Custom On-Device Validation (COV) Guide

Oct 15, 2026 · @Kenroy Granville

Custom On-Device Validation (COV) runs your own JavaScript inside the CodeREADr app to check, change or block a scan as it happens, even offline. This guide covers where the script goes, the rules it must follow, the three hooks, what each can change, and the helper functions for storage and lookups.

Setup

Add the script in the service's settings: edit the service, open **Advanced**, turn on **Enable on-device custom validation** and paste your code. Through the API, set the `custom_validation_script` variable.

- The script runs in the CodeREADr apps for iOS and Android.
- It runs on every service type, but what it may change depends on whether the service validates on the device or online (see Results by service type).
- When you change the script, devices receive it with the service's other settings. App users don't need to do anything.

Paste plain JavaScript. The app adds the `<script>` tags itself unless your code already ends with `</script>`.

Script rules

The app runs your script in a hidden web view and waits for each hook to return. A script that breaks these rules can freeze or crash the app.

1. **Run synchronously.** Don't use `async`, `await`, Promises or `fetch()`. A hook that returns a Promise can crash the app.
2. **Return a string.** Every hook returns `JSON.stringify(...)`, never a plain object.
3. **Return quickly.** The app user waits while your hook runs, so avoid long loops and slow network calls. On Android, if a hook hasn't returned within 15 seconds, the app goes on with its own result. On iOS the app waits, so a hook that never returns freezes it.
4. **Use synchronous `XMLHttpRequest` for network calls.** The script runs from a local page, so the server you call must allow cross-origin requests (`Access-Control-Allow-Origin: *`). Network calls only work while the device is online.

5. **No page access.** There is no `document` or DOM. Modern JavaScript (ES6 and later) is supported.
6. **Keep answer values as arrays,** and don't change the reserved properties (see Custom properties).

Catch errors inside your hooks. If a hook throws, the app may run it again, so a side effect such as a counter saved with `putString` could happen twice.

There is no console. To see a value while testing, return it as an error message:

```
return JSON.stringify({ "errorMessage": "Debug: " +
JSON.stringify(myVariable) });
```

A synchronous network call looks like this:

```
function getJson(url) {
  try {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", url, false);   // false = synchronous
    xhr.send(null);
    if (xhr.status === 200) {
      return JSON.parse(xhr.responseText);
    }
    return { error: "HTTP " + xhr.status };
  } catch (e) {
    return { error: e.message };
  }
}
```

The three hooks

Define any of these three functions; the app calls each one at its point in the workflow and waits for its answer. Answers build up as the workflow goes on, so a later hook also sees the earlier answers.

Hook	Runs when the app user	Answers it receives	Can stop the action	Can change scan value, status, response	Can change answers
`onSessionSave(data)`	Saves the Session Info questions	Session Info	Yes	No (there is no scan yet)	Yes

Hook	Runs when the app user	Answers it receives	Can stop the action	Can change scan value, status, response	Can change answers
`onCreateScan(data)`	Submits a new scan	Session Info + questions asked before submit	Yes	Value: yes. Status and response: on-device services only	Yes
`onUpdateScan(data)`	Saves questions asked after the scan (including those shown only for valid or only for invalid scans)	All of the above + those questions	Yes	No	Yes

```
function onSessionSave(data) { return JSON.stringify({}); }
function onCreateScan(data)  { return JSON.stringify(data); }
function onUpdateScan(data)  { return JSON.stringify(data); }
```

The data object

Each hook receives one argument, `data`, describing the scan at that moment. Field values are strings except `isCloudScan`, and `scanStatus` is `"1"` for valid and `"0"` for invalid.

```
{
  "scanValue": "050428375624",
  "scanStatus": "1",
  "scanResponse": "Recorded to device.",
  "answers": [
    { "qid": "1525", "qtype": "manual",   "qtext": "Comments:", "value":
["Donated $5 to the cause."], "eid": "" },
    { "qid": "303",  "qtype": "checkbox", "qtext": "Colors",    "value":
["yellow", "green"],         "eid": "" }
  ],
  "properties": {
    "time_zone": "GMT-04:00, GMT-4",
    "capture_type": "camera_scan",
    "app_used": "2.11.14r1625 for iOS-10.2.1",
    "barcode_format": "UPC_A",
    "scanned_at_utc": "2017-06-02T23:32:00.260Z",
    "scan_engine": "qmv1.9.6r4502"
  },
```

```
  "context": {
    "username": "MyUser",
    "userId": "39430",
    "deviceId": "39434",
    "serviceId": "9859",
    "serviceName": "MyService",
    "isCloudScan": false
  }
}
```

Some fields are missing when their data doesn't exist yet:

- `onSessionSave` receives only `answers`, `properties` and `context`.
- On online services, `onCreateScan` runs before the scan reaches the server, so `scanStatus` and `scanResponse` are not there yet.
- Every answer's `value` is an array, even for single-value questions.

