose name is of the format YYYYMMDD-SENSORID.VSL, x-mask is SENSORID.</td>
</tr>
<tr>
<td>x-timestring</td>
<td>x-timestamp in the format YYYY-MM-DD HH:MM:SS.mmm.</td>
</tr>
<tr>
<td>x-unixtime</td>
<td>The decimal UNIX time derived from x-timestamp.</td>
</tr>
</tbody>
</table>

**Sensor**, when used on a server, can collect fields of event data from any valid HTTP request or response header or variable available to it through the server's API. To collect such fields of data, you must specify the desired header fields or variables in the `txlogd.conf` configuration file for **Sensor**. For more information, see the *Data Workbench Sensor Guide*. 
Configuration Settings for Web Data

Many web-specific configuration settings for Site are defined in Log Processing and Transformation Dataset Include files.

Web-Specific Settings for Log Processing

Information about web-specific settings that are defined in Log Processing Dataset Include files that are delivered with Adobe profiles for Site.

The filtering defined by these settings occurs after the log entries leave the decoders and the transformations are applied but before evaluation by the Log Entry Condition.

- HTTP Status Filtering
- Robot Filtering

HTTP Status Filtering

You can configure your implementation of Site to remove log entries with sc-status codes of 400 or above from the dataset. Successful requests have status codes that are less than 400. Your default implementation includes a Log Processing Dataset Include file in which HTTP status filtering is configured.

To edit the configuration settings for HTTP status filtering

1. Open the Profile Manager within your dataset profile and open the Dataset\Log Processing\Traffic\HTTP Status Filter.cfg file.

   **Note:** If you have customized your implementation of Site, the file in which these configuration settings exist may differ from the location described.

2. Review or edit the values of the parameters of the file as desired. Use the following example as a guide.

```
<HTTP Status Filter.cfg
   <Log Processing Include LogProcessingInclude
   <Log Entry Condition AndCondition
   <0 RangeCondition
      Case Sensitive true
      Comments Comment
      Input sc-status
      Max 399
      Min 0
      Type NUMERIC
   
   Decoder Groups
   Fields
   Parameters
   Stage Default
   Transformations
```

For information about the Range condition, see Conditions.
3. Save the **HTTP Status Filter.cfg** file by right-clicking *(modified)* at the top of the window and clicking *Save*.

4. To make the locally made changes take effect, in the **Profile Manager**, right-click the check mark for the file in the **User** column, then click *Save to > <profile name>*. Where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

   **Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

**Robot Filtering**

You can configure your implementation of Site to use lookup files to remove log entries generated by known robots, test scripts, and IP addresses for internal users from your dataset. Your default implementation includes a **Log Processing Dataset Include** file in which robot filtering is configured.

**To edit the configuration settings for robot filtering**

1. Open the **Profile Manager** within your dataset profile and open the **Dataset\Log Processing\Traffic\Robot Filter.cfg** file.

   **Note:** If you have customized your implementation of Site, the file in which these configuration settings exist may differ from the location described.

2. Review or edit the parameters of the file using the following example and information as guides:

<table>
<thead>
<tr>
<th>-Robot Filter.cfg</th>
<th>LogProcessingInclude</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Log Processing Include</td>
<td>AndCondition</td>
</tr>
<tr>
<td>-Log Entry Condition</td>
<td>NotRobotCondition</td>
</tr>
<tr>
<td>Case Insensitive Robot Lookup File, Baseline</td>
<td>true</td>
</tr>
<tr>
<td>Robot Lookup File, Extended</td>
<td>$(Traffic Lookups)\Robots.txt</td>
</tr>
</tbody>
</table>

   The file includes a **NotRobotCondition** that is defined by the following three parameters:

   - **Case Insensitive Robot Filtering**: True or false. If true, letter case (upper/lower) is not considered in robot filtering.

   - **Robot Lookup File, Baseline**: The path and filename of the text file that contains a list of browser user agents that are known robots and are to be filtered out of the dataset. Adobe provides the baseline robot lookup file. If you do not specify a path, the data workbench server looks for this file in the Lookups directory within the data workbench server installation directory.

