NetSuite Shipment Tracking: Integrating Carrier Webhooks

By Houseblend Published August 5, 2025 40 min read

Real-Time Shipment Tracking in NetSuite via Carrier Webhooks

Overview of Shipment Tracking in NetSuite

NetSuite provides integrated shipping features that allow generating carrier labels, storing tracking numbers, and calculating real-time shipping rates within the ERP (Source: docs.oracle.com)(Source: docs.oracle.com). Traditionally, once an Item Fulfillment is created with an integrated carrier (FedEx, UPS, USPS, etc.), the carrier's **tracking number** is saved in NetSuite and can be communicated to

customers (Source: erppeers.com)(Source: erppeers.com). However, **real-time tracking status updates** (in-transit, delivered, exceptions, etc.) are not natively pushed into NetSuite – users would typically click the tracking link or manually check the carrier's site for updates. This means out-of-the-box NetSuite offers *visibility of tracking numbers* but **limited visibility of shipment status** after fulfillment.

Modern supply chain operations demand more proactive tracking. Customers expect to know where their order is at all times, and businesses want to **monitor shipments in-transit, get delivery confirmations, and handle delays or exceptions promptly** (Source: appsrhino.com)(Source: appsrhino.com). To address this, NetSuite developers and IT managers often implement ** real-time shipment tracking integration**. This involves capturing tracking events (like "Out for Delivery" or "Delivered") as soon as they occur and reflecting them in NetSuite records or dashboards. Achieving this requires custom integration because NetSuite does not have native webhook support for inbound events (Source: rollout.com). Instead, developers leverage ** carrier webhooks** – push notifications from carriers or third-party tracking platforms – to update NetSuite in real time.

Benefits of Real-Time Tracking via Webhooks

Real-time tracking using webhooks provides significant benefits over manual or batch tracking updates:

- **Immediate Status Updates:** Webhooks deliver data **instantly when an event occurs**, without waiting for a scheduled job or user query (Source: aftership.com)(Source: aftership.com). This means NetSuite can reflect a delivery or delay within seconds or minutes of the carrier reporting it, enabling rapid response. In contrast, using a tracking API without webhooks requires constant polling to "ask" for updates, which is slower and inefficient (Source: aftership.com).

- **Reduced Manual Effort and WISMO Inquiries:** Automated tracking updates in NetSuite reduce the need for staff to manually check multiple carrier websites or import status files. Customers and customer service reps get timely information, cutting down "Where Is My Order?" calls (Source: appsrhino.com)(Source: appsrhino.com). Fewer manual checks also mean fewer errors and labor savings.

- **Operational Efficiency:** Webhooks **eliminate frequent polling** of carrier APIs, saving API calls and network overhead (Source: stackoverflow.com). This event-driven approach is lighter on resources – the system only processes updates when there's an actual change, rather than

running cron jobs that often find no new information. According to an EasyPost engineer, using webhooks can simplify code and reduce bandwidth compared to continuous polling (Source: stackoverflow.com).

- **Proactive Exception Handling:** With real-time feeds, supply chain teams can be immediately notified in NetSuite of exceptions like *delivery exceptions, delays, address issues, or lost packages*. This allows proactive outreach – for example, automatically emailing a customer about a delay or initiating a reshipment process when a problem is detected, rather than discovering it after the fact. FedEx notes that **reliable event-driven updates, including exceptions and delays, let you optimize logistics and customer communication** (Source: developer.fedex.com)(Source: developer.fedex.com).

- **Enhanced Customer Experience:** By integrating tracking events into NetSuite (which can then trigger customer notifications or update customer portals), companies can provide an Amazon-like tracking experience. Customers receive **proactive notifications** (e.g., "Your order is out for delivery" or "Delivered at front door") without needing to track manually. FedEx's Advanced Tracking service emphasizes improving customer experience with proactive, branded notifications for status changes (Source: developer.fedex.com)(Source: developer.fedex.com).

- **Visibility and Analytics:** Having all tracking data in NetSuite in real time means better visibility for account managers and planners. They can see at a glance which shipments are in transit, delayed, or delivered. Over time, this data can be used for analytics – e.g., measuring carrier performance (on-time delivery rates, transit times) or identifying frequent exception causes. Some platforms even offer analytics on delivery events to optimize routes and carrier choices (Source: appsrhino.com).

In summary, **real-time tracking via webhooks transforms NetSuite from a passive system of record into an active tracking hub**, improving responsiveness and decision-making across fulfillment and customer service.

How Carrier Webhooks Work (Major Carriers and Aggregators)

Webhooks are essentially HTTP callbacks – the carrier's system sends an HTTP POST message to a URL you provide whenever a tracking event occurs (Source: aftership.com). Many major shipping carriers and tracking service providers now support webhooks or push notifications for tracking events:

- **FedEx:** FedEx offers an advanced service called **Advanced Integrated Visibility (AIV)** with webhook capabilities. FedEx AIV provides **near real-time tracking push notifications directly to your servers** (Source: developer.fedex.com). With the FedEx **Tracking Webhook APIs**, you can subscribe to updates either by account or by individual tracking numbers. For example, FedEx's *Tracking Account Number Webhook API* allows you to register your FedEx account so that **any shipment associated with that account (outbound, inbound, or third-party) will trigger a webhook event to your URL** (Source: developer.fedex.com)(Source: developer.fedex.com). This means if you ship with FedEx, you can get status updates (picked up, in transit, delayed, delivered, etc.) pushed to NetSuite for all those shipments. FedEx's webhook service can also include enhanced data like estimated delivery time updates and even **Picture Proof of Delivery** for delivered packages (with their premium features) (Source: developer.fedex.com)(Source: developer.fedex.com)</current_article_content>ments).

- **UPS:** UPS has introduced a **Track Alerts API** that provides push notifications. According to UPS, this API gives *"near real-time event updates"* and **pushes updates to the user as soon as available, with no constant polling required** (Source: developer.ups.com). With UPS Track Alert, the integration flow is: after you create a shipment (or obtain a tracking number), you call UPS's API to subscribe that tracking number along with a webhook URL. UPS will then send HTTP POST events to your URL for the next 14 days whenever there's a new scan or status update for that package (Source: developer.ups.com). (If a package isn't delivered within 14 days, the subscription can be renewed for further updates (Source: developer.ups.com).) UPS's webhook events include details like status type/code (e.g. picked up, in-transit, out-for-delivery, delivered, exception) and timestamps (Source: developer.ups.com)(Source: developer.ups.com). This **Track Alert** service is a paid offering on the UPS developer portal, but it significantly improves timeliness. It essentially turns UPS's tracking API from a pull model into a push model – **you submit up to 100 tracking numbers with a callback URL, and UPS notifies you of events for those shipments in near real time** (Source: developer.ups.com) (Source: developer.ups.com).

