dataZoa.com API Documentation
Published by Leading Market Technologies, Inc.
Version 2.0.2
Publication Date: May 10, 2018

Table of Contents

1 Introduction

1.1 About dataZoa

dataZoa.com is a website dedicated to the storage, manipulation, management and display of time-series data. dataZoa users maintain individual accounts that hold a mixture of private data, data from public sources, and data shared among other dataZoa users.

dataZoa supports a variety of methods to store and retrieve user data, including Drag-and-Drop among sites and programs, automated and scheduled retrieval, manual input and editing, and programmatic interfacing through an API (Application Programming Interface).

1.2 Scope of this document

This document is centered on the dataZoa API capabilities and methods, but includes some discussion of dataZoa formats and conventions. These discussions do not document all of dataZoa, but are intended to provide the understanding required to build meaningful applications.

This document assumes familiarity with commonly used Web programming concepts, in particular http POSTing and JSON encapsulation.

2 Terms and Concepts

2.1 API

The API provided for dataZoa is at heart a pure *data-exchange interface*. That is, this API does not require any libraries or other client-side code. The conventions and protocols used for the data interchange are common to typical Web applications; http(s) and JSON.

The target URL for all API calls is:

datazoa.com/api/apimain.asp

Thus, a typical GET type transaction might look like from an ordinary browser might look like:

http://www.datazoa.com/api/apimain.asp/?apikey=nqgjhkiqeolhlhdpgbaigjppnapghaplppgnkbql&func=version

Important: API calls that *write* data to dataZoa (e.g. SeriesCreate) must be issued *serially* and must not be issued in parallel. This is required to avoid race conditions and resource conflicts and to guarantee sequential allocation of serieskeys and other unique resource handles.

2.2 API Key

In order to use any functionality from the dataZoa API, you must supply your login credentials in the form of an *API Key*. This is a hashed alphanumeric key that is a unique function of your userid, password, and certain other information. You can obtain your API Key by logging into your dataZoa account, visiting the Profile page, and opening the "api key" link. Copy and paste the information at that link for use in your API calls.

Please remember, you must obtain an updated API Key whenever you change your password.

2.3 Metadata

Metadata is the dataZoa term for any attributes of data that are not dates or values of the actual time series. Some metadata, such as *frequency,* is functional; affecting the way data is stored and processed. Other metadata, such as *units*, may be purely descriptive.

2.4 Data Types

Because the dataZoa API employs JSON text transfers, strict binary data typing is not a concern, and the intrinsic text-to-binary conversion functions of most common languages work seamlessly.

The specific formats for *parameters sent* to dataZoa are shown in the call-specific documentation.

For *results returned* by dataZoa, users should anticipate that numeric values returned by the API can take on a variety of natural human-readable forms, such as: "*1", "1.0"," -0.1", "-.1" or "1.01E12."*

Similarly, date or date/time strings *returned* may variously include dates only or date/times, and will be shown in natural forms such as *"4/26/2014 or "4/26/2014 12:43:22 AM,"* with MM/DD/YYYY month/day ordering presumed.

2.5 Series and Serieskey

A dataZoa *series* is a single time series, comprising a Header, DataSection, and Footer. These three main sections are separated by one or more blank lines.

Series are identified by a unique key that is initially generated by dataZoa.com but can be used by the application builder to later replace a series. The key is built from the account *username* and a generated serial number. A typical *serieskey* might look like "UserAlice/00000321".

2.5.1 Header

Example:

Series title: M1 Money Stock
Series key: Citizens/00000556
Data-set title: Macro Data
Units: Dollars

Unit multiplier: Billions
Frequency: Weekly
Start date: 1/6/1975
End date: 11/28/2011
Release date: 12/15/2011
Publisher: Leading Market Technologies
Source: Board of Governors of the Federal Reserve System

2.5.2 Header Functional Keywords

2.5.2.1 "Series title:"

This is used for display purposes throughout dataZoa.

2.5.2.2 "Series key:"

See discussion above. Although it is part of the header as stored, it should always be supplied by dataZoa, never by the end user.

2.5.2.3 "Frequency"

If not supplied, dataZoa will infer the frequency of the data. If supplied, it must match one of the frequencies supported by dataZoa. As of this writing, these possible values are: Daily, Weekly, Monthly, Quarterly, Semiannual and Annual.

2.5.3 Descriptive Keywords

All keywords shown in the header example above that are not explicitly noted as Functional Keywords will be parsed and stored as meta-data, but do not have particular functional roles as of this writing.

2.5.4 DataSection

The DataSection begins following the first blank line after any Header section. If there is no Header, the DataSection can begin immediately.

A DataSection should have at least three lines of date value pairs.

