Automatisation du rapprochement des factures dans NetSuite avec SuiteScript

Publié le 27 août 2025 50 min de lecture

Automatisation de la concordance des factures fournisseurs dans NetSuite avec SuiteScript 2.x

Présentation de la concordance des factures (à 2 ou 3 voies)

La concordance des factures en comptabilité fournisseurs (AP) fait référence au processus de vérification des factures fournisseurs par rapport aux documents d'achat associés afin d'assurer leur exactitude et leur légitimité. La **concordance à deux voies** compare une facture uniquement à son bon de commande (BC), confirmant que les articles facturés, les quantités et les prix correspondent à ce qui a été commandé (Source: netsuite.com). La **concordance à trois voies** ajoute un troisième document – le reçu de marchandises (réception d'article) – pour vérifier que les articles facturés ont bien été reçus, en plus de la concordance avec le BC (Source: netsuite.com). En d'autres termes, la concordance à trois

voies recoupe la facture du fournisseur avec son BC correspondant et le bon de livraison pour s'assurer que tous les détails clés (par exemple, les quantités commandées par rapport aux quantités reçues, les prix et les extensions) concordent (Source: netsuite.com)(Source: netsuite.com). Cette étape supplémentaire aide à détecter les écarts ou la fraude (par exemple, articles facturés non livrés) avant que le paiement ne soit approuvé.

Différences clés : Dans la concordance à deux voies, le personnel de la comptabilité fournisseurs compare la facture du fournisseur uniquement au BC (vérifiant que les montants et les quantités facturés ne dépassent pas ce qui a été commandé). Dans la concordance à trois voies, ils s'assurent également que les articles ou services ont été reçus en bon état en comparant la facture aux documents de réception (bordereaux d'expédition ou réceptions d'articles dans NetSuite) (Source: netsuite.com). Par exemple, si seule une partie d'un BC a été reçue, la concordance à trois voies signalerait une facture qui tente de facturer la quantité totale du BC (Source: netsuite.com)(Source: netsuite.com). En vérifiant *ce qui a été commandé*, *ce qui a été reçu* et *ce qui est facturé*, la concordance à trois voies offre un contrôle robuste contre la surfacturation, les doublons ou le paiement de marchandises non livrées (Source: netsuite.com)(Source: netsuite.com).

La plupart des organisations mettent en œuvre des seuils pour déterminer quand la concordance à trois voies est requise (par exemple, en fonction du montant de la facture ou de la catégorie d'article). Cela évite que les factures mineures et à faible risque ne soient inutilement bloquées dans des approbations supplémentaires (Source: netsuite.com). Qu'elle soit à deux ou trois voies, l'objectif de la concordance des factures est le même : détecter les erreurs et les incohérences *avant* que les factures ne soient payées. Lorsqu'il est effectué manuellement, ce processus peut être fastidieux, en particulier dans les environnements de comptabilité fournisseurs à volume élevé (Source: netsuite.com). Comme nous le verrons ensuite, l' automatisation de la concordance des factures dans NetSuite via SuiteScript peut faire gagner un temps considérable et réduire les erreurs.

Problème commercial et justification de l'automatisation

La concordance manuelle des factures fournisseurs avec les BC et les réceptions est chronophage et sujette aux erreurs humaines. Les commis à la comptabilité fournisseurs doivent vérifier ligne par ligne que les prix, les quantités et les conditions concordent entre les documents, et faire le suivi de toute divergence. Dans une entreprise en croissance qui reçoit des centaines ou des milliers de factures fournisseurs, ce processus manuel devient un goulot d'étranglement et peut entraîner des retards, des erreurs non détectées, voire des paiements incorrects. Des études ont montré que *les erreurs de facturation et la fraude sont courantes*, et de petites erreurs (comme une faute de frappe dans le montant

d'une facture) ou des factures frauduleuses peuvent coûter aux entreprises un pourcentage significatif de leurs revenus si elles ne sont pas détectées (Source: netsuite.com)(Source: netsuite.com). Il existe donc une solide justification commerciale pour automatiser le processus de concordance des factures.

