SALESFORCE MARKETING CLOUD • AUTOMATION & DEVOPS

SFMC Bulk Refresh Journey Builder Triggered Sends 2026: Stop Doing It One by One

Editing an email in Content Builder does not refresh a live Triggered Send — SFMC serves the cached version until you pause, publish, and restart each one. Here is the WSProxy script that refreshes an entire folder in one automation, with a log DE.

by Genetrix Technology

Published June 10, 2026 • genetrix.tech/blogs

Genetrix Consulting LLC
304 S. Jones Blvd #5631, Las Vegas, NV 89107, USA | genetrix.tech

SFMC Bulk Refresh Journey Builder Triggered Sends 2026: Stop Doing It One by One

By Genetrix Marketing • Published June 10, 2026 • https://genetrix.tech/blogs/sfmc-bulk-refresh-journey-builder-triggered-sends-2026-stop-doing-it-one-by-one/

Editing an email in Content Builder does not refresh a live Triggered Send — SFMC serves the cached version until you pause, publish, and restart each one. Here is the WSProxy script that refreshes an entire folder in one automation, with a log DE.

You updated 100 email templates. Content blocks refreshed, subject lines approved, everything signed off. Then someone asks whether the Triggered Sends are actually picking up the new content. And you realize every single one still needs to be manually paused, republished, and restarted in Email Studio. One by one.

We ran into this on a client account where the marketing team had updated 80 transactional email templates before a major campaign launch. Nobody had told them that refreshing the email in Content Builder was not enough. The live Triggered Send Definitions were still serving the old cached version and would keep doing so until someone manually refreshed each one. This is one of those SFMC maintenance tasks that looks minor until you are actually sitting in front of it.

This guide covers why the refresh is necessary, what sequence the SOAP API expects, and the WSProxy script we now run as a standard step after any large template update — across the full folder in one automation, with a log DE so we know exactly what happened.

Why Editing the Email in Content Builder Is Not Enough

When you update an email asset in Content Builder — even one that is actively linked to a running Triggered Send Definition — the TSD does not pick up the change automatically. SFMC caches the email content at the point the TSD was last published. Every trigger that fires against that definition continues delivering the cached version until you force a refresh.

The manual fix is three steps per TSD: pause it, publish it, restart it. For a handful of TSDs that takes a few minutes. For 80, it is the better part of a morning and it introduces a real error risk. Someone forgets one, that triggered email quietly serves stale content, and nobody catches it until a subscriber reports it.

> **What the refresh actually does:** Republishing a TSD generates a new JobID linked to the current state of the email asset. The RefreshContent: true SOAP API property is the programmatic equivalent of clicking Publish in the UI. It forces the TSD to re-read the email, resolve all content block references, and create a new active job. All future triggers use this new job going forward.

One thing that catches people out when scripting this: the sequence matters and the API enforces it. You must set the TSD to Inactive before you can push RefreshContent: true. Trying to refresh an Active TSD without pausing first will fail, usually silently. The correct order is always Inactive, then refresh, then back to Active.

Triggers that fire during the Inactive window are queued, not dropped. SFMC holds any trigger events that arrive while a TSD is paused and processes them once it is restarted. Subscribers are not missed but they will see a short delivery delay. For most transactional use cases this is fine. If you are running sends with tight delivery SLAs, do this during off-peak hours.

The Script

We use WSProxy for this rather than the Core Library's TriggeredSend.Init() approach. The Core Library works for one or two TSDs but it is not suited for loops over large sets — more overhead per call, less control over errors, no clean way to log results. WSProxy talks directly to the SOAP API, handles arrays natively, and lets you write every outcome to a Data Extension so you have a full audit trail after each run.

Before running, create a Data Extension called TSD_Refresh_Log with this schema:

Field Name	Type	Len	Notes
RunID (PK)	Text	50	Timestamp plus index to identify the run
CustomerKey	Text	200	TSD external key
TSDName	Text	200	Friendly name for readability
PauseStatus	Text	50	OK or ERROR
PublishStatus	Text	50	OK or ERROR
StartStatus	Text	50	OK or ERROR
ErrorDetail	Text	4000	Full error message if any step fails
RefreshedAt	Date	-	Timestamp of this record

