Data Actions

Data actions read from or write to XINA databases.

Read Actions

SELECT

The primary read action in XINA. It closely mirrors the MySQL `SELECT` query, and returns data as a header of columns and list of rows. The full syntax for the `SELECT` object is available here.

Property	Value	Req	Default
action	`"select"`	?	
select	select	?	
rows	integer		`10,000`
use_strings	boolean		`false`
echo	boolean		`false`

The server response to a `SELECT` action will start with a header packet, containing a JSON array of JSON object(s) indicating the `name` of each column as a `string` and the XINA data type of each column as a `string`. This will be followed by packet(s) containing the data, as a JSON array of of JSON array(s) of values.

The optional `rows` property sets the limit of rows per packet. Note that this *does not* limit the total number of rows returned, this is set by the `limit` property of the select object.

If the `use_strings` property is `true`, all values will be stored as JSON strings instead of their associated JSON type.

If the `echo` property is `true`, the generated SQL query will be included in the header object in the `"query"` property. This is provided to support query debugging; it does not affect the query itself.

Example

Given a table `t` with two columns, `a` (`int(4)`), and `b` (`utf8text`), and three rows:

a	b
`0`	`"x"`
`1`	`"y"`
`2`	`"z"`

The following `SELECT` action:

```
{
 "action": "select",
 "select": {
  "from": "t"
 },
 "rows": 2
}
```

Would return three server packets.

First, the header information:

`100`

```
[
 {
  "name": "a",
  "type": "int(4)"
 },
 {
  "name": "b",
  "type": "utf8text"
 }
]
```

Second, the first two rows (limited to two by the `rows` property):

`100`

```
[
 [ 0, "x" ],
 [ 1, "y" ]
]
```

Third, the last remaining row (with the status code `200` indicating the end of the data):

`200`

```
[
 [ 2, "z" ]
]
```

FETCH

Reads specific types of data in a more structured format than the SELECT action. Although the syntax and response format differs depending on fetch type, all fetch actions share the boolean property `"count"`, which if true, overrides the default action output with a single value indicating the total count result for the current selection.

```
{
 "count": <integer>
}
```

FETCH RECORDS

Fetches records from a database.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"records"`	?	
database	database specifier	?	
records	records specifier		
fields	array of record field names		
attributes	arrray of attribute names		
where	expression		
order	array of order terms		default database order
limit	integer		1,000 (see below)
offset	integer		
children	boolean		`true`
count	boolean		`false`

Fetched records are returned as JSON objects, with each attribute and field name as a property key with the corresponding value. The `fields` and `attributes` property can be used to specify which fields and attributes (e.g. insert_at) are returned in the record. If the array is empty, all will be filtered out. If not provided or null, no filtering occurs. The `record_id` and `database_id` attribute is always included in the record and can't be filtered. In databases with the `file` feature enabled, each record will also include a generated presigned URL in the `"file_url"` property, and an S3 key reference in the `"file_key"` property.

If `children` is `true` and the specified database contains one or more child databases, all child records for each record will be included in the result. Each record with children will contain a `"children"` property, a JSON object with each key containing the name of the child database, and each value a JSON array of child record(s).

If both `records` and `where` are provided, they will be combined with a boolean `AND` operation.

The default limit for this operation is 1,000. Unlike the `SELECT` action which streams data directly from the underlying database, `FETCH` involves additional server overhead processing and formatting the result, so a limit

is enforced to maintain system performance. Exceeding the default limit explicitly is permitted but may cause performance issues depending on server configuration.

FETCH MULTIRECORDS

Fetches records from several databases at once.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"multirecords"`	?	
databases	databases specifier	?	
where	expression		
order	array of order terms		default order of first database
limit	integer		1,000 (see below)
offset	integer		
children	boolean		`true`
count	boolean		`false`

Fetched records are returned as JSON objects, in the same format as the standard FETCH RECORDS action, with the same behavior for file and child records. Each top level record will also include a `"database_id"`, with the numeric database ID of the database of origin for the record.

Internally this action uses the multi-database source, which creates a SQL UNION of the record tables of each database as a single virtual table. This is achived by unioning each column by field name. As such this action works best with databases with the same set of fields (names and data types). If databases each have fields with the same names but different types unpredictable server or client side errors may result.

FETCH PSEUDORECORDS

Fetches data from an arbirary query formatted as though it represents a set of records.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"pseudorecords"`	?	
select	select	?	
where	expression		
order	array of order terms		default order of first database
limit	integer		
offset	integer		
count	boolean		`false`

Each row of the result will be formatted as a JSON object, with each property key taken from the `SELECT` response header.

FETCH CRONS

Fetches all crons.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"crons"`	?	

FETCH FOLLOWS

Fetches all follows for a single user.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"follows"`	?	
user	user specifier		current user
count	boolean		`false`

FETCH KEYS

Fetches all keys for a single user. Fetching keys for a different user requires the `SUPER` privilege.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"keys"`	?	
user	user specifier		current user
count	boolean		`false`

FETCH LOGS

Fetches record logs from a database.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"logs"`	?	
database	database specifier	?	
records	records specifier	?	