- **DHL:** DHL provides webhook capabilities as well, though they vary by DHL division (Express vs. eCommerce). For example, **DHL eCommerce Americas** has a **Tracking Webhook Management API** that *"allows clients to register webhook endpoints for specific accounts and/or tracking numbers"* (Source: developer.dhl.com). DHL's **Shipment Tracking – Unified API (Push)** similarly enables subscribing to push notifications for shipment status across DHL's network (Source: developer.dhl.com). This means you can receive DHL tracking events (like clearance processing, in transit checkpoints, out for delivery, delivered, etc.) automatically. DHL's push API can send updates on routing changes or delivery time updates as well (Source:

developer.dhl.com). So, a NetSuite integration with DHL webhooks would let your system get continuous updates for, say, all packages shipped via DHL under your account or those you specifically subscribe to.

- **USPS:** Historically, USPS provided only pull-based tracking (Track/Confirm APIs), but recent updates indicate USPS now offers **near-real-time push notifications via webhooks** for tracking events (Source: postalpro.usps.com). As of mid-2024, the USPS API platform can send webhook notifications for package scans and status changes (Source: postalpro.usps.com). This is a significant improvement – for example, USPS can push events like "In Transit", "Out for Delivery", or "Delivered" to your endpoint, instead of requiring your system to periodically call USPS for updates. (USPS also has a legacy option like email updates, but webhooks provide a more direct integration for systems like NetSuite.)

- **Third-Party Tracking Aggregators:** Instead of integrating each carrier's webhook API separately, many businesses use aggregators such as **AfterShip, EasyPost, TrackingMore, or Shippo**. These services support **hundreds of carriers** and provide a unified tracking API and webhook mechanism (Source: appsrhino.com)(Source: appsrhino.com). For instance, AfterShip allows you to add tracking numbers (with carrier info) via their API, and then they will send webhook callbacks to your URL whenever any carrier updates the status. This is appealing because you integrate once with the aggregator, and it handles subscribing/polling all carriers in the background. EasyPost's Tracking API similarly lets you create a "Tracker" for a shipment, and then you receive webhooks for any status updates for that tracker (Source: easypost.com) (Source: easypost.com). Aggregators often standardize statuses (e.g., all carriers' events mapped to a common set of status codes like "Pre-Transit", "In Transit", "Out for Delivery", "Delivered", "Exception") and provide rich data like last location, signed recipient name for delivery, etc. (Source: easypost.com)(Source: easypost.com). In the context of NetSuite, you could use an aggregator to avoid writing separate logic for UPS, FedEx, DHL, etc. – the aggregator will hit your NetSuite webhook endpoint for any and all carriers' updates.

Reputable integration providers have started offering out-of-the-box solutions for this. For example, AfterShip provides a NetSuite connector that *"gets automated tracking updates for your NetSuite sales orders across 800+ courier services"* (Source: aftership.com). This kind of plugin uses the AfterShip webhook/feed behind the scenes to push tracking status into NetSuite, demonstrating the feasibility of a unified approach.

Overall, carrier webhooks work by **turning the carrier into the initiator** of communication. Rather than NetSuite (or an external script) constantly asking *"Has shipment #123 delivered yet?"*, the carrier's system announces *"Shipment #123 just got delivered"* to NetSuite. Each carrier's

implementation details differ (subscription setup, data format, time limits on subscriptions, etc.), but the core concept is consistent: **event-driven tracking for near real-time visibility** (Source: developer.ups.com)(Source: developer.fedex.com).

Setting Up Carrier Webhook Integrations with NetSuite

Integrating carrier webhooks with NetSuite requires some technical effort, as NetSuite doesn't natively support inbound webhooks. However, it can be achieved using **SuiteCloud customization** or middleware. Below is an outline of how to set up such integrations, with various options:

1. Create a NetSuite Endpoint (Suitelet or RESTlet)

Since NetSuite has no built-in webhook listener, you need to create a custom script that can receive HTTP POST requests from the carrier or aggregator (Source: rollout.com). There are two common script types for this:

- **Suitelet:** A Suitelet is a server-side script in NetSuite that can generate a custom URL (which can be made accessible without login). By marking a Suitelet as "Available Without Login," you essentially create a public-facing URL that external systems (like carriers) can POST to. The Suitelet code can read the request (JSON or XML payload from the carrier) and then perform actions inside NetSuite (e.g., find the corresponding Item Fulfillment record and update its status). Using a Suitelet, you have full control over the request handling and response. This is a straightforward way to accept webhooks directly into NetSuite. **Security note:** If using a no-login Suitelet, include a verification mechanism (like expecting a secret token or verifying an HMAC signature if the carrier provides one) to ensure the request truly comes from the carrier.
- **RESTlet:** A RESTlet is another script type that exposes custom REST endpoints in NetSuite. RESTlets typically require authentication (token-based or OAuth) with NetSuite credentials. This can be tricky for carriers, as you usually can't make them handle OAuth. However, one approach is to create a RESTlet and share credentials or a token with the webhook service (some third-parties allow including HTTP headers for auth in webhook calls, but many carrier webhooks won't support complex auth). If you cannot have the carrier sign in, a RESTlet might only work if you allow anonymous access (which is not typical) or if you use a middleware to relay (discussed below).

In either case, the script will parse the incoming data. For example, UPS might send a JSON payload with the tracking number and new status – your Suitelet/RESTlet would extract those, then perform a search in NetSuite for the record with that tracking number (e.g., find the Item Fulfillment or custom "Shipment" record). Once identified, you could update fields: perhaps set a custom "Shipment Status" field to "Delivered" or log the timestamp, etc., and save the record. You might also write the event to a custom child record (to keep a history of tracking events). Finally, the script should return an HTTP 200 OK response quickly to acknowledge receipt (more on this in *Handling Events* below).

Important: NetSuite **requires authentication for inbound requests by default**, which is a challenge for webhooks (Source: rollout.com). Making a Suitelet available without login bypasses that, but you must secure it by other means (e.g., secret token in URL or request). Alternatively, using a RESTlet would require a middleware that adds NS authentication.

