Orchestrate 7.0

WebHouse User Guide

Ascential Software Corporation

Orchestrate 7.0 WebHouse User Guide

September 2003

Ascential Software Corporation.
50 Washington Street
Westboro, MA 01581-1021
508 366-3888
508 389-8955 FAX

For technical support, send e-mail to: support@ascentialsoftware.com.

Contents

Preface

Describes this Guide and the conventions that it uses.

About This Guide

This book is a reference for anyone who uses Orchestrate to process web server clickstream logs. To use this book, you should have a basic familiarity with Orchestrate. The *Orchestrate 7.0 User Guide* describes the Orchestrate Shell (**osh**), the UNIX command-line interface to Orchestrate.

Organization of This Book

Chapter 1, "Introduction to WebHouse" gives you a summary of the extraction, transformation, and loading (ETL) capabilities of WebHouse for web server clickstream logs. It summarizes the expected log formats, tells you what output data sets are generated, and briefly outlines the functionality of each of the WebHouse operators. Finally, it presents an example WebHouse data flow.

Chapter 2, "WebHouse Operators" describes each WebHouse operator in detail.

The Orchestrate Documentation Set

These documents are available both in hardcopy and online PDF format.

- *Orchestrate 7.0 User Guide*
- *Orchestrate 7.0 Operators Reference*
- *Orchestrate 7.0 Installation and Administration Manual*
- *Orchestrate 7.0 Developer Guide*
- *Orchestrate 7.0 WebHouse User Guide*
- *Orchestrate 7.0 Record Schema*
- *Orchestrate 7.0 C++ Classes and Macros Sorted by Header File*
- *Orchestrate 7.0 C++ Classes and Macros Sorted by Name*

Typographic Conventions

Typographic Formats

Table 1 **Typographic Formats**

Format	Term	Examples
bold serif	Orchestrate-defined terms including class and function names, data types, and default field names.	**APT_Operator** **runLocally()** **APT_Operator::runLocally()** **cluster_predicted** **uint64**
	Orchestrate-defined macros	**APT_DECLARE_PERSISTENT**

Table 1 **Typographic Formats (continued)**

Format	Term	Examples
	Orchestrate-defined operators, operator options, and collection and partition methods	The **clusterquality** operator takes the **results** and **flatten** options. Use the **any** or **round robin** collection method.
bold italic serif	Orchestrate technical terms within the text that defines them	In ***pipeline parallelism***, each operation runs when it has input data available.
sans serif	Orchestrate menus and other GUI elements	**Tools > Check Config**
`fixed width`	C++ code	`return APT_StatusOk;`

	operating-system commands	`$ cd $APT_ORCHHOME/bin`
		`$ rm aptserv2`

	osh commands	`$ osh "clusterquality < kmeans.ds”`
	non-Orchestrate-defined C++ functions	`sprintf()`

italic	user-defined classes, functions, and field names	*MyOperator*
		selectState()
		MyOperator::selectState()
		Price

	Orchestrate variables	Specify an *insert_statement*

Cross-References

Most cross-references indicate sections located in this book. They are hotspots and appear in blue typeface in the online version of the document. When there are references to other books in the documentation set, their names appear in italics.

Syntax Conventions

Operator syntax is presented as you would enter it as part of an **osh** command. For a description of the general syntax of an **osh** command, refer to the *Orchestrate 7.0 User Guide*.

The following syntax conventions are used throughout this book:

- A vertical bar (|) separates mutually exclusive alternatives.
- Braces ({ }) are used with vertical bars (|) to indicate that one of several mutually exclusive alternatives are required, for example **{a | b}** indicates that **a** or **b** is required.
- If one or more items are enclosed in braces ({ }) and separated by commas the items are synonymous. Thus **{a , b}** indicates that **a** and **b** have exactly the same effect.
- Brackets (**[ ]**)indicate that the item(s) inside are optional. For example, **[a | b]** indicates **a**, or **b**, or neither. Occasionally, brackets do not indicate optional items but are part of the syntax. In these cases, the special nature of the brackets will be explicitly stated.
- Ellipsis (**...**) indicates that the preceding item occurs zero or more times. If a user-provided string is indicated, it may represent a different item in each occurrence. For example:
  - `[-key` *field* `...]` means *zero* or more occurrences of **-key** *field*, where *field* may be a different string in each occurrence.
  - To indicate *one* or more occurrences of an item, the item occurs first without brackets and ellipsis and then with, for example

    `-key` *field* `[-key` *field* `...]`

Using the Adobe Acrobat Reader

To read the document set online, you need the Adobe Acrobat Reader. If you already have Adobe Acrobat Reader installed, make sure it is version 4.05 or later and is the version of Reader with Search. If your version of Reader does not have Search, you will not be able to search across all the documents in the Orchestrate set.

To see if your copy of Reader has Search, look for the **Search** icon: on the Reader toolbar, and make sure it is present and not dimmed. The **Search** icon should be located alongside the somewhat similar **Find** icon: .

If you do not have the appropriate version of Acrobat installed, you may use the Acrobat Reader 4.05 included with Orchestrate. Find the version you need in one of the following platform-specific directories:

```
$APT_ORCHHOME/etc/acroread-sun-405.tar.gz
$APT_ORCHHOME/etc/acroread-aix-405.tar.gz
$APT_ORCHHOME/etc/acroread-osf1-405.tar.gz
$APT_ORCHHOME/etc/acroread-hpux-405.tar.gz
```

Use the UNIX `gunzip` and `tar` commands to unpack the files. Then `cd` to the directory `*.install` (where `*` contains an abbreviation of your platform name) and follow the instructions in the `INSTGUID.TXT` file.

The Orchestrate online documentation set and the full-text search index is located in `$APT_ORCHHOME/doc`.

Searching for Text in Orchestrate Documents

You can find specific words or phrases in this Guide and across all Orchestrate online documents using the Adobe Acrobat Reader.

■ **To find text in a single document:**

1. Open the document in the Adobe Acrobat Reader.
2. Choose **Edit > Find**
   or
   Click **Find** on the Adobe Toolbar
3. Enter the text you want to find in the **Find What** field and click **Find**.
4. Use **Ctrl+G** to find the next occurrence of the text.

■ **To find text across all documents:**

1. Choose **Edit > Search > Query**
   or
   Click **Search** on the Acrobat toolbar
2. In the **Adobe Acrobat Search** window, type the text you want to find in the **Find Results Containing Text** box, then click **Search**.
   a. If the following message appears at the bottom of the **Adobe Acrobat Search** window:
      ```
      No selected indexes are available for search
      ```
      Click **Indexes...** to bring up the **Index Selection** window.
   b. Click **Add**, then navigate the **Add Index** window to find the full-text search index, `Orchestrate.pdx`, located in `$APT_ORCHHOME/doc`.
   c. Select the index, then click **OK** on the **Index Selection** window.
3. The **Search Results** window lists the documents that contain your search text, ordered according to the number of search hits.
4. Select a book in the list.
5. Use the **Previous Highlight** and **Next Highlight** buttons to move to the previous or next instances of your search text.

Assistance and Additional Information

If you require assistance or have questions about Orchestrate, you can contact Ascential Customer Support by:

- Calling (866) INFONOW
- Writing support@ascential.com for any Ascential Software product or orch-support@ascential.com for Orchestrate-specific help.
- Logging onto the Ascential Support e.Service Web site at:

  www.ascential.com:8080/eservice/index.html

For current information about Ascential and its products log onto:

http://www.ascential.com/

1 Introduction to WebHouse

Gives an introduction to WebHouse and presents a sample data flow.

Overview

WebHouse is a set of Extraction, Transformation, and Loading (ETL) operators you use within Orchestrate to process web server clickstream logs. The operators support most web server log formats that contain elements found in the Common Log Format (CLF) and the W3C Extended Common Log Format (ECLF).

The output dataset is the product of the **sessionize** operator which separates clickstream records into groups. Each group of records represents a series of web content views accessed by a single user during a web session bounded by a termination condition. Each record in a group is given a common session ID.