Results by service type

`onCreateScan` can override a scan's result only where the device decides it. On on-device services the app has already validated the scan when your hook runs, and your hook receives that result. On online services the server decides afterwards, so there is nothing to override yet.

Service type	Validated by	What `onCreateScan` receives	Can override status and response
Validate On-Device (downloaded database)	The device	The database result: `"1"` or `"0"` with the value's response text, or an "Invalid Duplicate" result when duplicates aren't allowed and the value was scanned before	Yes
Record On-Device	The device	`"1"`, "Recorded to device."	Yes
Validate (online database)	CodeREADr servers	No status or response yet	No

Service type	Validated by	What `onCreateScan` receives	Can override status and response
Record (online)	CodeREADr servers	No status or response yet	No
Postback	Your server	No status or response yet	No

When an online service's device is offline, the scan is saved on the device with status `"-1"` and the response "Recorded to device." until it can be sent.

Because an on-device hook sees the app's result first, it can replace it. For example, it can turn an "Invalid Duplicate" into a more useful message, or accept a value the database marks invalid.

Returning a result

Every hook returns a string made with `JSON.stringify(...)`, in one of three forms.

Continue. Return the input unchanged, or with your changes to its answers. For `onSessionSave` with no changes, return an empty object.

```
return JSON.stringify(data);   // onCreateScan, onUpdateScan, or
onSessionSave after changing answers
return JSON.stringify({});     // onSessionSave, no changes
```

Stop the action. Return an `errorMessage`. The session save, scan or update does not go through. For a new scan, the app shows the message in an alert with an OK button instead of the scan result, plays no valid or invalid sound, and saves nothing.

```
return JSON.stringify({ "errorMessage": "This value is not allowed here." });
```

Override the result (`onCreateScan`). Return the scan value, status and response you want recorded. A new status and response take effect only on on-device services; on online services only the new scan value applies. Status is the string `"1"` for valid or `"0"` for invalid.

```
return JSON.stringify({
  "scanValue": data.scanValue,
  "scanStatus": "0",
  "scanResponse": "Not allowed"
});
```

Include all three fields. If `scanStatus` is missing, the scan is saved as not validated (`"-1"`).

`onUpdateScan` cannot change `scanValue`, `scanStatus` or `scanResponse`, on any service type.

Changing answers

All three hooks can change answers: set a new `value` on an answer in `data.answers` and return `data`. Only `value` may change.

- Keep `value` an array, even for one value: `["Approved"]`, not `"Approved"`.
- Leave `qid`, `qtype`, `qtext` and `eid` as they are. Changing them can cause errors.
- For checkbox and radio questions, each new value must exactly match one of the question's options.
- Don't rewrite GPS answers (`geo://lat,long,…`) or photo and file answers (a JSON string with file details). The app depends on their exact format.

```
function onCreateScan(data) {
  setAnswerVal("1525", data.answers, "Checked");
  return JSON.stringify(data);
}

function getAnswerObj(qid, answers) {
  if (Array.isArray(answers)) {
    return answers.find(a => a.qid == String(qid)) || false;
  }
  return false;
}

function setAnswerVal(qid, answers, newValue) {
  let answer = getAnswerObj(qid, answers);
  if (answer) {
    answer.value = Array.isArray(newValue) ? newValue : [newValue];
  }
}
```

Custom properties

Custom properties are for services that send scans directly to your own server:

- A Postback service with **Skip our server and go directly to yours** turned on (Direct Scan to URL; API variable `enable_direct_scan`).
- An on-device service with Auto Sync and **Sync scans directly to my server** turned on (API variable `auto_sync_up_url`).

On those services, add keys to `data.properties` and return `data`. Each key arrives at your server as a parameter with the scan.

When scans go to CodeREADr instead, custom keys don't appear in the web app, exports or the API. To keep extra data with those scans, write it to a question's answer (see Changing answers).

```
function onCreateScan(data) {
  data.properties["acme_zone"] = "A";
  return JSON.stringify(data);
}
```

- Change `data.properties` in place rather than returning a new object. The properties you return replace the scan's set, so a key you leave out is lost.
- Give your keys distinctive names, such as a company prefix, so they can't clash with the scan's own fields.
- Don't change or remove these reserved keys: `app_used`, `barcode_format`, `capture_type`, `code_replaced`, `created_at`, `created_at_utc`, `gps_location`, `mask_matched`, `scan_engine`, `scanned_at_local`, `scanned_at_utc`, `time_zone`, `updated_at`.

