# USC-IR

***Release 0.4.0-a***

**winter, zkldi**

**Jun 10, 2023**

# COMMON:

This is documentation for the USC-IR (unnamed_sdvx_clone Internet Ranking) spec. This standard should be followed exactly, to avoid incompatibilities with clients and skins.

CHAPTER

ONE

# SPEC INFORMATION

The USC-IR spec follows Semantic Versioning.

The current version of this spec is `v0.4.0-a`.

There's not really much more to it! This is intended to be a simple, clean spec, implemented by anyone.

CHAPTER

TWO

# COMMON API INFORMATION

## 2.1 Architecture

USC-IR operates over HTTP or HTTPS. The USC client will make requests to endpoints at given times, such as POSTing `/score/submit` when a score is achieved.

USC-IR only makes use of GET and POST, to simplify implementation.

USC-IR is intended to operate **on top** of HTTP(S), to simplify its implementation. Any request that is received by the server successfully should be responded to with a HTTP 200 OK. Rejection reasons that are not a failed request should be indicated using a USC-IR status code in the response body, as detailed in Response Format.

---

**Note:** Any HTTP response bearing a non-200 status code will be interpreted by the client as a generic failure, and will not be consumed.

---

## 2.2 Request Format

Unless otherwise specified, all data sent to the server is expected to be in JSON format, with `Content-Type: application/json` set in request headers.

To authenticate with the server, users are expected to send an Authorization header, containing `Bearer <token>`.

How the server distributes these tokens is up to them, but they are to be used to authenticate who is making what request.

The body sent is dependent on the endpoint targeted by the request.

## 2.3 Response Format

All data returned from the server is in JSON format.

---

**Note:** As noted above, if the server returns a HTTP code other than `200 OK` , the response body will not be consumed.

---

The below table indicates the two-digit USC-IR status codes to be used by the server in response to a request.

| statusCode | Title | Meaning |
|---|---|---|
| 20 | Success | Request succeeded. |
| 22 | Accepted | Request was received, but not yet acted upon. |
| 40 | Bad Request | The request was malformed. |
| 41 | Unauthorized | No token, or an invalid token, was provided. |
| 42 | Chart Refused | The server is not accepting scores for this chart. |
| 43 | Forbidden | The token has been banned. |
| 44 | Not Found | The requested item was not found. |
| 50 | Server Error | The server encountered an error while handling the request. |

### 2.3.1 Always Present Keys

Returned JSON objects will *always* have these keys.

| Key | Type | Description |
|---|---|---|
| `statusCode` | 20 \| 22 \| 40 \| 41 \| 42 \| 43 \| 44 \| 50 | See table above. |
| `description` | String | A human-readable message, which can be displayed to the user. |

### 2.3.2 Conditional Keys

These are keys that are only present under certain scenarios, such as keys that make no sense under certain `statusCode`s.

| Key | Type | Description |
|---|---|---|
| `body` | Object | Must be present on `statusCode 2X`. Contains the results of your request (such as score data, chart data, etc.) |

## 2.4 Endpoint Commonalities

All endpoints must obey the following assumptions:

1. All endpoints are authenticated. This means the Authorization header must be provided in the request, and that the server must respond with `41 Unauthorized` or `43 Forbidden` as and when appropriate.
2. An endpoint should respond with `40 Bad Request` for any request which has parameters that are invalid - this includes parameters which the server does not support, for instance an unsupported leaderboard mode.

## 2.5 Shared Structures

Several structures will be reused multiple times in this specification. Any such structures are detailed below, to avoid repetition.

This structure is the common format for a 'score' that the server will respond with whenever a score is requested.

Table 1: Server Score Object

| Key | Type | Description |
|---|---|---|
| `score` | Integer [0, 10'000'000] | The numeric score the user achieved. |
| `lamp` | 0 \| 1 \| 2 \| 3 \| 4 \| 5 | The lamp for this score. They correspond to the following; "NO PLAY", "FAILED", "CLEAR", "EXCESSIVE CLEAR", "ULTIMATE CHAIN", "PERFECT ULTIMATE CHAIN". |
| `timestamp` | Integer | Time in seconds elapsed since Unix Epoch, indicating when this score was achieved. |
| `crit` | Integer | Hits inside the critical window. |
| `near` | Integer | Hits inside the near window. |
| `error` | Integer | Missed notes. |
| `ranking` | Integer | The ranking for this score, (i.e. #5). |
| `gaugeMod` | "NORMAL" \| "HARD" \| "PERMISSIVE", | The gauge used for this score. |
| `noteMod` | "NORMAL" \| "MIRROR" \| "RANDOM" \| "MIRRAN" | The note modifier used for this score. |
| `username` | String | The username who achieved this score. |

CHAPTER

THREE

# HEARTBEAT - GET /

Used to check your connection to the server, and receive some basic information.