Two optional output datasets are also available: one contains records of session-summary data and the other contains dwell-time per page data. You can use the output datasets for generating filtered reports and for performing online analytical processing and data mining.

The six operators that are specific to WebHouse are summarized in the table below. They are described in detail in the next chapter.

Table 2 **A Summary of the WebHouse Operators**

Operator	Description
clf_reformat	Reformats the **date_time** input field into **local_timestamp** and **gmt_timestamp** fields, and processes the **http_command** input field to produce **http_operation**, **uri_string**, and **http_protocol_version** fields.
cookie_extract	Extracts individual cookie values from raw cookie data.
web_create_uid	Builds the **user_id** field based on extracted cookie fields or sets of cookie fields, or on the **ip_address** and **user_agent** input fields.
query_extract	Extracts values from the **uri_string** input field.
perlbuildop	Creates custom operators using an embedded Perl interpreter.
sessionize	Identifies and tags logical user sessions in web logs for the output dataset, and optionally outputs session-summary and dwell-time datasets.

Building a web log processing application within Orchestrate also requires the general Orchestrate operator set which is described in the *Orchestrate 7.0 Operators Reference.*

In particular, the operators listed below are useful. The **filter** operator has been enhanced to support WebHouse operators. The enhancements are outlined in "The filter Operator" on page 1-6.

- **filter**
- **group**
- **hash**
- **import**
- **sortfunnel**
- **lookup**

Example Data Flow

This section outlines an example WebHouse data flow. The example is included to introduce you to the basic functionality and a sample sequencing of the operators. You can utilize the operators in multiple ways to develop your own Orchestrate web log processing application. This section lists the typical input fields, summarizes content mapping, and briefly describes each operator in the order it appears in the example. A conceptual diagram of the sample data flow is shown in "Conceptual Illustration of a Typical WebHouse Application" on page 1-8.

Web Server Log File Input

The general flow of a WebHouse application begins with input data streams, each of which corresponds to a server log file set from a web server in a server farm. There may be any number of log files in a single log file set, but the files within a set must be in time-series order. The web server log records may contain any number of fields.

These are the web log file data required for the WebHouse example in this chapter:

- timestamp
- URI, if a content map field is not provided
- raw cookie, which is highly recommended and contains any number of user-defined cookie *name:value* pairs
- client IP address
- user-agent data

Content Mapping

Content mapping is the process of tagging the clickstream records with identifiers that define the content the user is viewing. Content may be defined as a page view, a product view, an area of a site, and so on. With static sites, it may be possible to simply use a raw URI as a content identifier, but with dynamically generated sites this is not sufficient.

You might encounter a condition in which multiple URIs show the same logical content. In such a case, you must define a table that maps a subset of the extracted URI query values to corresponding content identifiers.

Determining User Identification

User identification is resolved by the **web_create_uid** operator using either the cookie fields that are output from the **cookie_extract** operator or the **user_agent** and **ip_address** fields.

The **web_create_uid** operator determines the value of the **user_id** field using these prioritized steps:

1. It uses the first non-null cookie field specified as an option.
2. It concatenates the **ip_address** and **user_agent** input fields when all specified cookie fields are null but the **user_agent** field has a value.
3. It uses the **ip_address** input field.

Determining User Sessions

The **sessionize** operator creates logical groupings of records for later analysis. Each grouping represents a series of web-content views accessed by a single user during a web session. Here is an example of how **sessionize** determines user sessions:

> A web server collects page requests in a log file. The log file contains both a cookie and a timestamp. Many users make requests from the server simultaneously, so their requests are interspersed in the log file. In this case, the users are asked to request a special logout page when they are done using the server.
>
> To profile how different users utilize the server, the actions of each individual user need to be collected into logical groups, or sessions. An individual user is identified by a unique cookie which remains constant over multiple sessions. In this case, the **sessionize** operator relies on the logout page requests to differentiate the sessions. The **sessionize** operator assigns each record in a user session a common session ID.

The **sessionize** operator cannot always depend on special logout page requests to determine session boundaries. When logout is voluntary, the operator relies on time-out intervals or having one or more fields match a configured boundary expression. For the time-out interval, the operator, by default, uses the web standard of a 30-minute time out, but you can specify any number of minutes for the interval. If a user has not accessed the server within the time-out interval and then accesses it again, **sessionize** identifies that access as the first record of a new session and gives it a new session ID.

Operator Functionality

The import Operator

Each of the data streams begins with a single **import** operator. The importer reads the files in the log-file set in sequential order. The data in each of these files is required to be sorted on the timestamp field, which is the standard order of records in web-server log files.

The clf_reformat Operator

The imported data is then fed into a **clf_reformat** operator that standardizes log file formats and derives **local_timestamp** and **gmt_timestamp** fields from the input **date_time** field, and derives **http_operation**, **uri_string**, and **http_protocol_version** fields from the input **http_command** field.

The cookie_extract Operator

The standardized output is then passed to a **cookie_extract** operator for parsing of the raw cookie field, which contains multiple cookies. You specify which of the cookie fields to retain, and the operator then places each individual cookie into a separate Orchestrate field. If possible, one of these cookie values is designated the user ID.

The web_create_uid Operator

The extracted cookie fields are then fed into a **web_create_uid** operator to create the **user_id** field for the records. This operator can set the **user_id** to a specified cookie value, if one exists. If no cookie value exists, the input **ip_address** field is used in combination with the **user_agent** field. If a **user_agent** field is not present, the **ip_address** alone is used.

The **ip_address** and **user_agent** combination is not recommended because it is not a reliable key for sessionization. If possible, the user ID should be determined from a cookie that can uniquely identify a user.

The hash Partitioner Operator

The data is now hash partitioned on the **user_id** field.

The sort_funnel Operator

Up until this point, each of the data flows has been running sequentially. Once the data is hash-partitioned, the resulting parallel streams are fed into a multi-input/one-output **sort_funnel** operator. The operator uses the timestamp field as a sort key. The result is a single parallel stream in timestamp order.

The filter Operator

Now the data stream can optionally be fed into a **filter** operator to remove any unwanted records. Unwanted records may include image requests, requests made by known spiders, and so on.

To support WebHouse, the following features have been added to the **filter** operator:

- You can define one or more WHERE clauses using the **where** suboption.
- The **target** suboption of **where** allows you to specify the target dataset for a WHERE clause.
- The **first** option makes it possible to output records only to the data set corresponding to the first matched WHERE clause.

The query_extract and perbuildop Operators

The output is then passed to a **query_extract** operator which parses the query string in the **uri_string** field into multiple Orchestrate fields, each one corresponding to a single query value. If the input **uri_string** field query strings are in a nonstandard format, a **perbuildop** operator can be used instead of, or in addition to, the **query_extract** operator.

The lookup Operator

The output of the **query_extract** operator is fed into a **lookup** operator that uses the query value/content table to append content IDs to the clickstream records. At this point, the data can optionally be fed into another **filter** operator to remove all records that did not get a valid content identifier appended to them.

The sessionize Operator

From here the data is fed into a **sessionize** operator, which appends a **session_id** field to the records. A session is defined as a sequence of content views for the same **user_id**, where the subsequent views are requested within a specified time period or until one or more fields match a configured boundary expression. A boundary condition is specified in the same manner as a regular expression, similar to a WHERE clause in the **filter** operator. See the *Orchestrate 7.0 Operators Reference* for information about the `WHERE` clause.