Helper functions

The app provides `window.CRHOOK`, with functions for storing values on the device and for searching the service's downloaded database and the scans saved on the device. All of them work offline.

Storage

```
window.CRHOOK.putString(key, value)
window.CRHOOK.getString(key, defaultValue)
window.CRHOOK.putSharedString(key, value)
window.CRHOOK.getSharedString(key, defaultValue)
```

- `putString` saves a string on this device for the signed-in user and this service. `getString` reads it back, or returns `defaultValue` if nothing is saved.
- `putSharedString` and `getSharedString` work the same way, but the value is shared across all of the user's services on this device.
- Values never leave the device and are not shared with other devices or users.
- Don't rely on the return value of `putString` or `putSharedString`.
- Top-level variables in your script can keep their values between hook calls, but they are lost when the script reloads. Use `putString` for anything that must last.

Searching the database and saved scans

```
window.CRHOOK.dbSearch(value, exactValue, response, exactResponse, validity,
limit, isOrSearch)
window.CRHOOK.scanSearch(value, exactValue, response, exactResponse,
scanStatus, limit, isOrSearch)
```

`dbSearch` searches the copy of the service's database downloaded to this device. `scanSearch` searches the scans saved on this device for this user and service that have not been uploaded yet (see Multiple devices). Both return a JSON string; read it with `JSON.parse`.

Parameter	Type	Meaning
`value`	string or `null`	The barcode value to match. `null` = any value.
`exactValue`	`true` / `false`	`true` = exact match; `false` = the value contains this text.
`response`	string or `null`	The response text to match. `null` = any response.
`exactResponse`	`true` / `false`	`true` = exact match; `false` = the response contains this text.
`validity` (dbSearch) or `scanStatus` (scanSearch)	`"1"`, `"0"` or `null`	Only valid (`"1"`) or only invalid (`"0"`) entries; `null` = both. A string, not a boolean.
`limit`	string, such as `"10"`	The most results to return.
`isOrSearch`	`true` / `false`	When both `value` and `response` are given, `true` matches either one; `false` must match both.

Always pass all seven arguments, using `null` for filters you don't need.

`dbSearch` returns one object per database entry:

```
[
  { "value": "BARCODE_01", "response": "SOME RESPONSE", "isValid": "1",
"scanCount": "0" },
  { "value": "BARCODE_02", "response": "Another response", "isValid": "1",
"scanCount": "3" }
]
```

For example, to look up the scanned value in the database:

```
var rows = JSON.parse(window.CRHOOK.dbSearch(data.scanValue, true, null,
false, null, "1", false));
if (rows.length > 0) {
  var response = rows[0].response;   // the value's response text
  var isValid = rows[0].isValid;     // "1" or "0"
}
```

`scanCount` is kept only when the service does not allow duplicates; otherwise it is `"0"`. It counts every saved scan of the value, valid or invalid, including scans your script marked invalid. A scan stopped with `errorMessage` is not saved and not counted. Scans from other devices are included only as of this device's last database update.

`scanSearch` returns full scan objects in the same shape as `data` (see The data object), in no guaranteed order. It does not include the scan being created when called from `onCreateScan`.

Multiple devices

A COV script sees only what is on its own device. When several devices scan for the same service, they learn about each other's scans only through database updates, never instantly.

What your script reads	Where it lives	Includes other devices' scans	How current
`getString` / `getSharedString`	This device only	No	Immediate, for this device
`dbSearch` entries and `scanCount`	This device's copy of the service database	Yes, once uploaded to the server	As of this device's last database update (Auto Sync or download), plus this device's own scans since
`scanSearch`	Scans saved on this device and not yet uploaded	No	Scans leave the device when they upload. With Auto Sync on, that is within seconds

Two things follow:

- **Don't build counts or limits on `scanSearch` when Auto Sync is on.** Uploaded scans are removed from the device, so it usually finds few or none, even for scans made on the same device. It suits a device that uploads manually, and only until it uploads.
- **A rule that must hold across devices at the same moment needs online validation.** For example, "accept each value only once across all devices." Two devices that scan the same value before either has synced can both accept it. An online service checks each scan against the server as it happens.

Examples

Example scripts are on the Custom Validation page.