   - **Robot Lookup File, Extended**: The path and filename of an optional text file that contains a list of browser user agents or IP addresses that define robots specific to your implementation. This list can include internal
monitoring robots, test scripts, and IP addresses for internal users that should be filtered out of the dataset. If you do not specify a path, the data workbench server looks for this file in the Lookups directory within the data workbench server installation directory.

If a log entry’s browser user agent is not listed in either lookup file, the log entry is considered to be generated by a real visitor and is not filtered from the dataset.

**Note:** Matching in the robot lookup files uses substrings to compare against the c-ip and the cs(user-agent) log fields. If the search string starts with "$" it must match the front of the string being tested, and if it ends with "$" the search string must match the end of the string being tested. If the search string both begins with and ends with "$" the strings must match exactly for the log entry to be filtered out. For example, to test for all IP addresses in a class C block, you would use a string such as $231.78.123. to force a match at the front of the string. This would match addresses 231.78.123.0 through 231.78.123.255.

3. Save the file by right-clicking *(modified)* at the top of the window and clicking **Save**.

4. To make the locally made changes take effect, in the **Profile Manager**, right-click the check mark for the file in the **User** column, then click **Save to > <profile name>**, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

**Note:** If it is critical that the underlying log entries used to construct a dataset do not change (even if the transformations used to construct and update the dataset and its dimensions change), the Robot Lookup File, Baseline, and the Robot Lookup File, Extended, should be version controlled. Placing a version number on these files ensures that updates to the default robot lookup files do not unintentionally change previously constructed reporting datasets by adding or deleting entries in these files.

**Web-Specific Settings for Transformation**

Information about web-specific settings that are defined in **Transformation Dataset Include** files that are delivered with Adobe profiles for **Site**.

The conditions, dimensions, and parameters that are defined by these settings are created during the transformation phase of dataset construction.

- **Page View Condition**
- **URI Dimension**
- **Referrer Dimension**
- **Session Parameters**

**Page View Condition**

The **Page View Condition** is a condition operation that determines whether a particular log entry (that is, a page request) should be included in the data gathered about a visitor’s page view history. When the log entry satisfies the **Page View Condition**, it becomes an element of the Page View countable dimension. If a log entry does not satisfy the **Page View Condition**, its data fields still are accessible by other dimensions. In addition to the Page View dimension, the following dimensions can be affected by the results of the **Page View Condition**:
• URI and Page: These dimensions are directly affected by the **Page View Condition**. If the given page does not pass the Page View Condition, it is not be included in the URI or Page dimensions.

• Visitor Page Views and Session Page Views: The Visitor Page Views and Session Page Views dimensions are a count of the number of pages viewed by a visitor to or in a given session, respectively. Pages filtered out by the Page View Condition are not part of this count.

• Session Number: The Page View Condition has an indirect effect on the Session Number dimension. The Session Number dimension is created prior to the Page View Condition; therefore, when considering Session Number in relation to the Page Views, it is possible to have sessions with no page views.

Your default implementation of Site includes a Transformation Dataset Include file in which the Page View countable dimension and the related Page View Condition are defined.

For information about countable dimensions, see *Extended Dimensions*.

**To edit the configuration settings for the Page View Condition**

1. Open the Profile Manager within your dataset profile and open the `Dataset\Transformation\Traffic\Page View.cfg` file.

   Note: If you have customized your implementation of Site, the file in which these configuration settings exist may differ from the location described.

2. Review or edit the values of the parameters of the Page View Condition as needed. Use the following example as a guide. In this file, the Page View Condition is defined by a Copy transformation. Note that this file also contains the definition of the Page View countable dimension.

```plaintext
-Page View.cfg
  -Transformation Include TransformInclude
    -Extended Dimensions
      -0 Countable
        Name Page View
        Condition AndCondition
        Comments NotEmptyCondition
        Input x-is-page-view
        Comments Comment
        Hidden false
        Key
        Parent Session

  -Transformations
    -0 Copy
      Name Page View Condition
      Comments Comment
      Condition AndCondition
      Default
      Input "" "" x-is-page-view
      Output x-is-page-view
```
Note: For information about countable dimensions, see Extended Dimensions. For information about the Copy transformation, see Data Transformations.