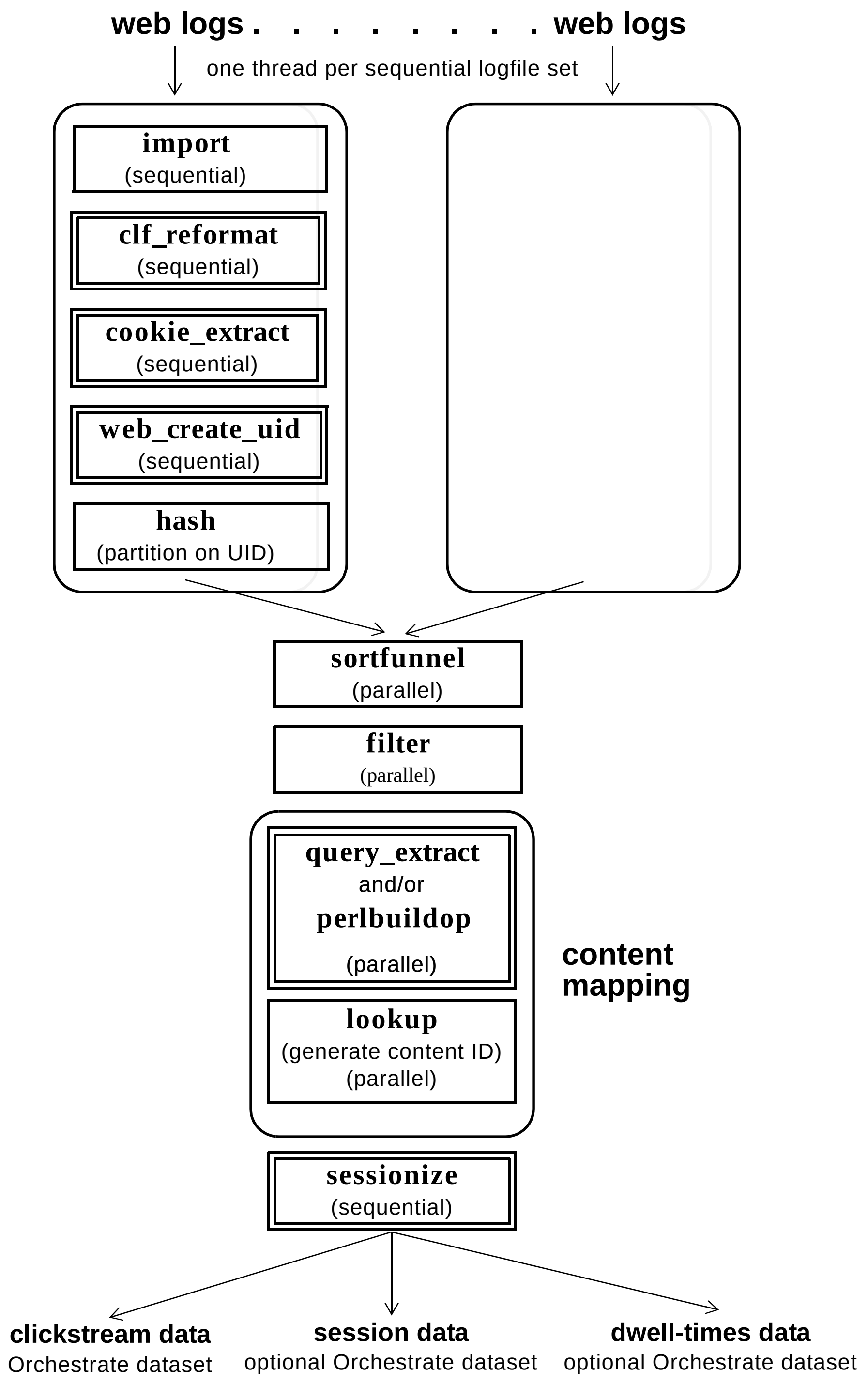
The **sessionize** operator outputs up to three datasets:

- The first dataset, which is always generated, is the clickstream in timestamp order with a **session_id** field added to the records.
- The second dataset is optional. It outputs session-level statistics and contains these fields: **user_id**, **session_id**, **duration** (in seconds), **boundary**, which is an index of the session-terminating boundary condition, and **records**, which is the number of records within the session.
- The third dataset is also optional. It contains dwell times for content views within sessions. Since there is no way of knowing the dwell time for the last content request for a session, the dwell time for this view is output as NULL. For each record, this dataset contains fields for **user_id**, **session_id**, **uristem** (or content identifier), and **duration**, which is dwell time in seconds. Users can optionally specify that other fields from the clickstream input be added to the output record.

At this point, any other aggregations or processing you specify can be performed on the sessionized clickstream records using standard Orchestrate operators. The data can then be loaded into a data warehouse using the standard Orchestrate database-load operators.

Using other standard Orchestrate operators, other data sources can be integrated into the clickstream data such as transaction data from an eCommerce site or registered user information. In such cases, the data is usually joined to the clickstream dataset using the timestamp and cookie fields.

Conceptual Illustration of a Typical WebHouse Application



2

WebHouse Operators

Contains a detailed description of each WebHouse operator.

Overview

This chapter describes the six WebHouse operators provided by Orchestrate. The operators support most web server log formats that contain elements found in the Common Log Format (CLF) and the W3C Extended Common Log Format (ECLF).

The operators specific to WebHouse summarized in the table below.

Table 3 **Short Descriptions of the WebHouse Operators**

Operator	Description	Page
clf_reformat	Reformats the **date_time** input field into **local_timestamp** and **gmt_timestamp** fields, and processes the **http_command** input field to produce **http_operation**, **uri_string**, and **http_protocol_version** fields.	**2 4**
cookie_extract	Extracts individual cookie values from raw cookie data.	**2 8**
web_create_uid	Builds the **user_id** field based on extracted cookie fields or sets of cookie fields, or based on the **ip_address** and **user_agent** fields.	**2 12**
query_extract	Extracts values from the **uri_string** input field.	**2 16**
perlbuildop	Creates custom operators using an embedded Perl interpreter.	**2 21**

Table 3 **Short Descriptions of the WebHouse Operators (continued)**

Operator	Description	Page
sessionize	Identifies and tags logical user sessions in web logs for the output dataset, and optionally outputs session-summary and dwell-time datasets.	**2 30**

In addition, building a web log processing application with Orchestrate requires the general Orchestrate operator set which is described in the *Orchestrate 6.1 Operators Reference*. In particular, the operators listed below are very useful. The **filter** operator has been enhanced to support WebHouse operators. The enhancements are outlined in "The filter Operator" on page 1-6 of the previous chapter.

- **filter**
- **group**
- **hash**
- **import**
- **sortfunnel**
- **lookup**

The clf_reformat Operator

The **clf_reformat** operator reads records from a single input and writes them to a single output. For each record read in, the **date_time** input field is processed to produce **gmt_timestamp** and **local_timestamp** output fields. The **date_time** input field is expected to be formatted in CLF or ECLF. The **http_command** input field is processed to produce **http_operation**, **uri_string**, and **http_protocol_version** output fields.

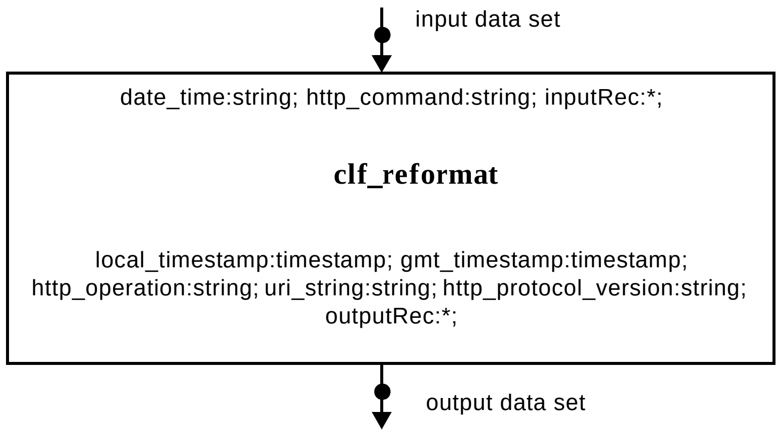
The following table shows how the output-field values are derived.

Table 4 **clf_reformat Operator Output-Field Value Derivations**

This output field	derives its value from
local_timestamp	the first token in the **date_time** input field
gmt_timestamp	the second token in the **date_time** input field
http_operation	the first token in the **http_command** input field
uri_string	the second token in the **http_command** input field
http_protocol_version	the third token in the **http_command** input field

The two input fields, **date_time** and **http_command**, are dropped from the output.