2. Using Middleware or iPaaS Solutions

A common architecture is to **decouple the webhook endpoint from NetSuite** for security and reliability. In this approach, you use a middleware server or an Integration Platform as a Service (**iPaaS**) tool to handle the webhook, then forward the data into NetSuite via its standard APIs (SuiteTalk or REST web services).

- **Middleware Server:** You can set up a lightweight web service (for example, an AWS Lambda, Azure Function, or a simple Node/Express app) that is publicly accessible. This endpoint receives the carrier's webhook call. Because you control this server, you can implement any necessary security (e.g., verifying a signature, or using an API Gateway with authentication). Once the middleware validates the message, it uses the **NetSuite API** to update records. NetSuite offers SOAP-based APIs (SuiteTalk) and REST-based APIs. For instance, the middleware could call NetSuite's REST API (2024.x version or SuiteTalk REST) to find and update the record with the given tracking number. Since the middleware is a full environment, it can store NetSuite credentials or tokens securely and handle the authentication that the carrier cannot. This approach aligns with the recommendation that *"one approach to handle authentication is to implement a middleware service that receives the webhook and then makes an authenticated call to NetSuite"* (Source: rollout.com). It adds a bit of complexity (you need to maintain the middleware), but improves security and allows more flexible processing (like reformatting data, queuing, etc.).

- **iPaaS / Integration Tools:** There are many NetSuite-focused integration platforms (Celigo integrator.io, Boomi, Jitterbit, MuleSoft, Workato, etc.) that can facilitate this. Some of these platforms let you create a **Webhook listener** as a trigger for a flow. For example, Celigo's integrator.io allows setting up a webhook URL (hosted by Celigo) that can trigger a flow when hit (Source: docs.celigo.com)(Source: celigo.com). You could configure a flow such that when the carrier calls that URL, the flow looks up the appropriate NetSuite record and updates it via a NetSuite connector. The iPaaS usually handles authentication to NetSuite for you (you store your NetSuite token or credentials in the connection). These tools often come with pre-built templates; for instance, Celigo lists **AfterShip integrations** and could manage "real-time tracking updates to customers" by connecting AfterShip's webhook to NetSuite's sales order or fulfillment records (Source: celigo.com). The benefit of iPaaS is reduced coding – you mostly configure mapping and let the platform handle reliability (it can queue events, retry on failure, and provide monitoring dashboards). The downside can be cost and reliance on a third-party platform, but for many organizations the ease-of-use is worth it.

Regardless of direct SuiteScript or middleware, **setting up the carrier side** is a crucial step. For each carrier or aggregator you use, you must configure your webhook URL and subscriptions:

- *For direct carrier APIs:* This often means going to the carrier's developer portal, enabling the tracking webhook service, and providing your endpoint URL (plus any verification tokens). For example, with UPS Track Alert, after getting API access, your system needs to **call UPS's subscription API** whenever you have new tracking numbers, as mentioned, sending your callback URL and the tracking IDs (Source: developer.ups.com). FedEx's portal would have you set up a "webhook project" with your URL and link it to your FedEx account number (Source: developer.fedex.com)(Source: developer.fedex.com) – then you might not need to call per tracking (FedEx can send for all shipments on that account automatically). DHL's API might require registering each tracking number or account as well. This initial setup is usually done in a script right after order fulfillment creation in NetSuite: e.g., a User Event script on Item Fulfillment "after submit" could invoke a RESTlet or external call to subscribe the shipment with the carrier API.

- *For aggregators:* Typically, you register a single webhook endpoint once in the aggregator's admin dashboard or API. For example, with AfterShip you would configure the webhook callback URL in your AfterShip account settings. AfterShip then calls that URL for every update on any tracking you're monitoring. You still need to send the tracking numbers to the aggregator (e.g., via their API or sometimes they have a NetSuite app that syncs new fulfillments to AfterShip). In practice, the **AfterShip NetSuite app** likely works by a scheduled sync or triggered sync that

sends new NetSuite shipments to AfterShip, and then AfterShip pushes updates back via the configured webhook (Source: aftership.com). EasyPost's API approach would have you create a `Tracker` object via API for each shipment (which can be automated with SuiteScript or middleware when a tracking number is generated), and in that call you specify the webhook URL; after that, EasyPost will POST events for that tracker automatically (Source: developer.ups.com)(Source: stackoverflow.com).

3. SuiteTalk and SuiteScript (Alternative Data Flows)

It's worth noting that some integrations might not push data into the core Item Fulfillment record, but instead use **custom records** or **SuiteTalk asynchronous flows**:

- One pattern is to create a **custom "Shipment Tracking" record** in NetSuite which stores the tracking number, carrier, latest status, etc., linked to the fulfillment. Each webhook event could insert a new entry or update a record. This avoids directly altering the fulfillment record outside of normal user operations, and provides an audit log of events. A scheduled script could then reconcile final statuses or send notifications if needed.

- Another approach is that rather than the carrier calling NetSuite directly, the carrier calls a middleware which **queues events** and then a NetSuite Scheduled Script periodically pulls from that queue. This is a hybrid of push and pull: you use webhooks to gather data quickly, but insert it to NetSuite on a schedule to avoid too many small transactions. However, with the performance of SuiteCloud increasing, many opt to update in real time if volume isn't extremely high.

NetSuite SuiteTalk (SOAP or REST) APIs are the method by which external systems update NetSuite if not using SuiteScript endpoints. For example, a middleware might use SuiteTalk SOAP to `update` the Item Fulfillment record's custom fields (requiring the internal ID of the record, which could be looked up by a saved search on tracking number). Or, using the 2023+ REST web services, the middleware could do a PATCH to the record if the API supports fulfillment records. In any case, ensure that the integration user or role has permission to edit fulfillment records or the custom tracking records you design.

Finally, don't forget to **enable any necessary features** in NetSuite: if using SuiteScript, the SuiteCloud features must be enabled (which they typically are if you're doing development). If using token-based authentication for SuiteTalk, ensure to create an integration record and role with needed permissions.