## 3.1 Expected Request

No data is expected.

## 3.2 Expected Response

The server should respond with 20, and some basic information in the `body`:

Table 1: Body

| Key | Type | Description |
|---|---|---|
| `serverTime` | Integer (Unix Seconds) | The current time according to the server. |
| `serverName` | String | The name of the server. This may be displayed to the user. |
| `irVersion` | String | The version of this spec implemented by the IR. |

CHAPTER

FOUR

# CHART TRACKED - GET /CHARTS/:CHARTHASH

Used to check if the server will accept a score for a given chart in advance of submitting it.

## 4.1 Expected Request

Request data is entirely in the URL; viz. the value of `:chartHash` should be the chartHash you are interested in.

## 4.2 Expected Response

If the server would refuse a score for this chart (e.g. it is blacklisted, or the server does not know it and does not accept unknown charts), respond with `statusCode 42`.

Otherwise, respond with `statusCode 20`. The difference between a tracked chart and an unknown chart on a server that accepts unknown charts may be distinguished in the response description.

CHAPTER

FIVE

# RECORD - GET /CHARTS/:CHARTHASH/RECORD

Used to retrieve the current server record for the chart with the specified hash.

## 5.1 Expected Request

No data is expected.

## 5.2 Expected Response

If the server refuses to track this chart, e.g. because it is blacklisted, it should respond with `statusCode 42`
Otherwise, if the server does not know of this chart, it should respond with `statusCode 44`. This is also the correct response if the server is aware of the chart but has no scores on it.
Otherwise, the server will respond with 20, and the following information in the body:

Table 1: Body

| Key | Type | Description |
|---|---|---|
| `record` | Server Score Object | The current server record. |

CHAPTER

SIX

# LEADERBOARD - GET /CHARTS/:CHARTHASH/LEADERBOARD

Used to retrieve some particular useful subset of the scores from the server.

## 6.1 Expected Request

The request is expected to include query parameters `mode` and `n`, where `n` is the limit of scores requested and `mode` is one of the following:

| mode | Meaning |
|---|---|
| best | Return the top n personal bests for this chart. |
| rivals | Return the top n personal bests by players designated as this player's rivals. (server implementation dependent) |

## 6.2 Expected Response

If the server refuses to track this chart, e.g. because it is blacklisted, it should respond with `statusCode 42`
Otherwise, if the server does not know of this chart, it should respond with `statusCode 44`.
Otherwise, the server will respond with 20, and the following information in the body:

Table 1: Body

| Key | Type | Description |
|---|---|---|
| `scores` | Array<Server Score Object> | The requested scores, sorted in descending order by their `score` field. |

CHAPTER

SEVEN

# SCORE SUBMISSION - POST /SCORES

Sends a score to the server.

## 7.1 Expected Request

Table 1: Root

| Key | Type | Description |
|---|---|---|
| `chart` | Chart Object | Contains information about the chart being played. This may be used by the server to accept new charts onto the IR. |
| `score` | Score Object | Contains information about the users' score. |

### 7.1.1 Chart Object

Table 2: Chart Object

| Key | Type | Description |
|---|---|---|
| `chartHash` | String | The unique identifier for the chart the user played. |
| `artist` | String | The artist who created the song. |
| `title` | String | The song title. |
| `level` | Integer [1,20] | The difficulty level assigned to this chart. |
| `difficulty` | 0 \| 1 \| 2 \| 3 | The difficulty of the chart. 0 = NOV, 1 = ADV, 2 = EXH, 3 = INF. |
| `effector` | String | The effector (charter) for the chart. |
| `illustrator` | String | The illustrator for the chart jacket. |
| `bpm` | String | A string representing BPM. For charts with multiple bpms, they are separated by a hyphen, like x.xx-y.yy. |