Data Flow Diagram


input data set
date_time:string; http_command:string; inputRec:*;
clf_reformat
local_timestamp:timestamp; gmt_timestamp:timestamp; http_operation:string; uri_string:string; http_protocol_version:string; outputRec:*;
output data set

Properties

Table 5 **clf_reformat Operator Properties**

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	record (date_time:string; http_command:string; inputRec:*;)
Output interface schema	record (gmt_timestamp:timestamp; local_timestamp:timestamp; http_operation:string; uri_string:string; http_version:string; outputRec:*;) The **date_time** and **http_command** fields are dropped from the output interface schema.
Transfer behavior	Input is copied to output with reformatting as specified
Execution mode	Parallel (default) or sequential
Partitioning method	Any (parallel mode)
Collection method	Any (sequential mode)
Preserve-partitioning flag in output data set	Propagated
Composite operator	No

Syntax and Options

The syntax for the **clf_reformat** operator in an osh command is shown below. All **clf_reformat** options are optional.

```
clf_reformat
    [-date_time_input fieldname]
    [-gmt_timestamp_output fieldname]
    [-http_command_input fieldname]
    [-http_operation_output fieldname]
    [-http_version_output fieldname]
```

```
[-local_timestamp_output fieldname]
[-uri_string_output fieldname]
```

Table 6 **clf_reformat Operator Options**

Option	Use
-date_time_input	`-date_time_input` *fieldname* Optionally specifies the name of the date-time input field; the default is **date_time** **.**
-gmt_timestamp_output	`-gmt_timestamp_output` *fieldname* Optionally specifies the name of the GMT timestamp output field; the default is **gmt_timestamp** **.**
-http_command_input	`-http_command_input` *fieldname* Optionally specifies the name of the HTTP command input field; the default is **http_command** **.**
-http_operation_output	`-http_operation_output` *fieldname* Optionally specifies the name of the HTTP operation output field; the default is **http_operation** **.**
-http_version_output	`-http_version_output` *fieldname* Optionally specifies the name of the HTTP protocol version output field; the default is **http_version** **.**
-local_timestamp_output	`-local_timestamp_output` *fieldname* Optionally specifies the name of the local timestamp output field; the default is **local_timestamp** **.**
-uri_string_output	`-uri_string_output` *fieldname* Optionally specifies the name of the uri string output field; the default is **uri_string** **.**

Example

For this `osh` syntax:

```
clf_reformat < input.dat > output.dat
```

when `input.dat` is:

```
record 1:
   date_time (nullable string): 20/Dec/1999:22:42:07 -0500
   http_command (nullable string): GET / HTTP/1.0
record 2:
   date_time (nullable string): 31/Dec/1999:21:58:52 -0400
   http_command (nullable string): GET /design_left.html HTTP/1.1
```

`output.dat` is:

```
record 1:
   local_timestamp (nullable timestamp): 1999-12-20 22:42:07
   gmt_timestamp (nullable timestamp): 1999-12-21 03:42:07
   http_operation (nullable string): GET
   uri_string (nullable string): /
   http_version (nullable string): HTTP/1.0
record 2:
   local_timestamp (nullable timestamp): 1999-12-31 21:58:52
   gmt_timestamp (nullable timestamp): 2000-01-01 01:58:52
   http_operation (nullable string): GET
   uri_string (nullable string): /design_left.html
   http_version (nullable string): HTTP/1.1
```

The cookie_extract Operator

In web server logs, the cookie field has this format:

```
name1=value1; name2=value2; name3=value3; ...
```

Each individual cookie to be used in further processing is extracted from the log to a separate raw input cookie field. The **cookie_extract** operator then extracts each specified cookie from the raw input cookie field and puts the cookie value in a specified output field. By default, the raw input cookie field is dropped.

All cookie fields to be extracted must be specified as arguments to the operator. Any cookies that are not present in a specific input record are output as NULL field values, and any cookies that are not specified as values to be extracted are ignored. By default, the cookie output field names are the same as the cookie names, but you can override this. The cookie values are all strings.

For example, the raw cookie input field:

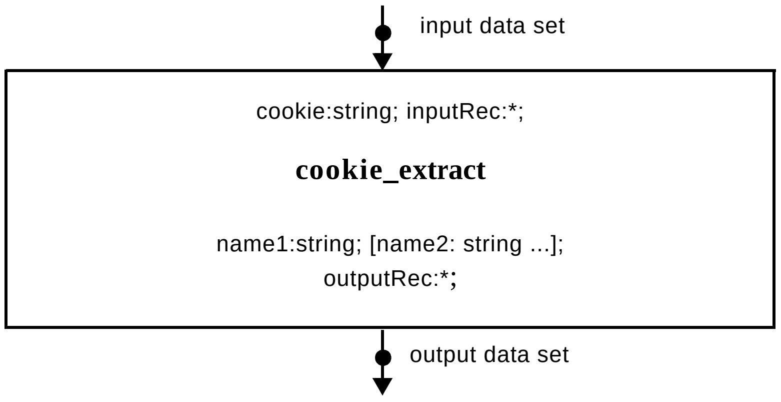
```
name1=value1; name2=value2; name3=value3
```

would be output as follows, provided that all three cookies are specified as arguments to the operator:

```
name1:string_value1;
name2:string_value2;
name3:string_value3
```

The **cookie_extract** operator reads records from one input and writes them to one output.

Data Flow Diagram


input data set
cookie:string; inputRec:*;
cookie_extract
name1:string; [name2: string ...];
outputRec:*;
output data set

Properties

Table 7 **cookie_extract Operator Properties**

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	record (cookie:string; inputRec:*;)
Output interface schema	record (name1:string; [name2: string ...;] outputRec:*;) By default, the input cookie field is dropped.
Transfer behavior	Input is copied to output if the **keep_cookie_input** option is set, otherwise the cookie field is dropped
Execution mode	Parallel (default) or sequential
Partitioning method	Any (parallel mode
Collection method	Any (sequential mode)
Preserve-partitioning flag in output data set	Propagated
Composite operator	No

Syntax and Options

The syntax for the **cookie_extract** operator in an osh command is shown below:

```
cookie_extract
    -cookie_output cookie_name
        [-cookie_field fieldname]
    [-cookie_output cookie_name
        [-cookie_field fieldname ] ...
    [-cookie_input fieldname]
    [-keep_cookie_input]
```

Table 8 **cookie_extract Operator Options**

Option	Use
-cookie_input	`-cookie_input` *fieldname* Optionally specifies the name of the raw cookie input field; the default is **cookie .**
-cookie_output	`-cookie_output` *cookie_name* `[-cookie_field` *fieldname*`]` Specifies the name of a cookie to be extracted. There can be any number of occurrences of this option. The optional suboption **-cookie_field** specifies the name of the output field that holds the corresponding cookie value. The default is the **cookie_name**.
-keep_cookie_input	`-keep_cookie_input` Optionally specifies that the raw cookie input field should be kept in the output records. The default is to drop it.

Example

For this osh syntax:

```
cookie_extract -cookie_output cookie1
               -cookie_output cookie2
               -cookie_output cookie3 < input.dat > output.dat
```

when `input.dat` is:

```
record 1:
   cookie (nullable string): cookie1=value11
   extra_data (string): aaaaa
record 2:
   cookie (nullable string): cookie1=value21; cookie2=value22
   extra_data (string): bbbbb
record 3:
   cookie (nullable string): cookie1=value31; cookie3=value33
   extra_data (string): ccccc
record 4:
   cookie (nullable string): cookie1=value41; cookie2=value42;
      cookie3=value43
   extra_data (string): ddddd
```

`output.dat` is:

```
record 1:
   cookie1 (nullable string): value11
   cookie2 (nullable string): NULL
   cookie3 (nullable string): NULL
   extra_data (string): aaaaa
record 2:
   cookie1 (nullable string): value21
   cookie2 (nullable string): value22
   cookie3 (nullable string): NULL
   extra_data (string): bbbbb
record 3:
   cookie1 (nullable string): value31
   cookie2 (nullable string): NULL
   cookie3 (nullable string): value33
   extra_data (string): ccccc
record 4:
   cookie1 (nullable string): value41
   cookie2 (nullable string): value42
   cookie3 (nullable string): value43
   extra_data (string): ddddd
```

The web_create_uid Operator

The **web_create_uid** operator reads records from one input and writes them to one output.