Handling Webhook Events: Updates, Delays, and Delivery Confirmation

Once the integration is set up, NetSuite will start receiving **event payloads** from carriers or aggregators. Handling these events properly is critical:

- **Parsing the Payload:** Webhook payloads are often in JSON (sometimes XML for certain older carrier APIs). The data typically includes at least a tracking number and a status update. It may also include a timestamp of the event, location details (city, state of scan), a description (e.g., "Delivered, left at front door"), and possibly a code for the event type. For example, EasyPost's webhook JSON (for a delivered event) contains a `description` like `"tracker.updated"` and an embedded `result` object with details: status `"delivered"`, `signed_by` (if applicable), and an array of `tracking_details` with the history of scans (Source: easypost.com)(Source: easypost.com). Your script should extract the relevant parts – usually the *current status* and maybe the current location and timestamp. If the payload doesn't explicitly say "delivered" or "out_for_delivery" in a standardized way, you might need to map carrier-specific codes. (UPS, for instance, might send status type "D" for delivery events (Source: developer.ups.com), whereas FedEx might send "DL" or a literal description like "Delivered").

- **Updating NetSuite Records:** With the parsed data, determine which record to update. If you have the NetSuite record ID stored somewhere (maybe you included it when subscribing, or use the tracking number as a key), use that to load the record. Otherwise, you may search for the tracking number. NetSuite's Item Fulfillment has a Tracking Number field (and a Packages sublist for multiple packages). Often, the simplest approach is to create a custom field on the Item Fulfillment or Sales Order, e.g., *"Latest Tracking Status"* and *"Last Tracking Update Time"*. Your integration can then set "Latest Tracking Status" to "Out for Delivery" or "Delivered" etc., and maybe fill a "Delivered Date" field when delivered. If you have multiple partial shipments per order, it might be better to track status per tracking number in a subrecord.

  In more advanced setups, you might create **a child custom record for tracking events**, e.g., "Tracking Event" with fields: Tracking Number, Status, Location, Timestamp, and link to the Fulfillment. Each webhook event inserts a new Tracking Event record. This provides a detailed timeline visible within NetSuite for each shipment. It's excellent for audits – you can see every scan event in NetSuite if needed.

- **Business Workflow on Events:** Once NetSuite is updated, you can leverage that data. For example:

- When a *Delivered* status is received, you might automatically complete the Sales Order or send a confirmation email to the customer (if not already sent). NetSuite could trigger an email alert via SuiteFlow or a small SuiteScript, using a template that thanks the customer and possibly asks for feedback now that the item arrived.
- If a *Delayed* or *Exception* status comes in (e.g., "Delivery Exception – Recipient not available" or "Package delayed due to weather"), you might create a Case or Task for the customer service team to proactively follow up. Alternatively, an email could be sent to the customer apologizing and informing them of the delay.
- If an *Out for Delivery* event is received, some businesses send a notification ("Your order is out for delivery today!"). This could be triggered from NetSuite using an email or even an SMS if integrated with a communication tool.
- For *Failed Delivery Attempt* events, you might want to notify the customer to contact the carrier or confirm address.

These are all optional workflows, but real-time data enables them. The key is storing the event info in NetSuite in a way that your saved searches or workflows can act on it. Many companies start with just updating the records for internal visibility, then later add customer-facing triggers once the data is trusted.

- **Multiple Package Handling:** If your orders can ship in multiple packages, and thus have multiple tracking numbers under one fulfillment, you need to handle that logic. NetSuite's standard fulfillment record can hold multiple tracking numbers (each associated with a package sub-record). You'll want to map incoming events to the correct package. One approach: maintain a mapping of tracking number to fulfillment **and** package index. If using a custom tracking record, you'd likely have one per tracking number anyway. When all tracking numbers under a fulfillment have a "Delivered" event, you might mark the whole fulfillment as delivered or complete. Until then, it might be partially delivered.
- **Delivery Confirmation in NetSuite:** Often the final event is "Delivered". Many organizations will use that to update a **Delivered Date** or a status flag on the fulfillment. If NetSuite's fulfillment record status doesn't directly support a "delivered" concept (fulfillments are typically just marked shipped when created), you can use a custom checkbox "Delivery Confirmed" that gets checked when delivered. This can drive reports of shipped vs delivered. It can also be used to automatically trigger invoicing: for example, some businesses only invoice once delivery is

confirmed (especially if they want to ensure the customer got the goods). With real-time data, you could automate that: when Delivered arrives, a script creates the Invoice for that Sales Order.

- **Handling Delays and Exceptions:** Carriers will signal exceptions (like "Exception: Address not found" or "Held at customs"). Deciding what to do for each event type is part of the integration planning. At minimum, log them in NetSuite so users can see. For critical exceptions, consider creating a **Notification**. This could be an email alert to the fulfillment manager or a dashboard portlet highlighting "Shipments with exceptions". The value is that the team can react same-day rather than after a customer complains. FedEx's service highlights that exceptions and delay event notifications are part of their push package to help businesses respond in real time (Source: developer.fedex.com).

- **Acknowledging the Webhook:** Technically, when NetSuite (or your middleware) finishes processing the event, it should return an HTTP 200 status promptly. Many webhook providers expect a quick response. UPS, for example, notes that the webhook endpoint should handle and respond to events **within milliseconds** for optimal performance (Source: developer.ups.com). In practice, you should aim to respond within a few seconds at most. If NetSuite processing (search and update) is slow, consider decoupling by quickly queuing the task and responding immediately, then doing the actual update in a scheduled manner. FedEx's documentation mentions they will retry events up to 3 times if they don't receive a successful acknowledgement (Source: developer.fedex.com). So ensure your integration catches and returns a success code (and only returns an error if you want a retry).

Security Considerations for Webhook Integrations

Security is paramount because exposing an endpoint that can modify NetSuite carries risk. Here are key considerations:

- **Authentication & Verification:** As discussed, NetSuite endpoints typically require authentication. If you open a Suitelet to the public, you **must** verify that incoming requests are from trusted sources. Use a shared secret or token. Many carriers or aggregators allow you to set a secret that they will include in the webhook (either as a header or part of the URL) – your code can check this matches a expected value. Some services (like Stripe or Twilio in other domains) sign their webhooks with an HMAC using a secret; if a carrier provides such a signature (e.g., in a header), implement the signature verification logic in SuiteScript to ensure it's legitimate. If the verification fails, do not process the data.

- **TLS/SSL:** Ensure your endpoint URL is HTTPS. Most carrier systems will require an HTTPS URL anyway. NetSuite's domains are HTTPS by default, so a Suitelet published on NetSuite will be secure (NetSuite provides a *.netsuite.com URL with SSL). If using middleware, use a TLS-enabled endpoint. This protects the data in transit (tracking info, etc.) from interception.