A DataSection can have more than one value column for each row. When multiple values columns are present, one unique series is created for each values column, applying the date column to each. All rows should have the same number of tokens. A pair of double quotes can be used as a placeholder for any missing tokens.

The first row can be a set of column labels. If present, the column labels are incorporated into the series titles. Multi-word column labels should be enclosed in double quotes.

Example:

DATE	"My Values"
1/5/1981	407.4
1/12/1981	409.4
1/19/1981	413.2
1/26/1981	413.8
2/2/1981	410.8
2/9/1981	413.6

…

2.5.5 Footer

The Footer begins following the first blank line after the DataSection, and can be free-form, except for certain functional keywords that will have significance if they start a line.

Example:

This is footer.
It is a great place for footnotes!

UPDATE_URL: http://research.stlouisfed.org/fred2/series/M1
UPDATE_FREQ: 1440
FAVICON_LOC: http://research.stlouisfed.org/favicon.ico
FAVICON_DEST: http://research.stlouisfed.org/fred2/series/M1?cid=121

2.5.6 Footer Functional Keywords

2.5.6.1 "UPDATE_URL:"

dataZoa will refresh data from this location.

2.5.6.2 "UPDATE_FREQ:"

dataZoa will refresh data this often – value in minutes; suggested, not guaranteed. As of this writing, the standard update cycle is once per day in the early morning US Eastern time zone.

2.5.6.3 "FAVICON_LOC:"

Location of a standard favicon to display with this data.

2.5.6.4 "FAVICON_DEST"

Location of click-through on the favicon.

2.6 Calculations (calculated series)

A *calculation* (also referred to as a *calculated series*) works like any other data series in dataZoa, but rather than holding raw numbers, its contents are mathematically derived from other series. Calculated series have all the normal attributes of a data series, plus a few special ones. The dataZoa API offers a set of functions that specifically address calculated series.

2.6.1 Terminology

Any series involved in a calculation is referred to as a ***node***. Note that this may be a calculated series with a formula or just an ordinary series that some calculation uses as an input.

A ***child node*** depends on other nodes. It has a formula and may be referred to as a ***calculated node***.

A ***parent node*** is used as an input by one or more child nodes. It may or may not be a child node in its own right.

A ***headwaters node*** is a parent that does not depend on any other node. It is an ordinary series, subject to ordinary manual or automatic updating, that happens to be used in a calculation somewhere.

A child node that no other node depends on is a ***terminal node.***

2.6.2 Usage Note

The interrelationships among nodes can be very complex. This is particularly important when a set of headwaters nodes all change at once. Rather than evaluating all downstream changes as each headwaters node is updated, it is necessary to group the "simultaneous" changes in a batch and then gather, sort and evaluate the dependent nodes in one coherent operation.

In the case of typical automatic or manual data updates within dataZoa, any batch processing of calculation updates happens automatically.

In the case where headwaters nodes are updated via the dataZoa API, however, there is no implicit "batching" of the updates, and therefore calculation updates are deferred and not addressed until a period "sweep" of calculated nodes is made. This may leave child nodes in need of recalculation for up to several hours.

The dataZoa API provides functions (CalcEvalDescendants, CalcEvalAccount) to initiate batches of downstream calculations as appropriate after batch updates of headwaters nodes are completed.

2.6.3 Version Note

The dataZoa calculation engine, *dataZephyr*, is available as a stand-alone desktop application. It was in use as a cooperating component of dataZoa before it was directly incorporated into the dataZoa user interface in December, 2014. When dataZephyr was merged into dataZoa, its desktop usage was deprecated, but any legacy

calculations remain supported. The dataZoa API distinguishes these legacy calculations as *version 1*, and all others as *version 2* or higher.

2.7 Grouplists

A dataZoa *grouplist* is an ordered collection of serieskeys that are used in the dataZoa user interface to organize sets of data.

In the dataZoa API, grouplists are referred to with a "glname" argument. Grouplist names are fairly free form, except that the vertical bar ("|") character is not allowed.

There is a special grouplist name, "All of Latest Upload," that is re-created whenever a series or group of series is acquired from a URL, either via the dataZoa UI drag-and-drop to target or through the API SeriesFromURL or SeriesCreate calls. This list can be queried and saved to a new unique name to keep series grouped as they are acquired.

2.8 Displays

A dataZoa *display* is a user-created table. chart. etc., as seen in the dataZoa account UI and as published around the world. In the dataZoa API, displays are referred to with a *displayhandle* argument. API functions that take displayhandle arguments typically also take an *ownerhandle* argument as well, in order to further disambiguate the displayhandle, particularly when working across related accounts.