Les principales motivations de l'automatisation sont les suivantes :

- **Précision améliorée et prévention de la fraude :** La concordance automatisée réduit les erreurs humaines lors de la comparaison des chiffres. Moins de fautes de frappe ou d'oublis signifie qu'il est plus facile de faire correspondre automatiquement les factures aux BC et de vérifier les quantités, les prix et d'autres détails, aidant ainsi à identifier les écarts et à prévenir les trop-payés (Source: netsuite.com). Cela aide également à signaler les factures potentiellement frauduleuses en s'assurant que le fournisseur et les montants sont attendus (Source: netsuite.com). Par exemple, un système automatisé peut comparer les détails des factures fournisseurs aux BC et mettre en évidence toute facture qui facture des articles non autorisés ou des quantités excessives.

- **Efficacité et économies de coûts :** En confiant le travail répétitif de concordance à un logiciel, les entreprises peuvent **diminuer le coût par facture traitée**. Le personnel de la comptabilité fournisseurs passe moins de temps à manipuler des documents et plus de temps sur les exceptions et les tâches à plus forte valeur ajoutée (Source: netsuite.com)(Source: netsuite.com). L'automatisation accélère également les approbations – les factures qui concordent peuvent être approuvées automatiquement pour paiement, raccourcissant le cycle et aidant à éviter les frais de retard ou à bénéficier des escomptes pour paiement anticipé (Source: netsuite.com)(Source: netsuite.com). Globalement, le coût de traitement des factures et le temps de cycle diminuent considérablement avec l'automatisation.

- **Application cohérente des politiques :** Les scripts automatisés appliquent les règles de concordance uniformément à chaque fois. Ils peuvent appliquer des **limites de tolérance** (par exemple, permettre un petit écart comme 2-3 % sur le prix ou la quantité) et ne diriger que les véritables exceptions pour examen humain (Source: netsuite.com)(Source: docs.oracle.com). Par exemple, un script de concordance automatisé pourrait approuver automatiquement une facture qui se situe dans une variance de prix de 3 %, mais *signaler tout ce qui dépasse ce seuil* pour enquête (Source: netsuite.com). Cette cohérence garantit que la politique d'entreprise (telle que « concorder à trois voies toutes les factures de plus de 5 000 $ ») est suivie systématiquement.

- **Meilleure visibilité et contrôle :** Une solution de concordance automatisée peut enregistrer chaque étape et créer une piste d'audit des approbations et des exceptions (Source: netsuite.com). Elle peut générer des rapports sur les factures non concordantes, les raisons courantes des écarts et les temps de traitement, donnant aux gestionnaires un aperçu du processus de comptabilité fournisseurs. Lors des audits, disposer d'une piste numérique des documents concordants et de la gestion des exceptions peut simplifier la conformité (Source: netsuite.com).

En résumé, **l'automatisation de la concordance des factures fournisseurs dans NetSuite répond à un besoin commercial clair** : elle réduit la charge de travail manuelle, accélère le processus de comptabilité fournisseurs, améliore la précision et renforce les contrôles contre les erreurs ou la fraude (Source: netsuite.com)(Source: netsuite.com). Ensuite, nous discuterons de la manière dont une telle automatisation peut être mise en œuvre à l'aide du framework SuiteScript 2.x de NetSuite.

Architecture de la solution et aperçu du flux de travail

L'automatisation de la concordance des factures dans NetSuite avec SuiteScript 2.x implique une combinaison de scripts personnalisés et éventuellement de workflows pour gérer la logique de concordance aux points appropriés du processus de comptabilité fournisseurs. À un niveau élevé, le flux de travail automatisé pourrait se dérouler comme suit :

1. **Capture/Saisie de la facture :** Une facture fournisseur (facture d'achat dans la terminologie NetSuite) entre dans le système – soit par saisie manuelle (saisie de la facture et sélection du BC associé), via la fonction OCR **Bill Capture** de NetSuite, soit par une intégration d'automatisation de la comptabilité fournisseurs tierce. À ce stade, l'enregistrement de la facture fournisseur est créé dans NetSuite (soit avec le statut **En attente d'approbation**, soit avec un statut personnalisé « À concorder »). La facture fait généralement référence à un numéro de BC (via le lien **Enregistrements associés** > **Bon de commande** ou des associations au niveau des lignes).

