Instructions for loading Assets for delivery to Dealers (or similar use cases).

1 - Edit the variables for "const" that apply.
a) timeInGMT > the time in GMT when duplicate checking for Special Characters resets.
b) apiKey > find your API key in your account profile.
c) serviceId > this defines which service to use for duplicate checking.
d) primaryDbId > this is the Asset database which can be empty or have values in it; you can create a Task to clear it at a scheduled interval, if required.
e) secondaryDbId > this is the Dealer database.
f) requireScanStatus > if set to "1" the value would need to be in the database already to update the response with the Dealer ID.
g) qidForDbResponse > this is the question prompt asking the app-user to enter the Dealer ID.
h) specialWords > these are the prefixes checked for duplicates
i) "msg" messages: edit the message given to the app-use for each case as needed.

2 - When completed, copy the script from "const timeInGMT" to the end below and paste it into the Advanced step under Middleware (ask your account exec for Middleware access).

----- script below ----

```
const timeInGMT = '00:00'; // specify time in GMT as HH:MM
const apiKey = 'ENTER'; // account api_key
const serviceId = 'ENTER'; // ID of this codeREADr loading service
const primaryDbId = 'ENTER'; // database ID to add Asset values to
const secondaryDbId = false; // set to a database ID if question answer should be validated.
const requireScanStatus = false; // Set to '1' to require a valid scan before applying this COV.
// AssetID=scanValue will be the db value, and DealerID=qidDbResponse
const qidForDbResponse = 'ENTER'; // question id used to add value to response text
const specialWords = ['T0', 'WC', 'C0', 'BC']; // these are NOT case-sensitive
const msgUpdateSuccces = "Asset update applied to database.";
const errMsgUpdateFailed = "Database update failed.";
const errMsgDuplicate = "Asset already scanned.";
const errMsgEmptyResponse = "No Dealer ID entered.";
const errMsgInvalidResponse = "Unknown Dealer ID entered.";
const errMsgValuesAreSame = "Asset Scanned as Dealer or Dealer Scanned as Asset.";

function getAnswerVal(qid, data) {
    let answerArray = data.answers;
    if (Array.isArray(answerArray)) {
        var value = false;
        if ( qid === false || qid == '') {
            if (answerArray.length > 0) {
                value = answerArray[0].value;
            }
```

```
        } else {
            var ans = answerArray.find(a => a.qid == qid);
            if (ans) {
                value = ans.value;
            }
        }
        if (value) {
            return Array.isArray(value) ? value[0] : value;
        }
    }
    return "";
}

/**
 * Returns true if retrieving from 'location' i.e. '/scan/retrieve/' or
 * '/database/retrieve/' has a response with a count > limit.
 **/
async function containsViaRetrieve(location, params, limit=0) {
    let apiResponse = await cr.post(location, params);
    apiResponse = JSON.parse(apiResponse);
    if (apiResponse['count'] > limit) {
        return true;
    }
    return false;
}

/*
 * Return true if scan value starts with any word in 'specialWords'
 * array and was scanned some time after 'timeInGMT'.
 */
async function isDuplicate(data) {
    let checkDuplicateValue = false;
    for (let word of specialWords) {
        if (data.scanValue.toLowerCase().startsWith(word.toLowerCase())) {
            checkDuplicateValue = true;
            break;
        }
    }
    if (!checkDuplicateValue) {
        return false;
    }
    let time1 = new Date();
    let time2 = new Date(time1.getTime());
    let t = timeInGMT.split(':');
```

```
    let hr = t[0];
    let min = t[1];
    if (min < 10) {
        min = "0" + min;
    }
    time2.setHours(hr, min, 0);
    let endDate = new Date(time2.getTime());
    if (time1 >= time2) {
        endDate.setDate(time2.getDate() + 1);
    }
    let startDate = new Date(endDate.getTime());
    startDate.setDate(endDate.getDate() - 1);
    let params = {
        scanValue: data.scanValue,
        serviceId: serviceId,
        status: ["1","0"],
        startTimestamp: startDate.toISOString(),
        //endTimestamp: endDate,
        timestampReceived: 1,
        timezone: 'GMT',
        limit: 2,
        api_key: apiKey
    };
    let found = await containsViaRetrieve('/scan/retrieve/', params, 1);
    return found;
}

/**
 * If the scan response is valid then try to parse the response as
 * as an array and append newItem. Otherwise, simply return an array
 * containing only 'newItem'.
 */
function responseAsList(data, newItem) {
    if (data.scanStatus == '1') {
        try {
            let itemList = JSON.parse(data.scanResponse);
            if (Array.isArray(itemList)) {
                itemList.push(newItem);
                return JSON.stringify(itemList);
            }
        } catch (e) { /* response is not an array. */ }
    }
    return JSON.stringify([newItem]);
}
```

```
async function crCustomValidate(data) {
    if (requireScanStatus !== false && requireScanStatus != data.scanStatus) {
        return data;
    }
    let isDupVal = await isDuplicate(data);
    if (isDupVal) {
        return {"scanStatus":"0", "scanResponse":errMsgDuplicate + "\n\n" + data.scanResponse};
    }
    let answerText = getAnswerVal(qidForDbResponse, data).trim();
    if (answerText == '') {
        return {"scanStatus":"0", "scanResponse":errMsgEmptyResponse};
    }
    if (data.scanValue.toLowerCase() == answerText.toLowerCase()) {
        return {"scanStatus":"0", "scanResponse":errMsgValuesAreSame};
    }
    if (secondaryDbId != '' && secondaryDbId !== false) {
        let vars = {
            value: answerText,
            databaseId: secondaryDbId,
            api_key: apiKey
        };
        let found = await containsViaRetrieve('/database/retrieve_value/', vars);
        if ( !found  ) {
            return {"scanStatus": "0", "scanResponse": errMsgInvalidResponse};
        }
    }
    let formattedResponse = responseAsList(data, answerText);
    let apiParams = {
        databaseId: primaryDbId,
        value: data.scanValue,
        response: formattedResponse,
        isValid: 1
    };
    let params = {
        params: JSON.stringify(apiParams),
        api_key: apiKey
    };
    let dbValueResponse = await cr.get('/database/upsert_value/', params);
    dbValueResponse = JSON.parse(dbValueResponse);
    if (dbValueResponse['status'] == 1) {
        return {"scanStatus": "1", "scanResponse": msgUpdateSuccces};
    }
    return {"scanStatus": "-1", "scanResponse":errMsgUpdateFailed+"\n\n"+data.scanResponse};
```

```
}
```