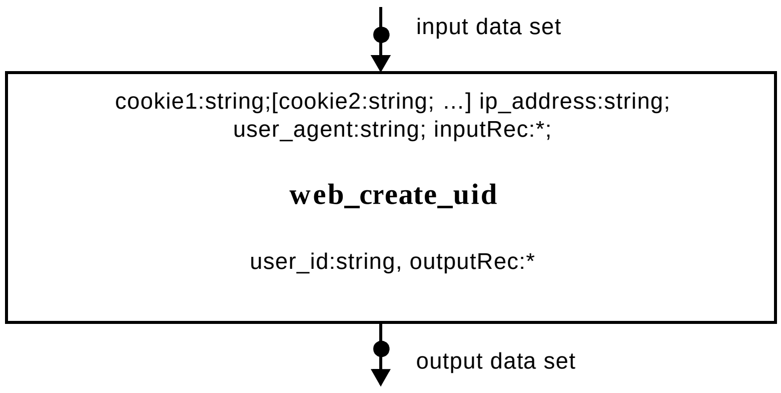
This operator creates the ouput **user_id** field from a set of one or more extracted cookie fields or from the **ip_address** and **user_agent** fields.

If any of the specified cookie fields are non-null, the first non-null value, based on option order, is copied into the **user_id** field. If all the specified cookie fields are null and the **user_agent** field is not null, the **ip_address** and **user_agent** fields are concatenated to form the **user_id**. If all the cookie fields and the **user_agent** field are null, the **ip_address** field is copied into the **user_id** field.

If a **-cookie_input** option has been set for the operator, the corresponding cookie field must be present in the input dataset schema.

The resulting output record contains all the fields in the input record, plus the **user_id** field.

Data Flow Diagram


input data set
cookie1:string;[cookie2:string; ...] ip_address:string; user_agent:string; inputRec:*;
web_create_uid
user_id:string, outputRec:*
output data set

Properties

Table 9 **web_create_uuid Operator Properties**

Property	Value
Number of input data sets	1
Number of output data sets	1

Table 9 **web_create_uuid Operator Properties (continued)**

Property	Value
Input interface schema	record (cookie1:string; [cookie2:string; ...] ip_address:string; user_agent:string; inputRec:*;)
Output interface schema	record (user_id:string; outputRec:*;)
Transfer behavior	Input is copied to output
Execution mode	Parallel (default) or sequential
Partitioning method	Any (parallel mode)
Collection method	Any (sequential mode)
Preserve-partitioning flag in output data set	Propagated
Composite operator	No

Syntax and Options

The syntax for the **web_create_uid** operator in an osh command is shown below. All **web_create_uid** options are optional.

```
web_create_uid
    [-cookie_input fieldname ...]
    [-ip_address_input fieldname]
    [user_agent_input fieldname]
    [user_id_output fieldname]
```

Table 10 **web_create_uid Operator Options**

Option	Use
-cookie_input	`-cookie_input` *fieldname* Specifies the name of the cookie input field. There can be any number of occurrences of this option. The default name is **cookie.**
-ip_address_input	`-ip_address_input` *fieldname* Specifies the name of the IP address input field; the default is **`ip_address`.**

Table 10 **web_create_uid Operator Options (continued)**

Option	Use
-user_agent_input	`-user_agent_input` *fieldname* Specifies the name of the user-agent input field; the default is **`user_agent`**.
-user_id_output	`-user_id_output` *fieldname* Specifies the name of the user ID output field; the default is **`user_id`**.

Example

For this `osh` syntax:

```
web_create_uid
   -cookie_input cookie_a
   -cookie_input cookie_b < input.dat > output.dat
```

When `input.dat` is:

```
record 1:
   cookie_a (nullable string): AAAAA
   cookie_b (nullable string): 11111
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
record 2:
   cookie_a (nullable string): NULL
   cookie_b (nullable string): 22222
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
record 3:
   cookie_a (nullable string): CCCCC
   cookie_b (nullable string): NULL
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
record 4:
   cookie_a (nullable string): NULL
   cookie_b (nullable string): NULL
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
```

`output.dat` is:

```
record 1:
   user_id (nullable string): AAAAA
   cookie_a (nullable string): AAAAA
   cookie_b (nullable string): 11111
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
```

```
record 2:
   user_id (nullable string): 22222
   cookie_a (nullable string): NULL
   cookie_b (nullable string): 22222
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
record 3:
   user_id (nullable string): CCCCC
   cookie_a (nullable string): CCCCC
   cookie_b (nullable string): NULL
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
record 4:
   user_id (nullable string): 208.206.40.191Mozilla/4.0
   cookie_a (nullable string): NULL
   cookie_b (nullable string): NULL
   ip_address (nullable string): 208.206.40.191
   user_agent (nullable string): Mozilla/4.0
```

The query_extract Operator

A URI generally has the following format:

```
location?name1=value1&name2=value2&name3=value3.....
```

Everything before the question mark is related to the location of the site that is being requested. Everything after the question mark is the query string. A query string may also be passed as a separate string field.

The **query_extract** operator breaks down the query string into columns of values and name:value pairs. In a query string, each name:value pair has a delimiter that is used by the **query_extract** operator to distinguish the pairs. In the example above, the column delimiter is the ampersand character (&). The association between a name and a value is determined by an association delimiter. The equal character (=) is the association delimiter used in the example. Both association and column delimiters are customizable options of the **query_extract** operator.

The **query_extract** operator extracts each variable from the raw URI field and puts its value in the proper output field. All query fields to be extracted must be specified as arguments to the operator. Specification order does not matter if all variables are name:value pairs; however, if any variables are value only, the fields must be specified in order.

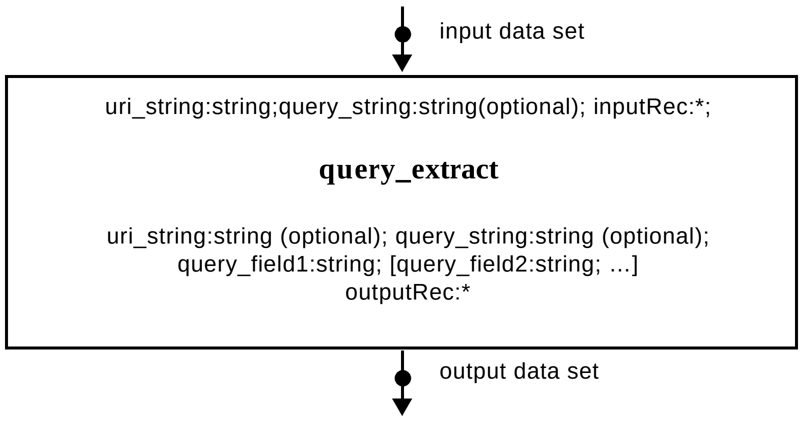
Any query fields that are not present in a specific record are output as NULL field values. Any query variables that are not recognized are ignored. By default, the raw input query field is dropped.

By default, each query field is assumed to be a value only, but you can override this if the query field is a name:value pair. For a name:value pair, the query field name is assumed to be the same as the output field name, but you can also override this by specifying a different name.

A value extracted from the query string is assumed to be URI encoded, where spaces are represented by plus (+) characters and all characters considered to be invalid by the HTTP specification are replaced with a percent (%) character followed by a two-digit hexadecimal version of their ASCII value.

The **query_extract** operator reads records from one input and writes them to one output.