2. **Déclenchement de la logique de concordance :** Un **SuiteScript** personnalisé est déclenché pour effectuer la concordance. Cela peut se produire en temps réel via un script **User Event (afterSubmit)** lors de la création de l'enregistrement de la facture fournisseur, et/ou en lot via un **script planifié/Map-Reduce** qui traite périodiquement les factures nouvelles ou en attente. Le script localise le BC associé et toute(s) réception(s) d'article pour ce BC :

    - En utilisant le BC lié à la facture (si la facture a été saisie par rapport à un BC) ou en recherchant le numéro/référence du BC sur la facture.
    - En récupérant l'enregistrement du BC (avec tous les articles et quantités) et tout enregistrement de réception d'article correspondant (marchandises reçues).
    - Si la fonction « Conformer la facture à la réception » est utilisée sur les BC (une option de réception avancée), le script peut également récupérer les lignes de réception spécifiques liées aux lignes de la facture. Sinon, le script prendra en compte la quantité totale reçue sur chaque ligne de BC.

3. **Effectuer la comparaison à 2/3 voies :** Le script compare les détails de la facture fournisseur au BC (concordance à 2 voies) et à la réception si applicable (concordance à 3 voies) :

- Pour chaque ligne d'article sur la facture fournisseur, vérifiez que la **quantité facturée** ne dépasse pas la quantité commandée sur le BC, et (pour la concordance à 3 voies) ne dépasse pas la quantité réellement reçue. Comparez également le **prix unitaire/taux** sur la facture au taux du BC.
- Calculez les différences ou les écarts. Les vérifications courantes incluent :
  - **Écarts de quantité :** par exemple, quantité facturée vs. quantité du BC, et quantité facturée vs. quantité reçue.
  - **Écarts de montant/prix :** par exemple, montant de la ligne de facture vs. montant de la ligne du BC (ou différences de prix unitaire).
  - **Non-concordance dans d'autres champs :** conditions, codes d'article ou emplacement, etc., si ceux-ci doivent correspondre au BC (Source: docs.oracle.com).
- Appliquez des **seuils de tolérance** : par exemple, autorisez un petit écart (par exemple, 5 % ou 50 $) en montant ou une différence de quelques unités en quantité avant de signaler une exception (Source: docs.oracle.com)(Source: docs.oracle.com). La tolérance peut être définie via des paramètres de script ou des champs personnalisés (tels que définis par l'entreprise) (Source: docs.oracle.com)(Source: docs.oracle.com). Si l'écart est dans la tolérance, la facture peut toujours être considérée comme « concordante ». S'il dépasse la tolérance, c'est une véritable exception.

4. **Décision/Action automatisée :** Basé sur la comparaison :
   - Si **tout concorde dans les tolérances définies** (c'est-à-dire une concordance « propre ») :
     - Le script peut marquer la facture fournisseur comme *approuvée* pour paiement ou déclencher une action de workflow d'approbation. Par exemple, il pourrait définir un champ personnalisé « Statut de concordance » sur « Concordant » ou modifier directement le statut de la facture en « Approuvé » si un processus d'approbation personnalisé est utilisé.
     - Optionnellement, une notification par e-mail peut être envoyée aux parties prenantes indiquant que la facture a passé la concordance à 2/3 voies et est approuvée pour paiement.
     - Tout workflow d'approbation associé dans NetSuite (tel que le workflow d'approbation de facture fournisseur à 3 voies de la SuiteApp) peut être automatiquement avancé. (Dans la SuiteApp native de NetSuite, un workflow de concordance à 3 voies approuverait automatiquement les factures sans écarts et ne dirigerait que les exceptions (Source: docs.oracle.com)(Source: docs.oracle.com).)
   - Si des **écarts ou des exceptions sont trouvés** :

- Le script signale la facture pour examen. Cela pourrait impliquer la définition d'un champ comme « Exception de concordance » ou « Nécessite approbation » et la fourniture de détails sur l'écart (par exemple, « La quantité facturée dépasse la quantité reçue de 5 unités »). Dans le workflow de la SuiteApp, ces factures sont acheminées vers un superviseur pour approbation manuelle (Source: docs.oracle.com).
- Une alerte par e-mail peut être envoyée (en utilisant le module `N/email`) au commis à la comptabilité fournisseurs ou au responsable des achats avec les détails de la facture et l'écart détecté (quantité, prix, etc.).
- La facture reste en statut d'approbation en attente ou est mise en attente. Elle ne sera pas payée tant qu'un utilisateur autorisé n'aura pas ajusté le BC/la réception, accepté l'écart ou obtenu un crédit/ajustement du fournisseur.