FETCH NOTIFICATIONS

Fetches notifications for a single user.

Property	Value	Req	Default

action	`"fetch"`	?	
fetch	`"notifications"`	?	
user	user specifier		current user
type	notification type		
seen	boolean		

Notifications will always be returned ordered by time, descending.

If `type` is provided, only notifications of the same type will be returned.

If `seen` is `true`, only seen notifications will be returned. If `seen` is `false`, only unseen notifications will be returned.

FETCH POSTS

Fetches wall posts.

Property	Value	Req	Default
fetch	`"posts"`	?	
wall	wall specifier		all walls
following	boolean		`false`
threads	boolean		`false`
post	post ID		
children	boolean		`false`
records	boolean		`false`

FETCH PREFS

Fetches preferences for a single user.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"prefs"`	?	
user	user specifier		current user

FETCH PREF DEFS

Fetches server preference definitions.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"pref_defs"`	?	

FETCH TASKS

Fetches task information.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"tasks"`	?	
from	task ID		
user	user specifier		
text	string		
where	expression		
order	array of order terms		recent first
limit	integer		1,000 (see below)
offset	integer		
count	boolean		`false`

FETCH TEAM SUBSCRIPTIONS

Fetches subscriptions for a single team.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"user_subscriptions"`	?	
team	team specifier	?	

FETCH SEQS

Fetches all task sequences.

Property	Value	Required	Default
action	`"fetch"`	?	
fetch	`"seqs"`	?	

FETCH USERS

Fetches user information.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"users"`	?	
users	users specifier		all users

Property	Value	Req	Default
order	array of order terms		recent first
limit	integer		1,000 (see below)
offset	integer		

FETCH USER SUBSCPTIONS

Fetches subscriptions for a single user.

Property	Value	Req	Default
action	`"fetch"`	?	
fetch	`"user_subscriptions"`	?	
user	user specifier		current user

DOWNLOAD

The download action generates a signed URL to download a file stored in the XINA system. Note that this does not actually perform the download; the returned URL can be used outside the XINA API to download the file.

DOWNLOAD RECORD

Generates a signed URL for a record file.

Property	Value	Req	Default
action	`"download"`	?	
download	`"record"`	?	
database	database specifier	?	
record	record specifier	?	
version	integer		most recent

DOWNLOAD POST

Generates a signed URL for a post file.

Property	Value	Required	Default
action	`"download"`	?	
download	`"post"`	?	
post	post ID	?	

Write Actions

INSERT

The INSERT action inserts one or more records into a XINA database.

By default, the action will fail if any records being inserted have duplicate key values already in the database. If a different `on_duplicate` property is set, duplicate records will be updated according to the rules in the table. Only fields explicitly set in the `INSERT` will be changed. This is analogous to an `INSERT ... ON DUPLICATE KEY UPDATE` MySQL query.

Property	Value	Req	Default
action	`"insert"`	?	
database	database specifier	?	
records	records data	?	
on_duplicate	`"fail"` or `"update"`		`"fail"`
fail_no_op	`boolean`		`false`

Examples

Given a starting database containing key field `k`, fields `f1`, `f2`, and `f3`, with tags enabled, containing the following two records:

k	f1	f2	f3	tags
a	1	2	3	t1
b	1	2	3	t1

And inserting records:

```
[
 { "k": "a", "f1": 4, "f2": null, "tags": ["t2"] },
 { "k": "c", "f1": 1, "f2": null, "tags": ["t2"] }
]
```

on_duplicate: `"fail"`

Action fails due to duplicate key value `"a"`. No change occurs.

on_duplicate: `"update"`

Record with key value `"a"` is updated, and record with key value `"c"` is inserted. Note that field `f3` of `"a"` is unaffected because no inserted records specified an explicit value for `f3`.

k	f1	f2	f3	tags

a	**4**	**null**	3	t1, **t2**
b	1	2	3	t1
c	**1**	**null**	**null**	**t2**

REPLACE

The REPLACE action inserts one or more records into a XINA database and **overwrites** any existing records with duplicate keys.

Property	Value	Req	Default
action	`"replace"`	?	
database	database specifier	?	
records	records data	?	
on_duplicate	`"update"`, `"delete"`, or `"trash"` (if trash enabled for database)		`"update"`
fail_no_op	`boolean`		`false`

Examples

Given a starting database containing key field `k`, fields `f1`, `f2`, and `f3`, with tags enabled, containing the following two records:

k	f1	f2	f3	tags
a	1	2	3	t1
b	1	2	3	t1

And replacing records:

```
[
 { "k": "a", "f1": 4, "f2": null, "tags": ["t2"] },
 { "k": "c", "f1": 1, "f2": null, "tags": ["t2"] }
]
```

on_duplicate: `"update"`

Record with key value `"a"` is updated, and record with key value `"c"` is inserted. Note that `f3` of `"a"` is now `null` and `t1` is removed because all fields are overridden by the incoming record.

k	f1	f2	f3	tags
a	**4**	**null**	**null**	**t2**
b	1	2	3	t1
c	**1**	**null**	**null**	**t2**

on_duplicate: `"trash"` or `"delete"`

Existing record with key value `"a"` is deleted (or trashed), and new records `"a"` and `"c"` are inserted.

k	f1	f2	f3	tags
b	1	2	3	t1
a	**4**	**null**	**null**	**t2**
c	**1**	**null**	**null**	**t2**

If `"trash"` is used, the trash table now contains the original `"a"` record.

k	f1	f2	f3	tags
a	**1**	**2**	**3**	**t1**

SET

The SET action sets a database to contain the provided, and *only* the provided, records. Other records already present in the database are removed (either trashed or deleted, depending on the provided configuration).