2.8.1 Displayhandle

A *displayhandle* is used to identify a display in a particular dataZoa account. The displayhandle can be either its *displayid* or its *displayname*.

2.8.1.1 displayid

A *displayid* (also known as the display's *hash*) is unique within the account that owns the display, but not necessarily across accounts. It is found in the display's embed URL as seen in any of the dataZoa display editors. In the embed URL, the displayid is the ten randomized characters following the "th=" argument; e.g. "&th=3E826612BB"

2.8.1.2 displayname

A *displayname* is not necessarily unique within the account that owns the display, but can be convenient to use as a reference if properly managed. It is the name as seen in the dropdown lists for any of the dataZoa display editors (e.g. "My green pie chart").

2.8.2 Attributes

Displays can have dozens of descriptive attributes. For any given display, they can be discovered by inspection using the API function DisplayGetAttr.

In the dataZoa UI display editors, users can set certain display attributes (e.g. background color) directly. These are referred to as *primitive attributes*. Other attributes (e.g. the embed URL), are referred to as *derived attributes* because their values are not entered directly by a user, but established as a function of other values.

Important note: Derived attributes can be set via the API, but may become overridden by other actions and conditions.

2.9 Accounts, Objects and Ownership

2.9.1 Accounthandle

When a dataZoa account is created, it is assigned three attributes that are unique to the account and cannot be changed: *username*, *origemail*, and a *dzuuid*. In the dataZoa API, an *accounthandle* can be any of these; they all resolve to the same account.

2.9.1.1 username (or accountname)

The account *username* was supplied by the user when the account was created. It appears as the first part of every serieskey, and can always be seen in the dataZoa UI in the "Profile/Preferences" panel.

Note: The term *username* is interchangeable with the term *accountname*.

2.9.1.1 origemail

The origemail ("original email") for the account is the email address supplied when the account was created. It is the email account where the account creation was verified and accepted.

2.9.1.2 Note: emailtouse

dataZoa has the concept of an *emailtouse*, which is where account-related correspondence is sent. It can be changed after the account is created and is thus not formally acceptable as an *accounthandle*, even though it is initially set to be the same address as the unchangeable *origemail*.

2.9.1.3 Dzuuid

A *dzuuid* (short for "DataZoa Unique User ID") is a unique serial number for a dataZoa account. It can be seen in the "account info" section of the "Stats" tab of the "Profile/Preferences" panel in the dataZoa UI.

2.9.2 Ownerhandle

The term *ownerhandle* refers to the *accounthandle* of the owner of an object.

2.9.3 Related Accounts

In dataZoa, the highest level of subscription, commercial/publishing accounts, support the notion of a collaborative group of *related accounts*. Related account groups have a special account which is the related accounts *leader*, and some number of associated *members*.

Certain objects within dataZoa can be copied and/or transferred among related accounts by the leader using the API. Grouplists and Displays can be manipulated without restriction. In the case of Series, copying is unrestricted, but transfer of ownership may be precluded by certain usage conditions. The SeriesXref API function is useful to see where and how series are used.

3 Functions

All calls are made to the API data interface URL, generally in the form:

http://www.datazoa.com/api/apimain.asp?apikey=xxxxxxxx&func=FUNC&ARGn...

where xxxxxxxx, FUNC and ARGs are specified as warranted.

Both the http and https protocols are supported.

The API will accommodate both GETs and POSTs as is convenient for the developer. GET arguments have precedence over POSTed arguments when there is conflict, and GETs are subject to the typical "few thousand character" limitations typically encountered in such interactions. dataZoa make no particular guarantees about data capacities of GET operations.

Responses are typically four JSON encapsulated fields; *func* (what you asked for), *code* (integer response code), *message* (for humans), and *payload* (to digest). Please see the Encodings discussion for further details.

Example:

https://www.datazoa.com/api/apimain.asp?apikey=e94140d81ba312888a810a8488165b94&func=version

Response when proper apikey is supplied:

```
{"func":"version","code":"0","message":"","payload":"2.0.1"}
```

Otherwise:

```
{"func":"version","code":"802","message":"keylogin failed","payload":""}
```

3.1 Utility calls

3.1.1 Func=Version

Purpose:
Determine current version of the dataZoa API

Arguments:
None.

Payload:
Version string.

Example:

```
Call: func=version

Response: {"func":"version","code":"0","message":"","payload":"1.1.1"}
```

3.1.2 Func=Runstate

Purpose:
Determine current runstate of the dataZoa API

Arguments:
None.

Payload:
Current runstate string.

Example:

```
Call: func=runstate

Response: {"func":"version","code":"0","message":"","payload":"NORMAL"}
```

Note: The other commonplace response is “READONLY” which is returned when the dataZoa site is in READONLY mode for maintenance.