3. Save the file by right-clicking (modified) at the top of the window, then click Save.

4. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

**URI Dimension**

If you are working with Site, you need to define the URI dimension whose elements are the URI stems of the website pages viewed. Your default implementation includes a Transformation Dataset Include file in which the URI simple dimension is defined.

For information about simple dimensions, see Extended Dimensions.

**To edit the configuration settings for the URI dimension**

1. Open the Profile Manager within your dataset profile and open the Dataset\Transformation\Traffic\URI.cfg file.

   Note: If you have customized your implementation of Site, the file in which these configuration settings exist may differ from the location described.

2. Review or edit the values of the parameters of the file as desired. Use the following example and information as guides.
The configuration settings for the URI dimension include the following two parameters:

- **Case Sensitive**: True or false. If true, letter case (upper/lower) is considered in identifying unique pages. The default value is true.

- **Maximum Elements**: The maximum number of elements (that is, URIs) for the URI dimension. The default value is 32768.

  *Note: Changing this value can cause serious performance issues. Do not change this value without consulting Adobe.*

- Save the `URI.cfg` file by right-clicking (modified) at the top of the window, then click **Save**.

- To make the locally made changes take effect, in the **Profile Manager**, right-click the check mark for the file in the **User** column, then click **Save to > <profile name>**, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.
**Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

**Referrer Dimension**

If you are working with **Site**, you need to define the Referrer dimension whose elements consist of the second level domains of the referrers of the first log entries in all sessions. Your default implementation includes a **Transformation Dataset Include** file in which the Referrer simple dimension is defined.

For information about simple dimensions, see *Extended Dimensions*.

**To edit the configuration settings for the Referrer dimension**

1. Open the **Profile Manager** within your dataset profile and open the *Dataset\Transformation\Traffic\Referrer.cfg* file.

   **Note:** If you have customized your implementation of **Site**, the file in which these configuration settings exist may differ from the location described.

2. Review or edit the values of the parameters of the file as desired. Use the following example and information as guides.

```
-Referrer.cfg
  -Transformation Include TransformationInclude
    -Extended Dimensions Simple
      -Name Referrer
      -Condition AndCondition
      -0 NeitherCondition
      -0 StringCondition
      -Matches 0 $(Internal Domains)
      Case Sensitive false
      Comments Comment
      Input cs(referrer-domain)
      Comments Comment
      Case Sensitive false
      Comments Comment
      Hidden false
      Input cs(referrer-domain)
      Load File
      Maximum Elements 0
      Operation FIRST ROW
      Parent Session
```

The configuration settings for the Referrer dimension include the Maximum Elements parameter, which specifies the maximum number of elements (that is, referrers) for the Referrer dimension. The default value is 32768.
Note: In the example above, the Maximum Elements parameter is set to 0. When this parameter is set to 0, the data workbench server uses its internal default value of 32768.

3. Save the Referrer.cfg file by right-clicking (modified) at the top of the window, then click Save.

4. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

Session Parameters
If you are working with Site, you can specify parameters that define the boundaries of a visitor's session on a website. These parameters are valid only when defined in a Transformation Dataset Include file within your Site implementation.

The following parameters are unique in that they can be members of Transformation Dataset Include file's Parameters vector, or they can be listed as individual parameters in the Transformation.cfg file. A parameter can be defined exactly once, so these parameters are defined either in the Transformation.cfg file or in the Parameters vector of the dataset include file - not in both files.

Maximum Session Duration and Session Timeout
Maximum Session Duration and Session Timeout are string parameters that define the length of a visitor's session. These parameters work with the Internal Domains parameter to determine session length.