- Tous les résultats et actions de concordance peuvent être enregistrés pour la piste d'audit. Par exemple, le script peut écrire une ligne dans les notes système de la facture fournisseur ou un enregistrement personnalisé « Journal de concordance des factures » indiquant que *à la Date X, la facture a été auto-approuvée* ou *signalée pour exception en raison d'un écart de prix de 10 %*.

5. **Gestion des différences de synchronisation facture-réception :** Une solution robuste tient compte des cas où une **facture arrive avant que les marchandises ne soient reçues** dans NetSuite. Dans de tels cas (courants dans certaines industries), la tentative de concordance initiale signalera la facture car aucune réception d'article n'est encore enregistrée. L'automatisation peut gérer cela en :
   - Laissant initialement la facture dans un état « en attente de réception » (exception).
   - Puis, lors d'une exécution ultérieure (par exemple, un script planifié qui s'exécute quotidiennement ou toutes les heures), vérifiez si le BC a depuis été reçu. Une fois la réception d'article saisie, le script réévalue la facture. Si elle concorde maintenant (c'est-à-dire que les marchandises sont reçues et que les quantités correspondent), le script peut auto-approuver la facture à ce moment-là. Ce « deuxième passage » garantit que les factures ne restent pas bloquées uniquement en raison de problèmes de synchronisation.
   - Cette approche a été discutée par les utilisateurs de la SuiteApp de concordance à 3 voies – ils ont envisagé un processus planifié pour *vérifier périodiquement les factures « en attente d'approbation » pour voir si leur BC a depuis été réceptionné et concorde maintenant*(Source: community.oracle.com). Dans un script personnalisé, la mise en œuvre est simple avec une recherche de factures fournisseurs en attente d'approbation et une itération à travers celles-ci pour trouver celles où le BC associé est entièrement reçu.

6. **Workflow d'exception :** Pour les factures qui restent non concordantes (par exemple, quantité insuffisante, prix trop élevé, facture en double, etc.), le workflow d'exception de l'entreprise prend le relais. Le script pourrait assigner une tâche à un utilisateur pour enquêter, ou simplement s'appuyer sur le processus régulier de l'équipe de comptabilité fournisseurs pour résoudre les non-concordances. L'automatisation aide en mettant en évidence le problème exact (par exemple, « Le prix sur la facture est de 105 $, mais le prix du BC est de 100 $ – 5 % au-dessus de la tolérance ») afin qu'il puisse être résolu (peut-être en contactant le fournisseur ou en mettant à jour le BC si justifié).

Considérations architecturales : L'automatisation peut être modulaire. Souvent, un **script d'événement utilisateur (afterSubmit)** gère la concordance immédiate pour chaque facture dès sa saisie, en définissant des indicateurs ou un statut d'approbation. En complément, un **script planifié ou map/reduce** pourrait gérer le traitement en masse ou la revérification des exceptions (comme décrit pour les réceptions en attente). La logique de concordance peut être encapsulée dans un **module de bibliothèque** que les deux types de scripts appellent, assurant la cohérence. De cette façon, vous implémentez l'algorithme de concordance une seule fois et le réutilisez dans plusieurs contextes – une bonne pratique pour la maintenabilité (Source: docs.oracle.com). Dans les sections suivantes, nous approfondirons les modules SuiteScript et les types de scripts impliqués dans cette solution.