3.2 Series Manipulation

These calls are used to manipulate series at dataZoa. Note that some calls take a single *serieskey* argument, while others accept a comma-delimited *serieskeylist*.

3.2.1 Func=SeriesExists

Purpose:

Determine whether a given series exists.

Arguments:

serieskeylist=*serieskey [,serieskey]…*

Payload:

Comma delimited list of serieskey:true or *serieskey:false*

Example:

```
Call: func=seriesexists&serieskey=UserAlice/00000213,UserAlice/00000214

Response: {"func":"seriesexists","code":"0","message":"","payload":"
UserAlice/00000213:true, UserAlice/00000214:true"}
```

3.2.2 Func=SeriesGet

Purpose:

Return a single series exactly as stored at dataZoa. Analogous to the dataZoa Drag-and-Drop drag source functionality.

Arguments:

serieskey=*serieskey*

startdate=*startdate* (optional; if present, return data on or after this date)

enddate=*enddate* (optional; if present, return data on or before this date)

maxtoload=*maxtoload* (optional; if present, along with at least one start or end date, return no more than this many observations)

payloadasjson= [**true** | **false**] (optional; default = false; if true payload will be JSON encoded)

Payload:

Text of series as stored by dataZoa.

Notes:

Date formats should be of the form MM/DD/YYYY.

If startdate, enddate and maxtoload are all supplied, maxtoload is calculated from enddate.

3.2.3 Func=SeriesCreate

Purpose:

Create one or more series from properly formatted text. Analogous to the dataZoa “Add New Series” functionality.

Arguments:

raw= [properly formatted plain text data]
allowreplacement= [**true** | **false**] (optional; default = false)

Payload:
Comma delimited list of newly created serieskey(s)

Notes:
If "allowreplacement=true" is present, the raw data series are expected to have headers that include an accurate "SERIES_KEY:" entry so that when the series are created they will entirely replace those with corresponding keys. If a "SERIES_KEY:" is not present in the series header, a new key will be created. This is the equivalent of checking the "Allow Replace" checkbox in the dataZoa create/update data entry area,

3.2.4 Func=SeriesReplace

Purpose:
Replace a single series entirely at dataZoa. Analogous to the dataZoa edit functionality.

Arguments:
serieskey=*serieskey*
strict= [**true** | **false**] (optional; default = true)
raw= [properly formatted plain text data]

Payload:
None.

Notes:
If "strict=false" is present, the data format requirements for SeriesReplace are the same as those for SeriesCreate, as described earlier in this document.

If "strict=false" is not present, the requirements for the data format for SeriesReplace are more restrictive, in that the data format must be nearly identical to the format in which dataZoa returns data.

3.2.5 Func=SeriesFromURL

Purpose:
Create one or more series by acquisition from a 3rd party URL. Analogous to the dataZoa URL drag-and-drop "target" functionality.

Arguments:
url=*any appropriate URL*

Payload:
Comma delimited list of newly created serieskey(s)

Example:

```
Call: func=seriesfromurl&url=http://research.stlouisfed.org/fred2/series/STLFSI?cid=121
```

```
Response:
{"func":"seriesfromurl","code":"0","message":"OK","payload":"UserAlice/00000187"}
```

3.2.6 Func=SeriesList

Purpose:

Enumerate all serieskeys in the account.

Arguments:

serieskeylist=*serieskey[,serieskey]…* | *glname* (optional; defaults = empty string)
pagesize=integer (optional; default = 8000, min = 10, max = 8000)
pagenum=integer (optional; default = 1)
titlefilt=string (optional; default = empty string)
fields= [*field* [,*field*…]] (optional (see table below); default = none other than serieskey)

Payload:

Comma delimited list of serieskeys

If additional fields are specified, payload is a comma delimited list of serieskey|fieldvalue|fieldvalue…

Example:

```
Call: func=serieslist&titlefilt=civilian&fields=date_modified,is_private

Response: {"func":"serieslist","code":"0","message":"OK","payload":"
UserAlice/00000190|4/25/2013 11:37:57 AM|true,UserAlice/00000130|4/25/2013 11:38:43
AM|true"}
```

Notes:

The "serieskeylist" argument controls the pool of which series to gather information for. This can be a single serieskey, a comma delimited serieskey list, a grouplist name or left unspecified implying the pool is all series in the account.

The “pagesize” argument controls the largest number of serieskeys that will be listed in one call. Setting a smaller pagesize may improve performance in some cases, but typically the default is fine.

The “pagenum” argument is used to specify which set of serieskeys is returned, in groups of pagesize. If you have more than the maximum pagesize number of series in your account, you will need to call SeriesList in a loop, incrementing pagenum, until no more keys are returned.