Data Flow Diagram


input data set
uri_string:string;query_string:string(optional); inputRec:*;
query_extract
uri_string:string (optional); query_string:string (optional); query_field1:string; [query_field2:string; ...] outputRec:*
output data set

Properties

Table 11 **query_extract Operator Properties**

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	record (uri_string:string; query_string:string; inptRec:*;) Note that **query_string** is optional.
Output interface schema	record (uri_string:string; query_string:string; query_field1:string; . . .query_field*N*:string; outputRec:*;) Note that **uri_string** and **query_string** must be explicitly requested using the **keep_query_string** flag.
Transfer behavior	Input is copied to output if the **keep_query_string** flag is set, otherwise the query string is removed.
Execution mode	Parallel (default) or sequential
Partitioning method	Any (parallel mode)
Collection method	Any (sequential mode)

Table 11 **query_extract Operator Properties (continued)**

Property	Value
Preserve-partitioning flag in output data set	Propagated
Composite operator	No

Syntax and Options

The syntax for the **query_extract** operator in an `osh` command is shown below.

```
query_extract
    -query_field fieldname
        [-variable_bound fieldname=fieldvalue]
        [-field_name fieldname=fieldvalue]
    [-associative_delimiters list-of-delimiters]
    [-column_delimiters list-of-delimiters]
    [-hostname_convert
        upper | lower]
    [-keep_query_string]
    [-no_reformat]
    [-separated_query fieldname]
    [-uri_string_input fieldname]
```

Table 12 **query_extract Operator Options**

Option	Use
-associative_delimiters	`-associative_delimiters` *list-of-delimiters* Specifies a list of possible delimiting characters to signify associations. For example: *variable* = *value*. The default is no association delimiter.
-column_delimiters	`-column_delimiters` *list-of-delimiters* Specifies a list of delimiting characters to separate query fields; the default is the end of the query string.

-hostname_convert	`-hostname_convert`
	`   upper | lower`
	
	Specifies the conversion of the remaining URI string to a uniform case; the default is to leave the case unchanged.

Table 12 **query_extract Operator Options (continued)**

Option	Use
-keep_query_string	`-keep_query_string` Specifies that the query string from URI is not be be removed. By default, the query string is removed.
-no_reformat	Do not undo the URI encoding of characters; the default is to automatically undo URI encoding in all output query fields.

-query_field	`-query_field` *fieldname*
	`[-variable_bound` *fieldname=fieldvalue*`]`
	`[-field_name` *fieldname=fieldvalue*`]`
	
	Specifies a field within the query to be output. It is filled by the first non-variable-bound input in the query string unless the **-variable_bound** flag is set.
	
	The **-variable_bound** optional suboption specifies that this field is bound to a variable. It has the format *fieldname=fieldvalue*. *Fieldname* is the same as that specified by the **-query_field** option; and by default, *fieldname* is assumed to be the same as the specified output field. All non-variable-bound fields must be declared in the order that they occur, but variable-bound fields can be in any order. Any fields that are not present in the query string are set to NULL on output.
	
	The **-field_name** optional suboption specifies that the query string input should come from a different field than the output field name, rather than the default field specified by the **-query_field** option. This is only meaningful if the **-variable_bound** suboption has also been specified.

-separated_query	`-separated_query` *fieldname* Specifies that the query submitted is stored in a field other than the URI field. This cannot be set to be the same as the **`uri_string_input`** field.
-uri_string_input	`-uri_string_input` *fieldname* Specifies the name of the URI input field; the default is **`uri_string`**.

Examples

For this osh syntax:

```
query_extract -column_delimiters '&'
          -associative_delimiters '='
          -query_field name1 -variable_bound
          -query_field name2 -variable_bound
          -query_field name3 -variable_bound
```

When the query input string is:

```
name1=value1&name2=value2&name3=value3
```

The output is:

```
name1:undo_uri_encode(value1);
name2:undo_uri_encode(value2);
name3:undo_uri_encode(value3)
```

When a **query_field** name conflicts with a field name in the schema, use the **-field_name** suboption. For example:

When the query input string is:

```
uri_string=value0&field1=value1...
```

use osh syntax similar to this:

```
query_extract
   -column_delimiters '&'
   -associative_delimiters '='
   -query_field uri_string_from_query
   -variable_bound -field_name uri_string
   -query_field field1 -variable_bound ...
```

The perlbuildop Operator

The **perbuildop** operator reads records from one or multiple inputs and writes them to one or multiple outputs.

This operator is similar to a standard Orchestrate **buildop** operator, but Perl is used to specify the operator logic rather than C/C++. This allows you to take advantage of Perl's robust text processing capabilities while also having the option of using an interpreted language to specify operator logic.

You implement this operator by:

- Explicitly defining the record-processing loop within your Perl code
- Specifying the input and output fields that are accessed by name within the operator
- Indicating whether output fields can be nullable
- Declaring the transfers of unreferenced fields from input to output datasets
- Indicating whether the Perl warning mode switch should be turned on

perlbuildop Records

There is a set of pre-defined Perl subroutines that allow your Perl code to read and write Orchestrate records and perform record transfers. Records are read and written using these subroutines:

```
getInputRecord(dataset_number)
putOutputRecord(dataset_number)
```

The `getInputRecord()` subroutine returns a non-zero code as long as there are more records to read from the specified dataset. When the dataset has been read to completion, `getInputRecord()` returns zero.

Record transfers of schema variables are performed using the following subroutine:

```
transfer(transfer_index)
```

The *transfer_index* argument is the index of the transfer declaration based on the order of your transfer options.

Input and output field values are accessed through arrays of hashes of scalars. There is one array to access input-field values and one to access output-field values. Each element of an array corresponds to a hash of the fields for a specific dataset. Each hash key value corresponds to a scalar that contains the value of the field named in the key. For example, the scalar containing the value of field *foo* on input dataset `0` could be accessed using:

```
$inputRecord[0]{'foo'}
```

And the value of field *bar* on output dataset 1 could be set by using the following scalar:

```
$outputRecord[1]{'bar'}
```

All input and output field values are accessed as scalars within Perl, regardless of their Orchestrate data type. This is adequate for data types such as strings and integers, but could pose a problem for composite data types such as dates and timestamps. Therefore, **perlbuildop** converts all field values to their string representation using the default modify adapter conversion.

If an input field is nullable and contains a null value, the corresponding scalar is undefined. If an output field is nullable, its output value can be set to null by undefining the corresponding Perl scalar. For example, the field *foo* on input dataset 0 contains a null value if this statement returns true:

```
!defined($inputRecord[0]{'foo'})
```

The nullable field *bar* on output dataset 1 can have its value set to null with this statement:

```
undef $outputRecord[1]{'bar'};
```

Defining perlbuildop Data Types

Orchestrate determines the data type of each output field you explicitly define in the output-field operator options as follows:

1. It loops over the transfer declarations and finds the first declaration whose output dataset matches the working output dataset and whose input dataset contains the working field on the schema.
2. If this field is found, it sets the data type of the output field to the data type of the input field. If this field is not found or no transfers are declared, it sets the data type of the output field to **string**.

This also applies to output field nullability if the **nullable** suboption has not been set for the field in the operator options. The **nullable** suboption overrides this derivation.

Defining perlbuildop Options

You can define the options for **perbuildop** and the Perl code segment in two ways. The first method is to specify each option as a separate Orchestrate option. The second method is to specify a single option that points to a specification file containing both the Perl code and all the options embedded in Perl comments.

You define these options by inserting comments into the specification file. For example:

`## INPUT_FIELDS` *dataset_number fieldname1 [fieldname2]* ...

Note Option comments should be on a single line. For readability, some examples below are displayed on two lines.

You must define one of these options for each input dataset. For example, to declare the fields *fielda* and *fieldb* from input dataset 0 for use in a **perlbuildop** operator, insert this comment line:

```
## INPUT_FIELDS 0 fielda fieldb
```

```
## OUTPUT_FIELDS 'dataset number' 'fieldname1'[:NULLABLE]
     ['fieldname2'][:NULLABLE] ...
```

The `NULLABLE` keyword is appended to a field if that field should be nullable on output. This overrides the field nullability derived from a transfer declaration, as described above.