SSJS Script Activity — Bulk TSD Refresh with Logging

```
<script runat="server">
Platform.Load("core", "1");

var prox  = new Script.Util.WSProxy();
var logDE = "TSD_Refresh_Log";
var runID = Now().toString().replace(/\s/g, "_");

var tsdList = [
    { "key": "TSD-CUSTOMER-KEY-001", "name": "Welcome Email" },
    { "key": "TSD-CUSTOMER-KEY-002", "name": "Order Confirmation" },
    { "key": "TSD-CUSTOMER-KEY-003", "name": "Password Reset" }
];

for (var i = 0; i < tsdList.length; i++) {
    var cKey  = tsdList[i].key;
    var cName = tsdList[i].name;
    var pauseStatus = "SKIP", pubStatus = "SKIP", startStatus = "SKIP", errDetail = "";

    try {
        var pauseResult = prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "
TriggeredSendStatus": "Inactive" });
        pauseStatus = pauseResult.Status;
        var pubResult   = prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "
RefreshContent": true });
        pubStatus   = pubResult.Status;
        var startResult = prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "
TriggeredSendStatus": "Active" });
        startStatus = startResult.Status;
    } catch(e) { errDetail = Stringify(e); }

    var de = DataExtension.Init(logDE);
    de.Rows.Add({
        "RunID": runID + "_" + i, "CustomerKey": cKey, "TSDName": cName,
        "PauseStatus": pauseStatus, "PublishStatus": pubStatus,
        "StartStatus": startStatus, "ErrorDetail": errDetail, "RefreshedAt": Now()
    });
}
</script>
```

When the TSD List Is Too Large to Maintain Manually

If your org has TSDs spread across multiple folders and keeping a hand-maintained CustomerKey list is not realistic, drive the script from a folder retrieval instead. The version below retrieves all active TSDs under a given Category ID and refreshes every one it finds.

SSJS Script Activity — Folder-Based Bulk Refresh

```
<script runat="server">
Platform.Load("core", "1");

var prox     = new Script.Util.WSProxy();
var logDE    = "TSD_Refresh_Log";
var runID    = Now().toString().replace(/\s/g, "_");
var folderID = XXXXXX;

var cols    = ["CustomerKey", "Name", "TriggeredSendStatus"];
var filter  = { "Property": "CategoryID", "SimpleOperator": "equals", "Value": folderID };
var results = prox.retrieve("TriggeredSendDefinition", cols, filter);

if (results && results.Results) {
    for (var i = 0; i < results.Results.length; i++) {
        var tsd = results.Results[i];
        if (tsd.TriggeredSendStatus !== "Active") { continue; }
        var cKey = tsd.CustomerKey, cName = tsd.Name;
        var pauseSt = "SKIP", pubSt = "SKIP", startSt = "SKIP", err = "";
        try {
            prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "TriggeredSendS
tatus": "Inactive" }); pauseSt = "OK";
            prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "RefreshContent
": true }); pubSt = "OK";
            prox.updateItem("TriggeredSendDefinition", { "CustomerKey": cKey, "TriggeredSendS
tatus": "Active" }); startSt = "OK";
        } catch(e) { err = Stringify(e); }
        var de = DataExtension.Init(logDE);
        de.Rows.Add({ "RunID": runID+"_"+i, "CustomerKey": cKey, "TSDName": cName, "PauseStat
us": pauseSt, "PublishStatus": pubSt, "StartStatus": startSt, "ErrorDetail": err, "RefreshedA
t": Now() });
    }
}
</script>
```

> **Schedule this off-peak.** Running it at 2 AM means the brief Inactive window per TSD has zero visible impact on subscribers. Running it during business hours means some triggers will queue — which SFMC handles fine, but it is worth being aware of before you run this on a high-volume transactional account.

Before You Run This

- Create the TSD_Refresh_Log DE first. The logging step will fail silently if it does not exist and you will lose your audit trail.
- Test against one or two non-critical TSDs before pointing it at your full list or folder.
- The folder-based version only processes Active TSDs. Anything in an intermediate state is skipped and shows up as SKIP.
- Run this as an SSJS Script Activity in Automation Studio, not from a CloudPage. CloudPages time out on large sets.
- After the run, send a test trigger on at least two TSDs to confirm the new content is live before signing off.
- If a TSD fails at the Pause step, do not re-run the whole script. Fix that TSD manually first, then re-run only the ones that failed.

Frequently Asked Questions

Does this work for TSDs connected to Journey Builder?

Yes. Journey Builder Triggered Send interactions use the same underlying TSD object model, so refreshing them via this script updates the content the Journey uses. In some Journey configurations you may also need to

republish the Journey itself for all changes to fully propagate.

How do I get all my TSD CustomerKeys without opening each one manually?

Run a WSProxy retrieve call and write the results to a DE: prox.retrieve with the TriggeredSendDefinition object and CustomerKey, Name, and TriggeredSendStatus columns. This gives you a full inventory you can use to build your tsdList array.

What if a TSD gets stuck Inactive because of a script failure?

Check the ErrorDetail field in the log DE to understand what caused the failure. To restart it, you can either use the Email Studio UI or push a direct TriggeredSendStatus: Active update via WSProxy. Do not retry the full script run until you have resolved what caused that specific TSD to fail.

Should we migrate off classic Triggered Sends entirely?

For new builds, yes. Salesforce's Transactional Messaging REST API is the modern replacement. It resolves content at send time rather than at publish time, which means this entire refresh problem simply does not exist. If you are maintaining a legacy implementation with dozens of TSDs, this script is the right fix for now. Migration to the Transactional Messaging API is the right call long term.