The “titlefilt” argument is used to limit the series considered to only those with titles that contain the value given.

The “fields” argument can contain any of:

Field name	Comments
sorting_date	As used in the main dataZoa list display
date_added	
date_modified	Last manual or automatic update

is_private	true or false
is_findable	true or false
start_date	
end_date	
Title	
updateurls_url	URL from which series is updated
Frequency	
Adjustment	
Unit	
Multiplier	
Publisher	
publisher_url	
n_viewers	Number of sharees + followers
is_borrowed	Is being "followed" by this account
n_borrowers	Number of "followers"

IMPORTANT: Because some returned fields could contain the comma or vertical bar delimiters as part of their text, all fields have substitutions made as needed, which the caller must reverse upon receipt. The substitutions are:

"_comma_" for ","
"_vertbar_" for "|"

3.2.7 Func=SeriesDelete

Purpose:

Delete series by key.

Arguments:

serieskeylist=*serieskey [,serieskey]…*

Payload:

Comma delimited list of *serieskey* for any series that were deleted.

3.2.8 Func=SeriesIsPrivate

Purpose:

To mark series as public or private, or to determine how currently marked

Arguments:

serieskeylist=*serieskey [,serieskey]…*
request= [**get** | **settrue** | **setfalse**]

Payload:

Comma delimited list of *serieskey:true* or *serieskey:false.* Note that any series not found by key will be reported as false.

3.2.9 Func=SeriesIsFindable

Purpose:

To mark series as findable or hidden, or to determine how currently marked

Arguments:

serieskeylist=*serieskey [,serieskey]…*
request= [**get** | **settrue** | **setfalse**]

Payload:

Comma delimited list of *serieskey:true* or *serieskey:false.* Note that any series not found by key will be reported as false.

3.2.10 Func=SeriesXREF

Purpose:

Cross-reference of where a particular dataZoa series is in use.

Arguments:

serieskeylist=*serieskey [,serieskey]…*
scope= [**self** | **all_related**]

- **self** – limit testing to series owner's account
- **all_related** – test for usages across all Related Accounts of the series owner

usages=*usage [,usage]…* (optional; default = displayed_in); usages among:

- **borrowed_by** – borrowed by *username*
- **shared_to** – shared to *username*
- **glmember_of** – member of *grouplistname*
- **displayed_in** – used by display *username/displayid*
- **child_of** – is a direct calculated child of *serieskey*
- **parent_of** – is part of the formula defining *serieskey*
- **descendant_of** – is a calculation that somehow depends on *serieskey*
- **ancestor_of** – somehow contributes to the formula for *serieskey*
- **all** – shorthand for all of the above

payloadasjson= [**true** | **false**] (optional; default = false; if true payload will be JSON encoded)

Payload:

Comma delimited list of *serieskey:USAGE user|USAGE user,serieskey:…*

Example:

```
Call:
func=seriesxref&serieskeylist=UserAlice/00000190&usages=BORROWWED_BY,SHARED_TO&scope=all_
related

Response: {"func":"seriesxref","code":"0","message":"OK","payload":"
UserAlice/00000190:BORROWED_BY mbdcab6|SHARED_TO fivePMB"}
```

Note:

SeriesXref evaluation can be very time consuming and compute intensive.

3.3 Calculation-related calls

These calls describe and manipulate calculated series in dataZoa.

3.3.1 Func=CalcList

Purpose:

Enumerate all calculated series in the account

Arguments:

None.

Payload:

Comma delimited list of serieskeys.

Example:

```
Call: func=calclist

Response: {"func":"calclist","code":"0","message":"","payload":"
UserAlice/00000227,UserAlice/00000228"}
```

3.3.2 Func=CalcGetAttr

Purpose:

Fetch the value of an attribute of a calculated series.

Arguments:

serieskey=*serieskey*
attrname=*attrname*, among:

All calculated series have these attributes:

- **parents** – return a list of serieskeys that are direct inputs to the formula that defines this series.

- ancestors – return *parents*, as above, plus all parents of those parents, recursively.
- **sortedancestors** -- return *ancestors*, as above, topologically sorted into their evaluation order
- **children** – return a list of serieskeys of calculated series that directly depend on this series
- **descendants** – return *children*, as above, plus all children of those children, recursively
- **sorteddescendants** – return *descendants*, as above, topologically sorted into their evaluation order

Calculated series that were made using the dataZoa UI also have these attributes:

- **version** – integer >= 1
- **formula** – calculation definition with "V1," "V2," etc. as the variable placeholders
- **vars** -- list of serieskeys, representing "V1," "V2," etc. in the formula
- **title** – series title, for convenience
- **conformity** – integer, see ComputeCloud documentation
- **nafill** -- integer, see ComputeCloud documentation

Payload:

Attribute value.