To declare the fields *fielda* and *fieldb* from output dataset `1`, with *fielda* being nullable, the comment line in **perlbuildop** is:

```
## OUTPUT_FIELDS fielda:NULLABLE fieldb
```

```
## DECLARE_TRANSFER 'input dataset number' 'output dataset number'
```

One of these options should be defined for each transfer of unused fields from an input dataset to an output dataset. You execute these transfers within the Perl code by using the `transfer()` subroutine based on the order the transfers are declared. Use the index 0 to execute the first `DECLARE_TRANSFER`, use index 1 for the second, and so on. For example, to declare a transfer of unused fields from input dataset `1` to output dataset `2`, the comment line is:

```
## DECLARE_TRANSFER 1 2
```

To invoke the warning mode in the Perl interpreter use this comment line:

```
## WARNING MODE
```

perlbuildop Example

Here is an example specification file:

```
## INPUT_FIELDS 0 in_field01 in_field02
## OUTPUT_FIELDS 0 out_field01 out_field02:NULLABLE
## OUTPUT_FIELDS 1 out_field11
## DECLARE_TRANSFER 0 0
## DECLARE_TRANSFER 0 1

while (getInputRecord(0)) {

// 1
$outputRecord[0]{'out_field01'} =
   $inputRecord[0]{'in_field01'};

// 2
if (!defined($inputRecord[0]{'in_field02'})) {
   undef $outputRecord[0]{'out_field02'};
}

// 3
$outputRecord[1]{'out_field11'} =
   $inputRecord[0]{'in_field01'};

// 4
transfer(0);

// 5
putOutputRecord(0);

// 6
transfer(1);

// 7
putOutputRecord(1);
}
```

In this file, the operator is accessing the fields *in_field01* and *in_field02* from input dataset `0`, the fields *out_field01* and *out_field02* from output dataset `0`, where *out_field02* is being declared as nullable, and the field *out_field11* from output dataset `1`.

Transfers have been declared from input dataset `0` to output dataset `0` and from input dataset `0` to output dataset `1`. Since no output fields match the input fields of any transfers that apply to their datasets, the fields have **string** data types within Orchestrate.

The code loops over the input records of input dataset `0` and executes the following seven statements for each input record:

1. Set the value of field *out_field01* in output dataset `0` to the value of field *in_field01* in input dataset `0`.
2. If the value of field *in_field02* in input dataset `0` is null, set the value of field *out_field02* in output dataset `0` to null.
3. Set the value of field *out_field11* in output dataset `1` to the value of field *in_field01* in input dataset `0`.
4. Transfer the unused input fields from input dataset `0` to output dataset `0`.
5. Write a record to output dataset `0` which contains the transferred field values as well as the field values set in the corresponding *outputRecord* array element.
6. Transfer the unused input fields from input dataset `0` to output dataset `1`.
7. Write a record to output dataset `1` which contains the transferred field values as well as the field values set in the corresponding *outputRecord* array element.

perlbuildop Input/Output Interfaces

You define the input and output interfaces, but are restricted to top-level fields for fields accessed by name within the operator code. Fields that are passed through this operator using transfer declarations are not subject to this restriction.

Syntax and Options

The syntax for the **perbuildop** operator in an osh command is shown below.

```
perlbuildop
    -spec_file filename (mutually exclusive with all other perbuildop options)
    |
    -code_segment pathname
    [-declare_transfer
        -transfer_from input_dataset_number
        -transfer_to output_dataset_number
     ...]
    -input_fields dataset_number
        -input_field fieldname [-input_field fieldname ...]
        [-partkey fieldname
            [-ci | cs]
            [param property=value_pairs]
         ...]
        [-sortkey fieldname
            [-ci | cs]
            [asc | desc]
            [nulls
                first | last]
            [param property=value_pairs]
         ...]
    [-input_fields dataset_number (suboptions) ...]
    -output_fields dataset_number
        -output_field fieldname [-output_field fieldname ...]
            [nullable]
        [-partkey fieldname
            [-ci | cs]
            [param property=value_pairs]
         ...]
        [-sortkey fieldname
            [-ci | cs]
            [asc | desc]
            [nulls
                first | last]
            [param property=value_pairs]
         ...]
    [-partsame]
    [-warning_mode]
```

Table 13 **perlbuildop Operator Options**

Option	Use
-code_segment	`-code_segment` *pathname* The path to the Perl code that is executed in the interpreter. This option is mutually exclusive with **-spec_file**, but required if **-spec_file** is not defined.

-declare_transfer	`-declare_transfer`
	    `-transfer_from` *input_dataset_number*
	    `-transfer_to` *output_dataset_number*
	
	Declare a transfer of unused fields from an input dataset to an output dataset. There can be any number of occurrences of this option. This option is mutually exclusive with **-spec_file**.
	
	The required suboptions **-transfer_from** and **-transfer_to** specify the input data set and output data set for the transfer.

-input_fields	`-input_fields` *dataset_number*
	     `-input_field` *fieldname*
	     `-partkey` *fieldname*
	         `-ci | -cs`
	         `-param` *property=value_pairs*
	     `-sortkey` *fieldname*
	         `-ci | -cs`
	         `-asc | desc`
	         `-nulls`
	             `first | last`
	         `-param` *property=value_pairs*
	
	The dataset number for an input field list. One of these options must be set for each input dataset attached to the operator. This option is mutually exclusive with **-spec_file**.
	
	The suboption **-input_field** specifies the name of an input field to be accessed by the operator. One or more field names must be declared.
	
	continued on the following page

Table 13 **perlbuildop Operator Options (continued)**

Option	Use
-input_fields *continued*	The suboption **-partkey** specifies the name of an input field to be used as a partitioning key. There can be any number of occurrences of this suboption. Using the **-partkey** suboptions, optionally specify **-ci** or **-cs** for case-insensitive or case-sensitive comparison, and optionally use the **-param** option to specify extra parameters for partitioning keys. Case-sensitive comparison is the default. The suboption **-sortkey** specifies the name of an input field to be used as a sort key. There can be any number of occurrences of this suboption. The **-sortkey** suboptions are as follows. Use **-ci** or **cs** (default) to optionally indicate case-insensitive or case-sensitive comparison. Use **-asc** or **-desc** to optionally specify ascending (default) or descending sort order. The **-nulls** suboption optionally specifies whether null values should be sorted first (default) or last. Using the **-param** suboption, you optionally specify *property*=*value* pair(s) as extra parameters for the sort key.

Table 13 **perlbuildop Operator Options (continued)**

Option	Use

-output_fields	`-output_fields` *dataset_number*
	    `-output_field` *fieldname*
	        `-nullable`
	    `-partkey` *fieldname*
	        `-ci | -cs`
	        `-param` *property=value*
	    `-sortkey` *fieldname*
	        `-ci | -cs`
	        `-asc | desc`
	        `-nulls first | last`
	        `-param` *property=value*
	
	The dataset number for an output field list. One of these options must be set for each output dataset attached to the operator. This option is mutually exclusive with **-spec_file**.
	
	The suboption **-output_field** specifies the name of an output field to be accessed by the operator. One or more field names must be declared. Using the **-nullable** suboption, optionally specify whether an output field is nullable.
	
	The suboption **-partkey** specifies the name of an output field to be used as a partitioning key. There can be any number of occurrences of this suboption. Using the **-partkey** suboptions, optionally specify **-ci** or **-cs** for case-insensitive or case-sensitive comparison, and optionally use the **-param** option to specify extra parameters for partitioning keys. Case-sensitive comparison is the default.
	