Maximum Session Duration specifies the longest length of session before a new session is started. This keeps web pages that have auto content refreshing from creating sessions that are arbitrarily long. If the referrer of a click is set to one of the entries in the Internal Domains parameter, this timeout is used to define the end of a session. No session may be longer than the specified Maximum Session Duration regardless of how many clicks it contains. The recommended value is 48 hours.

Session Timeout specifies the amount of time that needs to pass between log entries of a given visitor to determine the end of one session and the start of a new session (that is, the typical timeout used to define a user session). The recommended value of this parameter is 30 minutes. If the referrer of a click is not set to one of the referrers in the Internal Domains parameter, this timeout is used to define the session. If cs(referrer-domain) for a log entry is in the list of internal domains, then Maximum Session Duration determines whether the current log entry is part of an existing session or the start of a new session.

Consider a situation in which a visitor is called away from his computer for a period of time longer than the Session Timeout while in the middle of browsing the site. Upon returning, he continues browsing where he left off. Because the visitor never leaves the site or closes his browser, the cs(referrer-domain) of his next click is the same as the internal domain, and his original session remains active as long as the Maximum Session Duration setting is not reached. If the domain of the site is listed as an internal domain, and the maximum timeout is not reached, the visitor's interaction appears as a single session and not two separate sessions. However, if the visitor returns to his computer and his next click has an external (or blank) referrer, a new session begins.

Note: The Sessionize transformation's Timeout Condition also plays a role in determining the length of a visitor's session. If Session Timeout and Maximum Session Duration do not apply, the Timeout Condition is
checked to determine whether a log entry should be considered the start of a new session. For more information, see Data Transformations.

To edit the Maximum Session Duration and Session Timeout parameters

If you are working with Site, your default implementation likely includes a Transformation Dataset Include file in which the names and recommended values of these parameters are specified.

1. Open the Profile Manager within your dataset profile and go to Dataset\Transformation\Traffic\Session Parameters.cfg.

   Note: If you have customized your implementation of Site, the file in which these parameters are defined may differ from the location described.

2. Edit the values of the parameters as desired. Be sure to specify the desired units (minutes, hours, and so on).

3. Save the Session Parameters.cfg file by right-clicking (modified) at the top of the window and clicking Save.

4. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or the inherited profile to which the dataset include file belongs.

   Note: Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your changes are overwritten when you install updates to these profiles.

Internal Domains

Internal Domains is a vector parameter that lists domain level hosts (internal referrers) that should be treated as part of a particular website. These hosts are removed from the referrer dimension (which is a list of the external referrer information). When cs(referrer-domain) matches any of the strings listed in the set of internal domains, Session Timeout is ignored and Maximum Session Duration is used to determine session length.

The Internal Domains parameter also can be used to prevent the start of a new session when visitors move among a company's multiple domains associated in a way that exceeds session timeout. For example, consider a company that has parts of its site split across two domains: one is logged (xyz.com), and the other is not logged (xyz-unlogged.com). If these sites are integrated in a way that facilitates the seamless movement of traffic across
the two domains, it is not desirable to generate a different session each time the visitor moves from
xyz-unlogged.com domain back to the xyz.com domain. Listing xyz-unlogged.com as an internal domain
keeps sessions from being split into multiple sessions as a result of traffic across these two domains as long as the
Maximum Session Duration setting is not reached.

To add an internal domain

If you are working with Site, your default implementation includes a Transformation Dataset Include file for defining
the Internal Domains parameter. In this file, the parameter is named; you just enter the internal domains that you
want to include and save the updated file.

1. Open the Profile Manager within your dataset profile and go to
   Dataset\Transformation\Traffic\Internal Domains.cfg.

   **Note:** If you have customized your implementation of Site, the file in which the Internal Domains parameter
   is defined may differ from the location described.