Modules SuiteScript 2.x utilisés pour la concordance des factures

L'API SuiteScript 2.x de NetSuite fournit plusieurs modules essentiels à la création d'une automatisation de la concordance des factures. Les modules clés incluent :

- **Module N/record :** Ce module permet au script d'interagir avec les enregistrements NetSuite (créer, charger, modifier et enregistrer des enregistrements) (Source: docs.oracle.com). Pour notre cas d'utilisation, le module `N/record` est utilisé pour charger des enregistrements tels que les bons de commande (Purchase Orders), les factures fournisseurs (Vendor Bills) et les réceptions d'articles (Item Receipts), et pour créer ou mettre à jour des enregistrements si nécessaire. Par exemple, le script pourrait charger un enregistrement de bon de commande pour récupérer les quantités et les prix attendus à des fins de comparaison, ou mettre à jour un enregistrement de facture fournisseur pour le marquer comme approuvé. `N/record` permet également de lier des enregistrements – par exemple, si vous créez une facture fournisseur via un script, vous pouvez définir les **champs de référence de commande** (`orderdoc` et `orderline`) sur chaque ligne de facture afin qu'elle soit correctement liée à une ligne de bon de commande (Source: blog.prolecto.com)(Source: blog.prolecto.com). (Nous verrons des exemples de code plus tard.)

- **Module N/search :** Le module de recherche est utilisé pour interroger des enregistrements et trouver des données sans navigation manuelle (Source: docs.oracle.com). Le script de rapprochement utilise `N/search` pour trouver efficacement les enregistrements associés. Les exemples incluent :
  - Rechercher des enregistrements de réception d'articles pour un bon de commande donné (pour additionner les quantités reçues par article).
  - Rechercher toute facture fournisseur existante du même fournisseur avec le même numéro de facture (pour détecter les doublons).
  - Exécuter une recherche enregistrée pour toutes les factures fournisseurs "En attente d'approbation" chaque jour qui nécessitent une réévaluation. Le module `N/search` prend en charge la création de filtres et de colonnes à la volée, ou le chargement de recherches enregistrées préexistantes. Par exemple, on pourrait effectuer une recherche de réceptions d'articles où le champ **Créé à partir de** (lien vers le bon de commande) est un ID interne de bon de commande spécifique, pour obtenir toutes les réceptions pour ce bon de commande. Ceci est crucial pour la logique de rapprochement à trois voies.
- **Module N/runtime :** Le module d'exécution (runtime) fournit des informations sur le contexte et l'environnement d'exécution du script actuel (Source: docs.oracle.com). Il est particulièrement utile pour accéder aux paramètres du script et prendre des décisions spécifiques à l'exécution. Dans notre solution, nous pouvons définir des **paramètres de script** personnalisés (par exemple, *tolérance de quantité %*, *tolérance de montant %*, destinataires d'e-mails, etc.) lors du déploiement du script. En utilisant `N/runtime`, le script peut récupérer ces paramètres au moment de l'exécution – par exemple, `runtime.getCurrentScript().getParameter({ name: 'custscript_qty_tolerance_pct' })` pourrait renvoyer un pourcentage de tolérance configuré par l'administrateur. Cela permet un ajustement facile des règles de rapprochement sans modifier le code. De plus, `N/runtime` peut fournir le contexte d'exécution du script (afin que le code puisse se comporter différemment s'il s'exécute à partir d'un événement utilisateur ou d'un contexte planifié, si nécessaire) et l'utilisateur actuel (utile si nous voulons attribuer une action ou envoyer des notifications depuis le compte de l'utilisateur).
- **Module N/email :** Ce module permet au script d'envoyer des notifications par e-mail depuis NetSuite (Source: docs.oracle.com). Dans un processus de rapprochement automatisé, `N/email` est couramment utilisé pour alerter les utilisateurs sur les exceptions ou pour notifier les approbateurs. Par exemple, si une non-concordance est trouvée, le script pourrait envoyer un e-mail au responsable des achats avec les détails de l'écart. Le module d'e-mail peut envoyer des e-mails aux employés, aux fournisseurs ou à des adresses arbitraires, et peut être configuré pour du texte simple ou du contenu HTML riche. Une utilisation de base est `email.send()` avec des paramètres pour

l'auteur, les destinataires, l'objet et le corps. L'auteur doit être un utilisateur NetSuite valide (souvent un ID d'employé, tel qu'un commis aux comptes fournisseurs ou un utilisateur générique "NetSuite"). Exemple : le script pourrait faire :

```
Copy

email.send({
    author: runtime.getCurrentUser().id,
    recipients: ['ap-manager@example.com'],  // or employee IDs
    subject: 'Vendor Bill Match Exception',
    body: 'Invoice 1001 from ABC Corp exceeds PO amount by 10%. Please review.'
});
```

Ceci enverrait un e-mail au responsable des comptes fournisseurs avec un message concernant la facture spécifique qui n'a pas été rapprochée. L'utilisation de `N/email` garantit que les exceptions critiques ne passent pas inaperçues – les personnes concernées sont informées immédiatement.