- **Principle of Least Privilege:** The NetSuite script or integration user that processes the webhook should have minimal permissions – just enough to find and update the necessary records. Avoid using an administrator role. If using token auth, create a role that only has access to Item Fulfillment (or custom tracking record) and perhaps Sales Order fields that need updating. This limits potential damage if credentials somehow leak.

- **Data Sanitization:** Although it's unlikely an attacker can spoof a carrier easily (especially if you verify secrets), always program defensively. Parse JSON carefully and avoid eval or insecure parsing. Ensure that the data from the webhook (which might include text like city names or status descriptions) is stored or logged safely to prevent injection attacks on any subsequent process.

- **Firewall and IP Whitelisting:** If possible, restrict who can call your webhook URL. NetSuite itself can't easily restrict incoming connections by IP, but your middleware could. Some carriers publish IP ranges for their webhook servers – if feasible, configure your firewall to only accept traffic from those ranges. This is an extra layer of protection against random bad actors hitting your endpoint.

- **Timeouts and Retries:** As noted, carriers will retry if your endpoint is unresponsive. Design your integration idempotently – if the same "Delivered" event comes twice (due to a retry or duplicate), your code should handle it gracefully (e.g., check if you already marked that shipment delivered, and if so, ignore or just confirm). This prevents issues like double-notifying customers.

- **Logging and Alerts:** From a security standpoint, monitor your webhook for abnormal activity. If suddenly you receive a burst of events that doesn't match your shipments volume, it could indicate a problem (or simply a backlog clearing). Also consider logging authentication failures (if someone hits the endpoint with wrong secret, log and perhaps alert).

- **NetSuite Governance Limits:** One could consider this a security or a reliability concern – NetSuite scripts have governance (execution time and usage limits). If a webhook bombards your account with thousands of events (e.g., a carrier glitch sending repeats or if you bulk subscribe many old shipments), you need to ensure your integration can handle it without

exhausting governance. You might incorporate a queue or scheduling if volumes are high, to avoid a single script run hitting limits. Using SuiteCloud Plus or async script queues might be needed at enterprise scale.

In essence, treat the webhook integration as you would any external integration: secure the entry point, authenticate the source, and validate all inputs. By doing so, you can safely let carriers drive updates into your ERP without opening a backdoor.

Monitoring, Logging, and Error Handling

With a real-time integration in place, having proper monitoring and logging will help maintain it and troubleshoot issues:

- **Event Logging:** It's wise to log each webhook event receipt and the outcome. If using a Suitelet, you can utilize the NetSuite script log (nlapiLogExecution or console.log in SuiteScript 2.x) to record a summary like "Received DHL update for tracking 12345: status=Delivered". However, script logs in NetSuite have retention limits and can be hard to search en masse. A better approach is to create a **custom record** "Webhook Event Log" with fields: Tracking Number, Carrier, Event Type, Timestamp received, Processing result (success/failure), and maybe the raw payload (or part of it). Each time you get a webhook, create an entry. This provides a searchable history in NetSuite of all events. It's incredibly useful if, say, a customer claims they didn't get delivered – you can check the log and see that you got a "Delivered" event at a specific time, etc. It also helps if something failed – you might log an event with status "error: could not find matching record" for example.

- **Alerting on Failures:** If a webhook event fails to process (e.g., your code couldn't find the tracking number in NetSuite, or a record update threw an error), you should capture that and alert someone. The alert could be an email to an admin or an entry on a dashboard. NetSuite's SuiteScript can send emails via `nlapiSendEmail` or in 2.x using email module. For example, if a FedEx webhook comes in for a tracking number that NetSuite doesn't recognize (maybe a package that was manually shipped outside of system), you might notify the logistics coordinator that "FedEx reported an update for unknown shipment 9999." This helps identify integration gaps.

- **Retry Strategy:** As mentioned, carriers themselves retry a few times if they get no response. But what if your endpoint *did* receive the event but failed halfway through processing? In that case, the carrier thinks it succeeded (you returned 200 OK), but NetSuite didn't get updated

fully. To handle this, build idempotency and perhaps scheduled reconciliation. For idempotency: design the update to be safe if repeated (e.g., update "Latest Status" field regardless, or if the same status is already recorded, it's fine). For reconciliation: you could schedule a daily script that finds any shipments that are marked shipped but not delivered past a certain date and call the carrier API for a status (a fallback poll). This can catch any events that slipped through (though with robust webhooks this is rare). FedEx provides a "Retry API" for missed events (Source: developer.fedex.com) – possibly allowing you to query events if you think you missed something.

- **Dashboard & KPIs:** Consider adding a **tracking dashboard** in NetSuite for your operations team. This could be a saved search or SuiteAnalytics Workbook that lists all in-transit shipments with their last status and last update time. It could highlight any that have an exception or have had no update for an unusually long time. This acts as a monitoring tool – if a package has no movement for 10 days, maybe it's lost and needs investigation. Since you're updating statuses in near real-time, these reports become very useful for managing carrier performance and customer expectations.

- **Performance Monitoring:** If using an external middleware, monitor that service's performance. Many integration platforms have an interface showing success/failure counts and timing. If using SuiteScript only, monitor your script usage and execution time. A sudden spike in events might require scaling (if volume grows, maybe split the work or use SuiteCloud Plus to run scripts in parallel).

- **Testing and Sandbox:** Ensure you test the webhook integration in a NetSuite Sandbox or Release Preview environment if possible. Carriers often have a test mode or sandbox as well (e.g., FedEx and UPS provide test tracking numbers and environments). Test how your system handles multiple rapid events, out-of-order events (sometimes a delayed "out for delivery" might arrive after a "delivered" in rare cases), and erroneous data. Monitoring during testing will give confidence for production.

- **Logging Payload for Debug:** In early stages, you might log full JSON payloads to a secure storage (perhaps not NetSuite if large). This can help debug mapping issues. Be careful to remove or mask any sensitive info (though tracking data is usually not highly sensitive beyond maybe customer address, which NetSuite already has).

In summary, treat the webhook integration as a critical process: keep an eye on it. With good logging and monitoring, you will catch if, say, FedEx changed a field format and your parser started failing, or if your endpoint went down and no events have been received (one cue could be "no events in the log for 2 hours" when normally you get dozens – that should trigger an investigation).