2. Right-click Value for the Internal Domains vector parameter and click Add new > Value.
3. Edit the values as desired.

```
   -Internal Domains.cfg
   |   -Transformation Include TransformationInclude
   |   |   -Parameters
   |   |   |   -Name Internal Domains
   |   |   |   -Value
   |   |   |       0  abcsite.com
   |   |   |       1  123site.com
   |   |   |       2  xyzsite.com
```

4. Save the Internal Domains.cfg file by right-clicking (modified) at the top of the window and clicking Save.
5. To make the locally made changes take effect, in the Profile Manager, right-click the check mark for the file in
   the User column, then click Save to > <profile name>, where profile name is the name of the dataset profile or
   the inherited profile to which the dataset include file belongs.

   **Note:** Do not save the modified configuration file to any of the internal profiles provided by Adobe, as your
   changes are overwritten when you install updates to these profiles.
Time Zone Codes

Information about the Time Zone codes and formats.

Most time-based parameters in the data workbench server are specified in the following format:

- Month DD, YYYY HH:MM:SS TZone
- Example: August 13, 2013 22:30:00 EST

Time zones are expressed in a system-independent time zone format (Coordinated Universal Time) of the following format:

- UTC +hhmm dstrules

The sign (+) can be either a plus (+) or a minus (-) sign, and hhmm is the offset from UTC in hours and minutes. The optional variable dstrules specifies a set of rules to implement Daylight Saving Time or a similar clock-shifting policy.

If you specify dstrules, a tab-delimited file named dstrules.dst must be present within the Dataset\TimeZone directory of either the Base profile (for configuration files that are not associated with a particular dataset) or the dataset profile (for configuration files that are dataset-specific). The file specifies a time-zone independent set of rules for Daylight Saving Time. You can have different sets of rules for different years. The DST.dst file provided by Adobe in the Base profile specifies the standard U.S. rules established by the Energy Policy Act of 2005 (in effect starting 2007) and the U.S. rules for prior years.

Sample time zone entries are listed below:

- U.S. Eastern Daylight Time: Time Zone = string: UTC -0500 DST
- UTC time with no offset and no dstrules (corresponding to GMT): Time Zone = string: UTC -0000

When this format is used, the system time zone of data workbench server, data workbench, and Report machines need not be the same as the specified time zone. In addition, all active dataset profiles on a data workbench server machine need not have the same time zone setting.

The following table contains the list of codes you can use to specify time zones in time-based parameters.

**Time Zone Code Table**

If you are implementing Daylight Saving Time or a similar clock-shifting policy, you must save the .dst file containing the appropriate rules in the profile name \Dataset\Timezone directory on the data workbench server machine.

<table>
<thead>
<tr>
<th>Code</th>
<th>Time Zone</th>
<th>Offset from GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmt</td>
<td>Greenwich Mean</td>
<td>0</td>
</tr>
<tr>
<td>est</td>
<td>Eastern Standard</td>
<td>5</td>
</tr>
<tr>
<td>edt</td>
<td>Eastern Daylight</td>
<td>5</td>
</tr>
<tr>
<td>cst</td>
<td>Central Standard</td>
<td>6</td>
</tr>
<tr>
<td>cdt</td>
<td>Central Daylight</td>
<td>6</td>
</tr>
<tr>
<td>mst</td>
<td>Mountain Standard</td>
<td>7</td>
</tr>
<tr>
<td>mdt</td>
<td>Mountain Daylight</td>
<td>7</td>
</tr>
<tr>
<td>pst</td>
<td>Pacific Standard</td>
<td>8</td>
</tr>
<tr>
<td>Code</td>
<td>Time Zone</td>
<td>Offset from GMT</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>pdt</td>
<td>Pacific Daylight</td>
<td>8</td>
</tr>
</tbody>
</table>
Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Marketing Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

• Check the Marketing Cloud help pages for advice, tips, and FAQs
• Ask us a quick question on Twitter @AdobeMktgCare
• Log an incident in our customer portal
• Contact the Customer Care team directly
• Check availability and status of Marketing Cloud Solutions

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions for the Analytics suite can be added to our Customer Idea Exchange.

Legal

© 2015, Adobe
All rights reserved.
Published by Adobe Systems Inc.

Terms of Use | Privacy Center

A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at http://www.adobe.com/go/thirdparty.

06052013