En plus de ceux-ci, l'automatisation pourrait tirer parti de quelques autres modules SuiteScript dans des rôles de support :

- **N/log :** Bien que non listé dans les exigences de la question, il convient de noter que le script utilisera le module de journalisation pour enregistrer les détails d'exécution. `N/log` (ou l'objet global `log`) permet d'écrire des messages de débogage ou d'audit dans le journal du script, ce qui est inestimable pour le dépannage (Source: docs.oracle.com). Par exemple, la journalisation de chaque facture vérifiée et si elle a été approuvée automatiquement ou signalée fournit un historique dans le journal de déploiement du script.

- **N/error :** Un autre module de support, `N/error`, peut être utilisé pour lever des erreurs personnalisées lorsque quelque chose ne va pas (comme une condition fatale) (Source: docs.oracle.com). Bien qu'idéalement le script gérerait les exceptions avec élégance, `N/error` pourrait être utilisé pour créer une erreur qui déclenche un échec et est capturée par la journalisation de NetSuite (ou par un processus appelant). Dans notre contexte, nous essayons généralement de gérer les erreurs par facture et de continuer avec la suivante, mais si, par exemple, un enregistrement requis (bon de commande ou réception) ne peut pas être trouvé ou s'il y a un problème de gouvernance, l'utilisation de `N/error` pour annuler et notifier pourrait être appropriée.

En combinant ces modules – `record` pour la manipulation des données, `search` pour la récupération des données, `runtime` pour le contexte/les paramètres, `email` pour les notifications (et `log` / `error` pour le débogage) – nous disposons de la boîte à outils nécessaire pour implémenter la logique de rapprochement dans SuiteScript 2.x.

Stratégies d'implémentation (Types de scripts)

Plusieurs types de scripts SuiteScript 2.x peuvent être utilisés pour cette automatisation, chacun étant adapté à différents aspects du processus. Une solution robuste pourrait utiliser un mélange de ceux-ci :

Scripts d'événements utilisateur (Rapprochement en temps réel à la saisie)

Un **script d'événement utilisateur** (généralement un script *afterSubmit*) sur l'enregistrement de facture fournisseur peut effectuer le rapprochement immédiatement lorsqu'une facture est créée ou modifiée. Cette approche fournit un retour d'information en temps réel. Par exemple, dès qu'un commis aux comptes fournisseurs saisit une facture fournisseur et l'enregistre, le script afterSubmit peut vérifier le rapprochement à 2 ou 3 voies :

- Si la facture passe le rapprochement, le script pourrait l'approuver automatiquement ou marquer un champ (afin que l'interface utilisateur ou un workflow sache qu'elle est rapprochée).
- Si elle échoue, le script pourrait faire passer un champ "Statut de rapprochement" à "Exception" et même envoyer automatiquement une alerte à l'utilisateur approprié.

Avantages : La facture est validée immédiatement, et tout problème est détecté au point de saisie. Le personnel des comptes fournisseurs n'a pas à attendre un processus par lots – il pourrait même être informé de corriger quelque chose immédiatement (par exemple, s'il a accidentellement saisi une quantité ou un prix erroné, le script pourrait potentiellement le rejeter ou le corriger avant que la facture ne soit entièrement enregistrée, en utilisant un beforeSubmit si désiré).

Considérations : Les scripts d'événements utilisateur s'exécutent dans le cadre de la transaction d'enregistrement, ils doivent donc être efficaces. Les meilleures pratiques de NetSuite conseillent de maintenir l'exécution des événements utilisateur en dessous d'environ 5 secondes (Source: docs.oracle.com). Si un bon de commande a de nombreuses lignes, le script parcourra et comparera potentiellement des centaines de lignes – ce qui peut être fait en quelques secondes si le code est bien écrit, mais des calculs lourds ou de grandes recherches pourraient ralentir l'enregistrement. Ainsi :

- Il est judicieux de ne récupérer que les données nécessaires (par exemple, charger le bon de commande et éventuellement effectuer une recherche de réceptions, mais éviter plusieurs recherches imbriquées par ligne).