Property	Value	Req	Default
action	`"set"`	?	
database	database specifier	?	
records	records data	?	
on_duplicate	`"update"`, `"delete"`, or `"trash"` (if trash enabled for database)		`"update"`
on_remove	`"delete"` or `"trash"` (if trash enabled for database)		`"trash"` if enabled, `"delete"` otherwise
fail_no_op	boolean		`false`

Examples

Given a starting database containing key field `k`, fields `f1`, `f2`, and `f3`, with tags enabled, containing the following two records:

k	f1	f2	f3	tags
a	1	2	3	t1
b	1	2	3	t1

And setting records:

```
[
 { "k": "a", "f1": 4, "f2": null, "tags": ["t2"] },
```

```
  { "k": "c", "f1": 1, "f2": null, "tags": ["t2"] }
]
```

on_duplicate: `"update"`

Record `"a"` is updated, record `"c"` is inserted, and record `"b"` is deleted (or trashed, depending on `on_remove`). Note that `f3` of `"a"` is now `null` and `t1` is removed because all fields are overridden by the incoming record.

k	f1	f2	f3	tags
a	**4**	**null**	**null**	**t2**
c	**1**	**null**	**null**	**t2**

on_duplicate: `"trash"` or `"delete"`

All existing records are deleted (or trashed, depending on `on_remove`), and new records `"a"` and `"c"` are inserted.

k	f1	f2	f3	tags
a	**4**	**null**	**null**	**t2**
c	**1**	**null**	**null**	**t2**

UPDATE

The UPDATE action updates the values of one or more fields and/or attached files of one or more records in a single database.

> This documentation applies to XINA 9.2 and above.

Property	Value	Req	Default
action	`"update"`	?	
database	database specifier	?	
records	records specifier	?	
fields	`jsonobject` map of fields to values to update (see below)		
expressions	`jsonobject` map of fields to expressions to update (see below)		
file	string object ID of file to update (see below)		
fail_no_op	boolean		`false`

The `fields` and `expressions` properties are JSON objects, where each key is interpretted as a field specifier in the context of the current database. For the `fields` property, each value is interpretted as a literal JSON value for the type of the specified field. For the `expressions` property, each value is interpretted as an expression, with the evaluated result of the expression stored with the record. These are provided separately because expressions would otherwise not be distinguishable from JSON object value literals.

The `file` property may be provided for databases with the `file` feature enabled. When the file is updated, associated file record attributes (`file_size`, `file_type`, etc) will be updated automatically from the new file.

If a single field is referenced more than once across the `fields` and `expressions` object, the action will fail, as the result would be ambiguous.

By default, if no records are found matching the records specifier, or no values are provided for `fields`, `expressions`, and `file`, the action will complete successfully without any changes occuring. If `fail_no_op` is `true`, the action will fail. Note, however, that `fail_no_op` will only detect these specific no-op conditions; it is possible that no changes will occur if provided update(s) do not actually change any fields of matched record(s).

DELETE

The DELETE action deletes one or more records from a database.

*Note that deleted records and all associated data are **permanently deleted** and cannot be restored.*

This action requires the `DELETE` database privilege.

Property	Value	Req	Default
action	`"delete"`	?	
database	database	?	
records	records	?	
fail_no_op	boolean		`false`

TRASH

The `TRASH` action moves one or more records into the trash table of a database. This is only available in databases with the trash feature enabled, otherwise the action will fail.

This action requires the `TRASH` database privilege.

Property	Value	Req	Default
action	`"trash"`	?	
database	database	?	
records	records	?	
fail_no_op	boolean		`false`

RESTORE

The `RESTORE` action moves one or more records from the trash table of a database into the record table. This is only available in databases with the trash feature enabled, otherwise the action will fail.

If any records being restored have duplicate keys as other records currently in the database the action will fail.

This action requires the `TRASH` database privilege.

Property	Value	Req	Default
action	`"restore"`	?	
database	database	?	
records	records	?	
fail_no_op	boolean		`false`

DISPOSE

The `DISPOSE` action deletes one or more records from the trash table of a database. This is only available in databases with the trash feature enabled, otherwise the action will fail.

*Note that disposed records and all associated data are **permanently deleted** and cannot be restored.*

This action requires the `DELETE` database privilege.

Property	Value	Req	Default
action	`"dispose"`	?	
database	database	?	
records	records	?	
fail_no_op	boolean		`false`

Revision #60
Created 9 June 2022 16:17:19 by Nick Dobson
Updated 6 February 2026 14:03:26 by Bradley Tse