### 7.1.2 Score Object

Table 3: Score Object

| Key | Type | Description |
|---|---|---|
| `score` | Integer [0, 10'000'000] | The numeric score the user achieved. |
| `gauge` | Float | The gauge the user had at the end of the chart. Depending on gameflags, this should be used by the server to determine the clear type on the chart. |
| `timestamp` | Integer (unix_seconds) | Time in seconds elapsed since Unix Epoch, indicating when this score was achieved. |
| `crit` | Integer | Hits inside the critical window. |
| `near` | Integer | Hits inside the near window. |
| `early` | Integer | Hits inside the near window which were early. |
| `late` | Integer | Hits inside the near window which were late. |
| `combo` | Integer | Best combo reached. |
| `error` | Integer | Missed notes. |
| `options` | Options Object | The options in use. Includes gauge type, etc, see below. |
| `windows` | Object: {`perfect` , `good` , `hold` , `miss`, `slam` } | Indicates what the hit windows were for this score. The defaults are; 46, 150, 150, 300, and 84 respectively. |

**Warning:** It is highly advised for servers to reject scores with non-standard `score.windows` unless specifically implementing a hard-mode option.

**Warning:** `score.timestamp` is in unix seconds, which is different to the default in languages of the JavaScript family (unix_miliseconds) and the .NET family (Ticks). Make sure to account for this if your server expects a different format for time!

### 7.1.3 Options Object

Table 4: Options

| Key | Type | Description |
|---|---|---|
| `gaugeType` | Integer | An enum value representing the gauge type used. 0 = normal, 1 = hard. Further values are not currently specified. |
| `gaugeOpt` | Integer | Not used at the moment. Intended for blastive rank, etc. in the future. |
| `mirror` | Boolean | If mirror is enabled. |
| `random` | Boolean | If random is enabled. |
| `autoFlags` | Integer | A bitfield of elements of the game that are automated. Any non-zero value means that the score was at least partially auto. |

## 7.2 Expected Response

If the server refuses to track this chart, e.g. because it is blacklisted, or because the server does not know of it and rejects unknown charts, it should respond with `statusCode 42`.
If the server has received the score, but is holding it in a queue, e.g. for servers which only begin displaying new charts after a certain number of unique players submit scores for it, it should respond with `statusCode 22`. In this case, `body` should be as follows, but with only `sendReplay` (if the server desires the replay - otherwise, an empty object.)

Otherwise, returns the standard API response, with `body` as follows:

Table 5: Body

| Key | Type | Description |
|---|---|---|
| `score` | Server Score Object | A Server Score object representing the user's personal best score. |
| `serverRecord` | Server Score Object | A Server Score object representing the current server record. |
| `adjacentAbove` | Array<Server Score Object> | An array of 0 to N Server Scores adjacently above the user's PB. |
| `adjacentBelow` | Array<Server Score Object> | An array of 0 to N Server scores adjacently below the user's PB. |
| `isPB` | Boolean | True if the score sent in the request is the user's new PB. |
| `isServerRecord` | Boolean | True if the score sent in the request is the new server record. |
| `sendReplay` | String | If provided, the server is requesting that the replay be sent using the value of this key as the identifier. |

**Warning:** `body.score` **always returns the users PB**. It does **NOT** necessarily return the score you sent.

**Warning:** Several key assumptions are made about the response by the client, which must be upheld by the server. They are as follows:

- `adjacentAbove` will never contain the current server record.
- The returned scores will always descend in the set [... `adjacentAbove`, `score`, ... `adjacentBelow`]. For clarification, see the note below.
- An individual user should only have a maximum of one score in the above set. This is because the scores sent should always be personal bests, not any stored score.
- As a corollary to the above, the requesting user's scores can never appear in the adjacent scores, since their personal best will always be contained in `score`.

**Note:** The server may decide on the value of N to use for adjacentAbove/Below. However, there is limited space to display the scores. For maximum compatibility with skins, a value of 2 or 3 is recommended.

**Note:** The use for `score.adjacent[Above|Below]` and `score.serverRecord` is illustrated in the table below.

| Element | Score | Ranking |
|---|---|---|
| `serverRecord` | LV.MINI 10,000,000 | #1 |
| | … | |
| `adjacentAbove[0]` | zkldi 95,753,163 | #8 |
| `adjacentAbove[1]` | NEIL.C 94,472,194 | #9 |
| `score` | YOU 93,193,547 | #10 |
| `adjacentBelow[0]` | POG 92,541,147 | #11 |
| `adjacentBelow[1]` | CHAMP 91,260,754 | #12 |

CHAPTER

EIGHT

# REPLAY SUBMISSION - POST /REPLAYS

Used to submit the replay for a given score when requested by the server.