NetSuite doesn't natively show an error if a Suitelet fails (since it's an inbound request), so the onus is on your team to monitor your solution. The investment in good logging and alerting will pay off by preventing silent failures.

Case Studies and Sample Use Cases

Real-time tracking via webhooks in NetSuite is increasingly adopted by companies aiming for tight integration between operations and customer service. Here are a few illustrative use cases:

- **eCommerce Retailer:** Consider an online retail brand fulfilling orders from its warehouse through FedEx. Without webhooks, their staff or customers would check tracking manually. After implementing FedEx AIV webhooks into NetSuite, the **NetSuite dashboard shows live status for each package**. For instance, an order's Item Fulfillment record now updates to "Out for Delivery" the morning the FedEx truck is out, and "Delivered" with a timestamp once completed. The system emails the customer "Your package was delivered at 3:45 PM" automatically. This retailer saw reduced support tickets since customers got timely updates and could even self-serve by checking their order status on the company's NetSuite Customer Center, which was updated in real time. As ERP consultants noted in a FedEx integration scenario, this kind of setup *"provides real-time visibility into your shipping operations directly from NetSuite"* (Source: erppeers.com) and was especially valuable during peak season: high-volume fulfillment during holidays was manageable because tracking updates and customer notifications were automated, even as hundreds of shipments went out daily (Source: erppeers.com).

- **Wholesale Distributor:** A B2B distributor shipping pallets and packages via multiple carriers integrated a third-party aggregator (AfterShip) with NetSuite. As shipments leave via UPS Freight and DHL, tracking events from both carriers funnel into NetSuite through AfterShip's unified webhook. The distributor created a **custom "Shipment Status" field on their transfer orders** in NetSuite. When a shipment is delayed or an exception occurs (e.g., freight shipment delayed due to weather), the system flags that transfer order record in red and emails the logistics team. One notable outcome was improved inventory planning: if an incoming inventory transfer was marked "Delayed" in NetSuite, purchasing could proactively adjust production schedules. The integration provider's case study highlighted that *streamlining tracking and getting real-time updates ensured timely deliveries and enhanced operational efficiency* (Source: celigo.com) for the distributor's supply chain.

- **3PL (Third-Party Logistics) Provider:** A 3PL managing fulfillment for multiple client companies used NetSuite and offered a portal for its clients. By integrating carrier webhooks (UPS, USPS, FedEx) into NetSuite, the 3PL was able to present each client with up-to-date tracking statuses on all their orders via Suitelet portal pages. This eliminated the clients' need to track shipments themselves. The 3PL configured **NetSuite SuiteAnalytics reports** that automatically show "Shipments Delivered Today" and "Shipments Delayed/Exception" for each client, updated continuously. This value-add service was enabled by the webhook integration and helped the 3PL differentiate itself. Internally, the 3PL also used saved searches to monitor any shipments that have been in transit beyond a threshold (e.g., >10 days) to investigate with carriers.

- **Comparison with Email-Based Notifications:** As a contrast, consider that before webhooks, some companies used email notifications from carriers (like UPS Quantum View emails) that were sent to a mailbox, then parsed by scripts (Source: stackoverflow.com). Shopify is cited as having historically used this method to trigger "order delivered" emails (Source: stackoverflow.com). One company attempted this with NetSuite – they had a scheduled script read an Outlook mailbox for "Your package has been delivered" emails. This was clunky and prone to errors when email formats changed. Transitioning to direct webhooks eliminated this unreliable middle step, demonstrating the superiority of direct integration.

These examples underscore common benefits: **improved customer satisfaction, fewer manual tasks, and better internal visibility**. A key theme is that integrating via webhooks turns NetSuite into a more real-time system for physical operations, which many NetSuite customers (who use it for order fulfillment) found to be a substantial upgrade over daily batch updates.

Common Challenges and Solutions

Implementing real-time tracking in NetSuite via webhooks can come with challenges. Here are common ones and how to address them:

- **NetSuite's Lack of Native Webhooks:** By far the first challenge is that NetSuite wasn't built with webhook endpoints. As noted, **custom endpoints must be built** (Source: rollout.com). The solution is to leverage SuiteScript or middleware as described. Developers sometimes are concerned about making a Suitelet publicly accessible. The solution is to secure it with tokens and also consider using external platforms where possible for an extra layer. Utilizing a robust iPaaS that specifically mentions webhook support with NetSuite (like Celigo or others) can accelerate the build if you're not comfortable writing the entire stack.

- **Authentication and Carrier Restrictions:** Some carriers (especially older ones) might not support including custom headers or tokens in webhook requests. For example, a carrier might only allow a URL and sends a generic POST without any auth. In such cases, security relies on obscurity (the URL is a long random string) plus request validation like IP whitelisting or content validation. If this is a concern, the **middleware approach is often the safest** (Source: rollout.com) – keep the endpoint off of NetSuite and in a place where you have more control. Another solution used is to implement a **token in the URL path** (e.g., https://rest.netsuite.com/app/site/hosting/restlet.nl?script=123&deploy=1&token=ABC123XYZ) where `token` is a secret that the script checks. As long as the URL isn't guessable, and it's over HTTPS, this is fairly secure.

- **Data Mapping and Consistency:** Each carrier has its own terminology and event structure. Mapping these into a uniform status in NetSuite can be tricky. A solution is to define a **standard status list in NetSuite** (e.g., Pending Pickup, In Transit, Out for Delivery, Delivered, Exception, Attempted) and then map carrier events to these. You might have a mapping table or logic in code (for example, UPS "D" and FedEx "DL" both map to Delivered). Testing is required to ensure every relevant event is caught. Using an aggregator like AfterShip simplifies this, because they provide standardized statuses (Source: aftership.com) – you then just use their status field.

- **Volume and Rate Limits:** If your business ships a very large number of packages, the sheer volume of webhook calls could be high. NetSuite Suitelets can handle a moderate load, but you might hit script governance limits if hundreds of events pour in simultaneously (e.g., after a system downtime or every morning when overnight deliveries all get delivered around the same time). Solutions include:

  - Queueing events (e.g., middleware writes to an AWS SQS queue or NetSuite's queue record) and processing them in batches.

  - Implementing **SuiteCloud Plus**, which gives additional concurrency for workflows and scripts, so multiple events can be processed in parallel by multiple script deployments.

  - Throttling subscriptions: if absolutely needed, you could choose not to subscribe to every single checkpoint (maybe subscribe only to delivered and exception events), though most prefer full visibility.