Example:

```
Call:  func=calcgetattr&serieskey=UserAlice/00000298&attrname=descendants

Response:
{"func":"calcgetattr","code":"0","message":"","payload":"PublisherBob/00000327,PublisherA
lice/00000112"}
```

3.3.3 Func=CalcEvalAccount

Purpose:

Re-evaluate all calculated series in the account as needed

Arguments:

None.

Payload:

None.

Example:

```
Call: func=calcevalaccount

Response: {"func":"calcevalaccount","code":"0","message":"","payload":""}
```

Note:

Launches any required calculations and returns asynchronously. May take considerable elapsed time depending on the number of series in the account. May be throttled to regulate resource usage. To avoid excessive throttling, consider using the more sophisticated function CalcEvalDescendants.

3.3.4 Func=CalcEvalDescendants

Purpose:

Re-evaluate calculated series that in some way depend on the serieskeylist supplied

Arguments:

serieskeylist=*serieskey[,serieskey]…*

Payload:

None.

Example:

```
Call: func=calcevaldescendants

Response: {"func":"calcevaldescendants","code":"0","message":"","payload":″"}
```

Note:

Launches any required calculations and returns asynchronously. May take considerable elapsed time depending on the number of series in the account. May be throttled to regulate resource usage.

In pseudo code, typical appropriate usage would be:

```
headwaters={“PublisherAlice/00000422”,” PublisherAlice/00000424”}
for each serieskey in headwaters
        call SeriesReplace serieskey, new datavalues
next
call CalcEvalDecendants headwaters
```

3.4 Grouplist Manipulation

These calls are used to manipulate Grouplists at dataZoa.

3.4.1 Func=GLList

Purpose:

Enumerate all grouplists in the account

Arguments:
None.

Payload:
Comma delimited list of grouplist names.

Note:
Commas in returned names will be encoded to %2C so that result set itself can be comma delimited.

3.4.2 Func=GLDelete

Purpose:
Delete a grouplist.

Arguments:
glname=*glname*

Payload:
None, but may return a descriptive message.

Note:
The descriptive message will always report the grouplist as dropped, whether or not it existed at the time. Use the function SeriesFromGL if you need to know whether a grouplist actually exists before or after GLDelete.

3.4.3 Func=GLFromSeries

Purpose:
Create a grouplist.

Arguments:
glname=*glname*
serieskeylist=*serieskey [,serieskey]…*

Payload:
None, but may return a descriptive message and a status code of 200 to indicate a problem condition.

3.4.4 Func=SeriesFromGL

Purpose:
Returns the serieskey(s) for each series contained within a given grouplist.

Arguments:
glname=*glname*

Payload:

Comma delimited list of *serieskeys*. Note that any series no longer available will be reported with "NOTFOUND:sidGUID" where the sidGUID is an internal dataZoa encoding of the key that is no longer available. .

.

3.5 Display Manipulation

3.5.1 Func=DisplayList

Purpose:

Enumerate all displays in the account

Arguments:

type= [**table** | **chart** | **latestvalues** | **datablock** | **widget**] (optional; default = all display types)
fields= [*field* [,*field*…] (optional; default= no extra fields), where *field* among:

- **name –** return name of display
- **type –** return display type

Payload:

Comma delimited list of displayids.

If additional fields are specified, payload is a comma delimited list of *displayid*|*fieldvalue*|*fieldvalue*…

Example:

```
Call: func=displaylist&fields=name,type

Response: {"func":"displaylist","code":"0","message":"OK","payload":" F925660947|CHART|12
T1,6217644B0C|TABLE|AAA League"}
```

Note:

IMPORTANT: Because some returned fields could contain the comma or vertical bar delimiters as part of their text, all fields have substitutions made as needed, which the caller must reverse upon receipt. The substitutions are:

"_comma_" for ","
"_vertbar_" for "|"

3.5.2 Func=DisplayCreate

Purpose:

Create a new display. Analogous to the "New" choice in any of the display editors in the dataZoa UI.

Arguments:

type= [**table** | **chart** | **latestvalues** | **datablock** | **widget**]
name= *displayname*

serieskeylist=*serieskey[,serieskey]…* | *glname*

Payload:
A new *displayid*

Example:

```
Call: func=displaycreate&type=table&name=Elegance&serieskeylist=UserAlice/00000190,
UserAlice/00000191

Response: {"func":"displaycreate","code":"0","message":"OK","payload":"B217633B0C"}
```

3.5.3 Func=DisplayClone

Purpose:
Create a new copy of an existing display. This is analogous to opening an existing display in any of the display editors in the dataZoa UI and then saving under a new name.