	The suboption **-sortkey** specifies the name of an output field to be used as a sort key. There can be any number of occurrences of this suboption.
	
	continued on the following page

Table 13 **perlbuildop Operator Options (continued)**

Option	Use
-output_fields *continued*	The **-sortkey** suboptions are as follows. Use **-ci** or **cs** (default) to optionally indicate case-insensitive or case-sensitive comparison. Use **-asc** or **-desc** to optionally specify ascending (default) or descending sort order. The **-nulls** suboption optionally specifies whether null values should be sorted first (default) or last. Using the -**param**suboption, you optionally specify *property=value* pair(s) as extra parameters for the sort key.
-partsame	`-partsame` Optionally specify that Orchestrate partition all input data sets using the partitioning method specified for dataset 0.
-spec_file	`-spec_file` *filename* Specifies the name of a specification file which contains all the information to run a **perlbuildop**. This option is mutually exclusive with all the other options. The format of this file is described in "Defining perlbuildop Options" on page 2-22.
-warning_mode	`-warning_mode` This is an optional option that executes the Perl interpreter in warning mode. It is mutually exclusive with **-spec_file**.

The sessionize Operator

Description

The **sessionize** operator reads records from one input and writes them to one output. You can also specify two optional dataset outputs: a summary dataset and a dwell-time dataset.

This operator creates a logical grouping of records for later analysis. The input is a sequence of records in timestamp order, containing a user ID and optionally other fields. The operator creates a session ID for each user ID and adds the field to the output.

The session ID changes when one of these conditions is met:

- The user ID has not been seen in a previous record.
- The interval between the timestamp on the current record and the most recent record with the same user ID is more than your specified time.
- One or more of the other fields match a configured boundary expression. A boundary condition is specified in the same manner as a regular expression, similar to a `WHERE` clause in the **filter** operator. See the *Orchestrate 6.1 Operators Reference* for information on the **filter** operator and its `WHERE` clause.

When the **-summary** option is specified, a summary dataset is generated as an additional output. As each session terminates, a record is written to this summary dataset.

Each record in the summary dataset includes:

- the user ID
- the session ID, which is also the session start time
- the duration of the session in seconds
- the reason for termination, which can be `a timeout or` the index of the matching boundary condition
- the number of records in the session

When the **-dwell** option is specified, a dwell dataset is generated as an additional output. For each record of a session after the first, a record is written to this dwell dataset.

Each record in the dwell dataset includes:

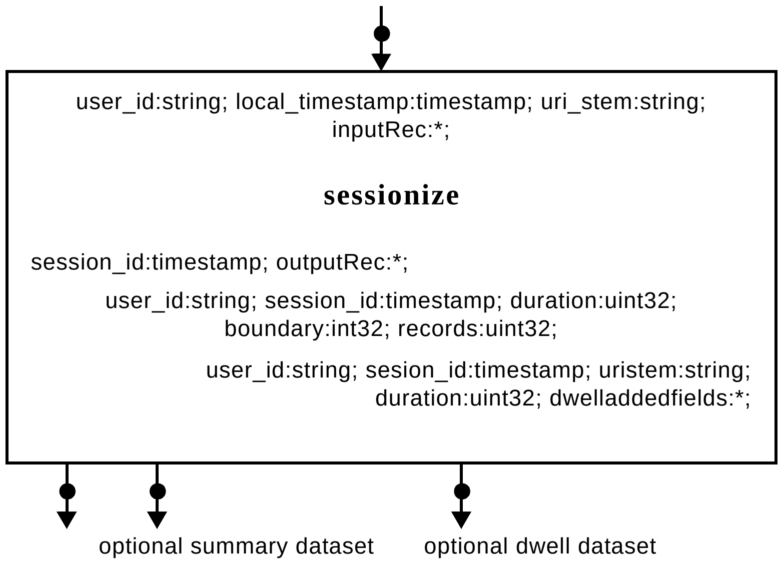
- the user ID
- the session ID, which is also the session start time
- the URI of the previous record

- the duration of the time between URI requests in seconds
- additional fields you specify obtained from the input

The last record of the session has a NULL dwell time because there is no subsequent record from which to determine dwell time. When records in the input represent pages, the dwell output is the load time added to the dwell time per page.

The dwell data set can optionally contain extra fields from the clickstream using the **-add_field** suboption of **-dwell**.

Data Flow Diagram


user_id:string; local_timestamp:timestamp; uri_stem:string; inputRec:*;
sessionize
session_id:timestamp; outputRec:*;
user_id:string; session_id:timestamp; duration:uint32; boundary:int32; records:uint32;
user_id:string; sesion_id:timestamp; uristem:string; duration:uint32; dwelladdedfields:*;
optional summary dataset
optional dwell dataset

sessionize Input/Output Interfaces

The input interface uses **eAny** partitioning. Its schema is:

```
record ( user_id:string; local_timestamp:timestamp; uri_stem:string;
         inRec:*; )
```

The output interface does not change the **preserve-partitioning** flag. Its schema is:

```
record ( session_id:timestamp; outRec:*; )
```

The summary output does not have **preserve-partitioning** set. Its schema is:

```
record ( user_id:string; session_id:timestamp; duration:uint32;
         boundary:int32; records:uint32; )
```

The boundary field contains the index of the matching boundary condition, or -1 to indicate that the session has timed out, or -2 to indicate that the session was terminated by end-of-data.

Preserve-partioning is not set for dwell output. Its schema is:

```
record ( user_id:string; session_id:timestamp; uri_stem:string;
         duration:uint32; dwellAddedFields:*; )
```

The **dwellAddedFields** are fields that are specified to be passed through to the dwell output. This is done so that the dwell dataset does not need to be merged with another output in order to retrieve fields.

Properties

Table 14 **session Operator Properties**

Property	Value
Number of input data sets	1
Number of output data sets	1 by default; plus 1 or 2 optional datasets
Input interface schema	record (user_id:string; local_timestamp:timestamp; uri_stem:string; inRec:*;)
Output interface schema	record (session_id:timestamp; outRec:*;) (optional) record (user_id:string; session_id:timestamp; duration:uint32; boundary:int32; records:uint32;) (optional) record (user_id:string; session_id:timestamp; uri_stem:string; duration:uint32; dwellAddedFields:*;)
Transfer behavior	The input record is transferred to output 0. If the input record has a field with the same name as the session ID field, that input field is dropped from the transfer.
Execution mode	Parallel (default) or sequential
Partitioning method	hash, range, entire
Collection method	Any (sequential mode)

Table 14 **session Operator Properties (continued)**

Property	Value
Preserve-partitioning flag in output data set	No
Composite operator	No

Syntax and Opions

The syntax for the **sessionize** operator in an `osh` command is show below.

```
sessionize
    [-boundary expression1 expression2 ...]
    [-buckets integer]
    [-dwell [-add-field fieldname1 -add-field fieldname2 ...]]
    [-doStats]
    [-sessionID field]
    [-summary]
    [-timeout minutes]
    [-timestamp field]
    [-uri field]
    [-userID field]
```

Table 15 **sessionize Operator Options**

Option	Use
-boundary	`-boundary` *expression1 expression2 ...* An expression that identifies a session boundary. The format of the expression is the same as that for the **filter** operator. Multiple boundary conditions may be given.
-buckets	`-buckets` *integer* The number of buckets that should be used in the hash table. Using more buckets makes processing faster, but uses more memory. The optimal value is one bucket per eight simultaneous sessions. For example, a specification for a million simultaneous sessions might be: `-buckets 125000` The default is 1000.

Table 15 **sessionize Operator Options (continued)**

Option	Use
-dwell	`-dwell` `   -add-field` *fieldname* Specifies that a dwell output dataset should be generated. The optional suboption **-dwell** that is used to transfer a field from the input into the dwell dataset. It takes a fieldname as input and may be referenced more than one time to pass through more fields if needed.
-doStats	`-doStats` Specify that performance statistics be written at the end of a run.
-sessionID	`-sessionID` *field* The name of the output field to used as the session ID; the default is **session_id** **.**
-summary	`-summary` Specifies that a summary output dataset should be generated.
-timeout	`-timeout` *minutes* The inactivity timeout interval in minutes; the default is 30.
-timestamp	`-timestamp` *field* The name of the input field to use as the timestamp; the default is **local_timestamp** **.**
-uri	`-uri` *field* The name of the input field to use as the URI string; the default is **uri_stem** **.**
-userId	`-userId` *field* Name of the input field to use as the user ID; the default is **user_id**.

Index