  - Also ensure the external side is not sending more than necessary – some carriers allow filtering event types. UPS's Track Alert, for instance, may send all intermediate scans; if you only care about delivered or exception, you might filter in your code or perhaps choose to

ignore certain events to reduce noise.

- **Order of Events and Idempotency:** Webhooks might not always arrive in the exact sequence of occurrence, or you might get duplicate sends. This can confuse a naive implementation (e.g., marking delivered, then getting another in-transit scan late). The solution is to design idempotently and handle out-of-order:
  - Maintain a small state model: e.g., have a field for "Delivered flag". If an event comes in marking delivered, set that and perhaps ignore any further non-delivery updates. Alternatively, record the last known status timestamp and don't overwrite a status with an older timestamp event.
  - In practice, out-of-order is rare for final statuses, but duplicates can happen. Always check if the status is already what the event says before updating (though even if you update again, it's fine; just don't trigger duplicate customer emails in that case).
- **Testing and Carrier Support:** Each carrier's webhook API might have nuances. A challenge is **testing end-to-end** because you often need real tracking numbers. You can mitigate this by using test tracking numbers provided by carriers (UPS and FedEx have dummy numbers that simulate events in their test environment). Ensure to test not just the "happy path" (normal delivery) but exceptions like a delivery attempt failure. Work closely with carrier support or aggregator support if things aren't working – sometimes webhook activations on their side require certain account settings.
- **Comparing Webhooks vs. Polling (Organizational Buy-in):** Occasionally, you need to justify this project to management, who might ask "Why not just check tracking once a day?". The challenge is demonstrating the ROI of real-time. Here, present the **comparison**: Polling (or manual checks) means delays in info, potential to miss critical events, and higher workload. Webhooks provide immediacy and labor savings. Technically, polling many shipments via API can also run into rate limits, whereas webhooks push only changes. In fact, as a FedEx rep might note, with webhooks you *"get near real-time track updates… without having to poll"*, improving workflows and customer comms (Source: developer.fedex.com)(Source: developer.fedex.com). If needed, implement a small pilot on one carrier to measure the impact (e.g., track reduction in customer inquiries or faster issue resolution) to validate the approach.
- **Handling Legacy Processes:** Some companies might have existing shipping scripts that print labels and fulfill orders (like using the old Shipping Label Integration or third-party systems). Integrating tracking status may require modifying those processes to capture necessary info

(like capturing the internal ID when sending a subscription). Coordination between the team managing shipments and the team managing the ERP integration is key. The solution is to incorporate the webhook subscription call at the point of label creation or fulfillment creation.

By anticipating these challenges and addressing them with the solutions above, one can implement a **robust, real-time tracking integration**. As one Stack Overflow expert pointed out, "real-time" will always have some small delay (when the carrier posts an update to when you receive it), but it's vastly superior to periodic polling (Source: stackoverflow.com). Ultimately, the result is a NetSuite system that stays in sync with the physical world of package movement, enhancing both operational control and customer satisfaction.

Webhooks vs Other Approaches: A Comparison

To appreciate the value of carrier webhooks, it helps to compare with traditional tracking update methods:

Illustration: Traditional API Polling Workflow — Your system must repeatedly request tracking info from carriers, processing responses each time (Source: aftership.com). This continues until a change is found (e.g., status update), leading to inefficiency.

In a **Polling model**, as shown above, NetSuite (or a middleware script) would periodically call each carrier's tracking API for each in-transit package. For example, every 6 hours, ask "Has package 1Z999 delivered? Has package 1Z1000 delivered?" etc., and then update records. The downside is obvious: most of those calls return "no change" until a delivery or exception occurs. It's a waste of resources and not truly real-time (the update could be up to 6 hours late depending on schedule). Polling needs careful scheduling to balance timeliness vs. API limits – polling too often can hit carrier rate limits or burden your system; polling too infrequently delays critical updates. On the plus side, polling logic is simple to implement and works with any carrier API without needing the carrier to support webhooks. Historically, this was the primary method before webhooks were offered.

Illustration: Webhook Workflow — The carrier or aggregator pushes tracking updates to your NetSuite endpoint in real time (Source: aftership.com). No continuous asking — updates are received as events, drastically reducing latency and overhead.

In a **Webhook model**, as depicted above, you register the shipment or account one time, and then simply **wait for incoming calls**. The carrier does the work to notify you. This is event-driven and much more efficient. There's no need to constantly request data, and your system only works when

there is an actual update (Source: aftership.com)(Source: aftership.com). The result is timely updates (often within moments of the carrier scan) and reduced complexity in code (no loop of calls and comparisons, just a handler for events).

To summarize the comparison, the table below highlights differences:

APPROACH	HOW IT WORKS	PROS	CONS
Real-Time Webhooks	Carrier sends HTTP POST to NetSuite/middleware on new events (push).	- Near-instant updates (event-driven)	
- No polling overhead			
- Scales efficiently with event volume (Source: developer.ups.com)	- Setup complexity (webhook endpoint, security)		
- Requires carrier support or aggregator service			
- Handling authentication challenges in NetSuite (Source: rollout.com)			
API Polling	NetSuite or script periodically calls carrier API for status (pull).	- Simpler to implement (straight API calls)	
- Works with any carrier that has an API (no special support needed)	- Delayed information (depending on interval)		
- High number of API calls (inefficient) (Source: aftership.com)			
- Possible rate limit issues and missed fast changes			
Manual Updates	Human staff or customers check carrier site, then update NetSuite or inform customers.	- No development needed (short-term)	

APPROACH	HOW IT WORKS	PROS	CONS
- Uses carriers' native tools (emails, websites)	- Labor-intensive and error-prone		
- Not scalable for volume shipments			
- Customers get inconsistent info, slower updates (poor experience)			

In practice, manual updates are untenable for any sizable operation – they lead to delays and frustration. Polling was a necessary workaround and is still used as a backup or for carriers without webhook support. But as carriers modernize (UPS and USPS have clearly moved towards push models (Source: developer.ups.com)(Source: postalpro.usps.com)), webhooks represent the state-of-the-art approach. Companies leveraging webhooks have a competitive edge in providing information transparency. As one source humorously put it, webhooks are like *"the express delivery system of the API world"*, whereas polling is like continuously refreshing a page to see if there's news (Source: medium.com).