Arguments:
displayhandle=*diplayhandle* of an existing display
ownerhandle=*ownerhandle* of the owner of displayhandle argument
name=*displayname*
serieskeylist=*serieskey[,serieskey]…* | *glname* (optional; default = same series as source display)

Payload:
A new *displayid*

Example:

```
Call: func=displayclone&displayhandle=B217633B0C&ownerhandle=UserAlice&name=Elegance
Redfined&serieskeylist=UserAlice/00000192,UserAlice/00000193

Response: {"func":"displayclone","code":"0","message":"OK","payload":"EE25633CDF"}
```

Notes:
Permission to clone a display from one account to another may be mediated by the display's READABILITY attribute and/or Related Account membership.

3.5.4 Func=DisplayGetAttr

Purpose:

Fetch the value of a particular display attribute and/or enumerate all the values used in a display.

Arguments:

displayhandle=*diplayhandle* of an existing display
ownerhandle=*ownerhandle* of the owner of displayhandle argument
attrname= [*attrname* | **allprimitive** | **allderived** | **allmerged**], where:

- *attrname* is a specific attribute name (see notes below)
- **allprimitive** requests all of the fundamental explicit display attributes
- **allderived** requests all of the display attributes that depend on other attributes
- **allmerged** requests all the attributes, both primitive and derived

Payload:

When a single *attrname* is requested, the value of the requested *attrname* is returned. When one of the "**all**…" *attrnames* is requested, a list of *attrname*:*attrvalue* pairs is returned (with pairs being delimited by vertical bars).

Example:

```
Call:  func=displaygetattr&displayhandle=B217633B0C&ownerhandle=UserAlice&attrname=HEIGHT

Response: {"func":"displaygetattr","code":"0","message":"","payload":"333"}
```

Notes:

Unlike most API arguments, values for the *attrname* argument are case sensitive.

DataZoa displays can have many attributes. See the notes for the DisplaySetAttr function for a table of some of the more common and useful attributes.

IMPORTANT: Because some returned fields could contain the colon or vertical bar delimiters as part of their text, all fields have substitutions made as needed, which the caller must reverse upon receipt. The substitutions are:

"_colon_" for ":"
"_vertbar_" for "|"

3.5.5 Func=DisplaySetAttr

Purpose:

Set the value of a particular display attribute.

Arguments:

displayhandle=*diplayhandle* of an existing display
ownerhandle=*ownerhandle* of the owner of displayhandle argument

attrname=*attrname*
attrvalue=*attrvalue*

Payload:

Empty, although a confirmation message may be returned in the message field of the response.

Example:

```
Call:
func=displaysetattr&displayhandle=B217633B0C&ownerhandle=UserAlice&attrname=HEIGHT&attrva
lue=333

Response: {"func":"displaysetattr","code":"0","message":"Attribute 'HEIGHT'
set.","payload":""}
```

Notes:

Unlike most API arguments, values for the *attrname* argument are case sensitive.

An attribute may be removed by setting its value to the constant "##REMOVE##"

DataZoa displays can have many attributes. Some of the more common and useful ones are illustrated in this table:

Attrname	Description	Typical attrvalue
AUTOATTRIBUTION	Infer attribution text from data	True
AUTOFOOTNOTE	Infer footnote text from data	True
BG_COLOR	Background color	#332211
FG_COLOR	Foreground color	#112233
FONTFAMILY	Font family	verdana, Arial, sans-serif
HEADCOLOR	Color of display header	#333333
HEADERTEXTCOLOR	Color of display header text	#FFFFFF
HEIGHT	Overall height	333
HORIZ_MARGIN	Outside display margin	6
HOVERNOTES	Offer details when hovering display titles	True
LABLEFILTERS	Common title "label knockouts"	(billions)
SERIES	List of series shown in display	UserAlice/00000190
TEXCOLOR	Main text color	#111111
TITLE	Overall title	Graduation Rates
VERT_MARGIN	Outside display margin	6
WIDTH	Overall width	400

3.6 Ownership Manipulation

3.6.1 Func=RAGetLeader

Purpose:

Returns the *accounthandle* information for the leader of related accounts group of the caller.

Arguments:

format= [*username* | *dzuuid* | *origemail* | *emailtouse*] (optional; default = username)

Payload:

An *accounthandle*, formatted as requested.

Example:

```
Call: func=ragetleader

Response:
{"func":"ragetleader","code":"0","message":"","payload":"PublisherCompany,4422,dzleader@p
ubco.com,dzleader@pubco.com"}
```

Note:

When called by accounts that do not participate in a related accounts group, the leader is simply the account itself.