## 8.1 Expected Request

**Note:** This endpoint is expected to receive data with a Content-Type of `multipart/form-data`, as a result of the fact that it sends a file. Regardless, the server is expected to **respond** with a Content-Type of `application/json`.

The data will contain `identifier`, which is the identifier sent by the server under `sendReplay` after score submission. It will also contain the replay file under the key `replay`.

## 8.2 Expected Response

If the identifier does not correspond to one of the requesting player's scores, the server should respond with `statusCode 44`.

Otherwise, if the identified score already has a replay, the server should respond with `statusCode 40`.

Otherwise, the server can respond with `statusCode 20`, and the regular format thereof. No particular data is required in the body response.

CHAPTER

NINE

# IR GLOBALS

This page documents variables and functions added to the global scope which are accessible in every script.

## 9.1 IRData

IRData contains values that are relevant to skins intending to make use of the IR.

### 9.1.1 States

The following constants are accessible under the `IRData.States` table, which correspond to the USC-IR status codes for use in skins. There are also three extended codes, which are not sent by the server but are instead used by USC. The meaning is expressed below.

| Name | Value |
|---|---|
| Unused | 0 |
| Pending | 10 |
| Success | 20 |
| Accepted | 22 |
| BadRequest | 40 |
| Unauthorized | 41 |
| ChartRefused | 42 |
| Forbidden | 43 |
| NotFound | 44 |
| ServerError | 50 |
| RequestFailure | 60 |

- Unused: IR is not being used by the client (no base URL has been specified, etc.)
- Pending: Request has not yet received a response.
- RequestFailure: The request failed for a generic reason (non-200 HTTP code, malformed response, etc.)

### 9.1.2 Active

The value of `IRData.Active` is `true` if an IR URL has been set in the config. Otherwise, it is false.

## 9.2 Request Functions

The below functions are accessible under the `IR` table. They are used to make requests of the IR.

All of these functions are asynchronous and take a callback. This callback is called with the exact JSON returned by the server, as a Lua table. If the request fails, the table will have `statusCode 60` and a generic description.

Here is an example usage:

```
function heartbeatResponse(res)
    if res.statusCode == IRState.Success then
        game.Log(string.format("Connected to %s", res.body.serverName), game.LOGGER_INFO)
    else
        game.Log("Can't connect to IR!", game.LOGGER_WARNING)
    end
end

IR.Heartbeat(heartbeatResponse)
```

### 9.2.1 Heartbeat(callback)

Performs a Heartbeat request.

### 9.2.2 ChartTracked(hash, callback)

Performs a Chart Tracked request for the chart with the provided hash.

### 9.2.3 Record(hash, callback)

Performs a Record request for the chart with the provided hash.

### 9.2.4 Leaderboard(hash, mode, n, callback)

Performs a Leaderboard request for the chart with the provided hash, with parameters mode and n.

CHAPTER

# TEN

# RESULT SCREEN

The following fields are added to the `result` table on the results screen.

```
string chartHash //the hash of the chart that was just played
int irState //current state of the IR score submission request (a USC-IR code, including␣
↪extensions 0/10/60)
string irDescription //the description in the IR response (nil if irState is 0 or 10)
ServerScore[] irScores //more details below, nil if irState != 20
```

**Note:** This screen is a special case where the request will be automatically performed by the game, rather than being requested in Lua.

## 10.1 ServerScore

irScores is an array of ServerScores, whose structure matches the ServerScore structure detailed in the score submission endpoint page, with these two additions:

```
bool yours //this score belongs to the current player
bool justSet //this score belongs to the current player, and is the score that was just␣
↪achieved.
```

An example usage of these extra values can be found in the default skin: if `yours` is true, the score has a gold border. If `justSet` is true, the timestamp is 'Now'.

The array of scores is constructed according to specific rules which make it as easy as possible to display.
In almost all cases, the scores can simply be displayed in order.
It is constructed as follows:

- The first element is the server record, unless the server record should be omitted.
  - The server record is omitted if the player's PB is the server record, as in this case `score` is the server record.
- The next 0-N elements are the scores in `adjacentAbove` from the IR response.
- The next element is `score`, i.e. the current player's PB. This element has some special added values, detailed at the bottom of this page, for ease of use.
- The remaining 0-N elements are the scores in `adjacentBelow` from the IR response.

CHAPTER

# ELEVEN

## SONG WHEEL

The `difficulty` objects available in Lua on the Song Wheel have been modified to include a `string hash` which is the chart hash, for use with the IR.