By choosing webhooks for shipment tracking in NetSuite, businesses align their systems with real-time operations and customer expectations in the modern e-commerce and logistics environment.

Conclusion

Real-time shipment tracking in NetSuite via carrier webhooks bridges the gap between your ERP and the movement of goods in the real world. Implementing this integration provides **immediate, actionable visibility**: NetSuite can automatically reflect a package's journey from warehouse to doorstep. For NetSuite developers and supply chain IT managers, the effort to set up webhooks – through SuiteScript endpoints, SuiteTalk calls, or middleware – is rewarded with streamlined operations and improved service levels.

In this report, we covered how NetSuite's standard shipping features can be extended from just printing labels and storing tracking numbers (Source: erppeers.com)(Source: erppeers.com) to actively tracking shipments post-fulfillment. Using carrier webhooks (FedEx's AIV, UPS's Track Alerts, DHL's push API, USPS's webhook support, or aggregators like AfterShip/EasyPost) allows NetSuite to receive near real-time updates on shipment status without polling (Source:

developer.ups.com)(Source: aftership.com). We outlined technical integration options, from Suitelets/RESTlets in NetSuite (with appropriate security) (Source: rollout.com) to external middleware and iPaaS solutions that can manage the heavy lifting (Source: rollout.com). We discussed handling the incoming events – updating records, triggering workflows on delivery or delay, and ensuring acknowledgment and reliability (with retries and logging) (Source: developer.fedex.com)(Source: developer.ups.com).

Security and monitoring considerations were emphasized to maintain a robust and secure interface between carriers and NetSuite. Real examples illustrated that businesses employing these integrations have seen reductions in customer inquiries, faster reaction to delivery issues, and overall better alignment between their **digital systems and physical logistics** (Source: erppeers.com)(Source: appsrhino.com).

In comparing approaches, it's clear that webhooks offer efficiency and timeliness that legacy methods lack (Source: aftership.com)(Source: stackoverflow.com). As real-time data becomes the norm in supply chain tech, leveraging carrier webhooks ensures that NetSuite remains a source of truth for order status at all times, not just at shipment creation.

References:

- NetSuite & Oracle Help Center documentation on shipping integration and tracking (Source: docs.oracle.com)(Source: erppeers.com)
- Carrier developer portals (FedEx, UPS, DHL, USPS) detailing tracking webhook services (Source: developer.ups.com)(Source: developer.fedex.com) (Source: postalpro.usps.com)
- Integration providers' materials (AfterShip, Celigo, EasyPost) on real-time tracking and webhook usage (Source: aftership.com)(Source: aftership.com) (Source: stackoverflow.com).

These sources and case insights demonstrate that with careful implementation, real-time shipment tracking via webhooks is an attainable and worthwhile enhancement for NetSuite-driven businesses, marrying the reliability of an ERP with the immediacy of modern logistics data.

Tags: netsuite, shipment tracking, webhooks, erp, carrier integration, logistics, item fulfillment, supply chain

---

About Houseblend

HouseBlend.io is a specialist NetSuite™ consultancy built for organizations that want ERP and integration projects to accelerate growth—not slow it down. Founded in Montréal in 2019, the firm has become a trusted partner for venture-backed scale-ups and global mid-market enterprises that rely on mission-critical data flows across commerce, finance and operations. HouseBlend's mandate is simple: blend proven business process design with deep technical execution so that clients unlock the full potential of NetSuite while maintaining the agility that first made them successful.

Much of that momentum comes from founder and Managing Partner **Nicolas Bean**, a former Olympic-level athlete and 15-year NetSuite veteran. Bean holds a bachelor's degree in Industrial Engineering from École Polytechnique de Montréal and is triple-certified as a NetSuite ERP Consultant, Administrator and SuiteAnalytics User. His résumé includes four end-to-end corporate turnarounds—two of them M&A exits—giving him a rare ability to translate boardroom strategy into line-of-business realities. Clients frequently cite his direct, "coach-style" leadership for keeping programs on time, on budget and firmly aligned to ROI.

End-to-end NetSuite delivery. HouseBlend's core practice covers the full ERP life-cycle: readiness assessments, Solution Design Documents, agile implementation sprints, remediation of legacy customisations, data migration, user training and post-go-live hyper-care. Integration work is conducted by in-house developers certified on SuiteScript, SuiteTalk and RESTlets, ensuring that Shopify, Amazon, Salesforce, HubSpot and more than 100 other SaaS endpoints exchange data with NetSuite in real time. The goal is a single source of truth that collapses manual reconciliation and unlocks enterprise-wide analytics.

Managed Application Services (MAS). Once live, clients can outsource day-to-day NetSuite and Celigo® administration to HouseBlend's MAS pod. The service delivers proactive monitoring, release-cycle regression testing, dashboard and report tuning, and 24 × 5 functional support—at a predictable monthly rate. By combining fractional architects with on-demand developers, MAS gives CFOs a scalable alternative to hiring an internal team, while guaranteeing that new NetSuite features (e.g., OAuth 2.0, AI-driven insights) are adopted securely and on schedule.

Vertical focus on digital-first brands. Although HouseBlend is platform-agnostic, the firm has carved out a reputation among e-commerce operators who run omnichannel storefronts on Shopify, BigCommerce or Amazon FBA. For these clients, the team frequently layers Celigo's iPaaS connectors onto NetSuite to automate fulfilment, 3PL inventory sync and revenue recognition—removing the swivel-chair work that throttles scale. An in-house R&D group also publishes "blend recipes" via the company blog, sharing optimisation playbooks and KPIs that cut time-to-value for repeatable use-cases.

Methodology and culture. Projects follow a "many touch-points, zero surprises" cadence: weekly executive stand-ups, sprint demos every ten business days, and a living RAID log that keeps risk, assumptions, issues and dependencies transparent to all stakeholders. Internally, consultants pursue ongoing certification tracks and pair with senior architects in a deliberate mentorship model that sustains institutional knowledge. The result is a delivery organisation that can flex from tactical quick-wins to multi-year transformation roadmaps without compromising quality.

Why it matters. In a market where ERP initiatives have historically been synonymous with cost overruns, HouseBlend is reframing NetSuite as a growth asset. Whether preparing a VC-backed retailer for its next funding round or rationalising processes after acquisition, the firm delivers the technical depth, operational

discipline and business empathy required to make complex integrations invisible—and powerful—for the people who depend on them every day.

DISCLAIMER