3.6.2 Func=RAGetMembers

Purpose:

Enumerates the *accounthandle* information for each member of the related accounts group that the caller belongs to.

Arguments:

format= [*username* | *dzuuid* | *origemail* | *emailtouse*] (optional; default = username)

Payload:

A list of *accounthandles*, formatted as requested.

Example:

```
Call: func=ragetmembers

Response:
{"func":"ragetmembers","code":"0","message":"","payload":"PublisherCompany,UserAlice,User
Bob"}
```

Note:

When called by accounts that do not participate in a related accounts group, the list of members is simply the account itself.

3.6.3 Func=RAChown

Purpose:

Change ownership of objects among members of a Related Accounts Group.

Arguments:

keytype= [**serieskey** | **glname** | **displayhandle**, **serieskey**]
keylist=*key* [, *key*…]
currentownerhandle=*ownerhandle*
newownerhandle=*ownerhandle*

Payload:

Comma delimited list of *key:true* or *key:false*; true if the change of ownership was successful, false otherwise.

Example:

```
Call:
func=rachown&keytype=displayhandle&currentowner=UserAlice&newowner=PublisherCo&keylist=
EE25633CDF,2276643EED

Response:
{"func":"rachown","code":"0","message":"","payload":”EE25633CDF:true,2276643EED:true"}
```

4 Encodings

4.1 URLEncode before sending argument values

While not strictly needed in every case, it is good practice to URLEncode the *value* portion of any arg=value pairs being sent in an API request.

When POSTed data is read by the API receiver, a request header of

"Content-type","application/x-www-form-urlencoded"

is presumed.

4.2 JSON decode responses

Generally speaking, there will be four standard JSON encapsulated values returned for any function call, as discussed earlier. Within those responses, however, further "un-escaping" may be required, as discussed below.

4.3 Miscellaneous escapes

After extracting the JSON payload of a response from the API, certain substitutions made in the payload will need to be translated back from an escaped form before the payload is back to its original representation.

Specifically, and in this order, perform these substitutions:

""" → double quote character
">" → ">"
"<" → "<"
"\n" → system native newline
"\t" → system native TAB
"\\" → "\"

5 API Usage Governors

To ensure balanced throughput, API calls are subject to certain usage limitations, described here. The numeric limits given are per account. These limitations are upper bounds, and may be dynamically lowered during periods of high overall system loading.

Generally speaking, dataZoa handles the limitations transparently with usage governors, so that API client programs themselves do not have to implement rate limits.

5.1 Connection limit (simultaneous calls)

It is often sensible to launch several API client processes in parallel to reduce wall clock time for a given task. Please note that no more than 50 API calls per account can be in process at any one time. When exceeded, the API call will fail with error code 901 (E_CONLIMIT). While the connection limit may be adjusted downward dynamically, it will never be adjusted below the value 5. Therefore, applications can either limit themselves to 5 request threads or else try for more and be able to adjust for fewer.

5.1 Rate limits

There are limits to the rate at which API requests will be processed. No more than 50 requests per second or 500 requests per minute will be served. As a rate limit is approached, dataZoa will begin to apply small sleep periods to requests before processing them. Thus, the calling application does not need to implement a rate governor. While there are error codes (e.g. E_HPSLIMIT) defined for excessive rates, they are not returned in normal practice.

In the case of evaluation request (e.g. CalcEvalAccount), the applicable rate limit is two calls per minute. This small number reflects the proper usage of such calls, which are typically intended to be used only at the conclusion of a batched set of calls to SeriesReplace.

6 API Versioning

The dataZoa API uses a three part versioning scheme; Version.Revision.BuildNumber.

BuildNumber is incremented with each public release, and is zeroed when Revision is incremented. It will change for bug fixes or undocumented changes.

Revision is incremented when documented changes are made, such as new functions or arguments. It is zeroed when Version is incremented.

Version is incremented when entirely new families of capabilities are introduced.

7 Standard Response Codes, applicable to all Calls

0 = Success

901 = Too many simultaneous connections (E_CONLIMIT)
911 = Too many hits per second (E_HPSLIMIT)
912 = Too many hits per minute (E_HPMLIMIT)
913 = Too many evaluation requests per minute (E_EPMLIMIT)

941= Permission denied
943= Forbidden
944= Not found

950= Internal error
951= Unimplemented

800 = Malformed authentication
801 = Unable to authenticate – user not recognized
802 = Unable to authenticate – incorrect password

700 = Malformed request
701 = Unknown function
702 = Missing argument
703 = Timed out
704 = RUNSTATE too low
705 = Invalid argument
706 = Invalid argument value