- Une logique complexe pourrait être déchargée : par exemple, le script d'événement utilisateur pourrait simplement signaler la facture comme nécessitant une révision, puis un script planifié effectuerait le travail lourd plus tard. Cependant, la plupart des logiques de rapprochement (quelques comparaisons et une ou deux recherches) sont gérables dans un afterSubmit.

Les événements utilisateur ont accès au nouvel enregistrement et à l'ancien enregistrement (en cas de modifications), ce qui est utile. Par exemple, lors de la création, `context.newRecord` donne la facture fournisseur qui vient d'être saisie. Le script peut obtenir l'ID interne du bon de commande associé à partir de `newRecord.getValue({ fieldId: 'purchaseorder' })` si la facture est saisie via un bon de commande, ou à partir de l'`orderdoc` de chaque ligne d'article si le lien est fait ligne par ligne. Il utilise ensuite `record.load` pour charger ce bon de commande et effectuer des vérifications.

Il faut également gérer le contexte : le script pourrait vouloir s'exécuter uniquement lors de la création (et peut-être lors de la modification, si un nouveau rapprochement est nécessaire lorsqu'une facture est modifiée). Nous pouvons vérifier `context.type` pour ignorer les exécutions sur d'autres événements. De plus, si le script lui-même met à jour l'enregistrement (marquant des champs), nous voulons éviter une récursion infinie – souvent gérée en utilisant un champ personnalisé ou une vérification `contextType` pour ne pas redéclencher lors de la propre mise à jour du script.

Scripts planifiés (Traitement par lots)

Un **script planifié** peut être déployé pour s'exécuter périodiquement (par exemple, toutes les nuits ou toutes les heures) afin de traiter les factures par lots. Ceci est utile pour :

- Gérer les factures qui n'ont pas été rapprochées en temps réel (ou si le rapprochement en temps réel n'est pas implémenté).
- Revérifier les factures en attente en raison de réceptions manquantes (comme décrit précédemment).
- Traiter un grand volume de factures pendant les heures creuses pour réduire la charge sur le système pendant la journée.

Un script planifié s'exécute en arrière-plan et peut itérer sur un ensemble d'enregistrements. Par exemple, le script pourrait effectuer une recherche de toutes les factures fournisseurs avec le statut "En attente d'approbation" (ou une case à cocher personnalisée "Nécessite un rapprochement" cochée), puis parcourir les résultats et effectuer la logique de rapprochement sur chacun d'eux.

Avantages : Le traitement par lots peut gérer de grands volumes sans affecter l'expérience immédiate de l'utilisateur. Il peut également consolider les notifications – par exemple, envoyer un e-mail récapitulatif de toutes les exceptions trouvées dans la journée.

Considérations : Les scripts planifiés SuiteScript 2.x ne **cèdent pas automatiquement** pour les limites de gouvernance (comme les unités d'utilisation ou le temps) (Source: docs.oracle.com). Si le script doit traiter un très grand nombre d'enregistrements, il pourrait potentiellement rencontrer des limites de gouvernance ou des temps d'exécution longs. Dans de tels cas, NetSuite recommande d'utiliser des scripts Map/Reduce pour l'évolutivité (Source: docs.oracle.com)(Source: docs.oracle.com). Un script planifié convient pour des volumes modérés ou si vous implémentez un "yielding" manuel (par exemple, se replanifier après un certain nombre d'enregistrements). Mais pour la simplicité, si le volume est élevé, passez directement à Map/Reduce (voir section suivante).

Les scripts planifiés peuvent être configurés pour s'exécuter à des moments spécifiques. Les meilleures pratiques de NetSuite suggèrent de planifier les scripts lourds pendant les heures creuses (par exemple, de 2 h à 6 h du matin, heure du Pacifique) pour minimiser les conflits avec les utilisateurs interactifs (Source: docs.oracle.com). Vous pouvez déployer le script pour qu'il s'exécute quotidiennement, ou même plusieurs fois par jour, selon la rapidité avec laquelle vous souhaitez que les non-concordances soient traitées. Le déploiement du script peut également être déclenché à la demande via l'interface utilisateur ou via `N/task.submit()` à partir d'un autre script si nécessaire.

Scripts Map/Reduce (Traitement par lots évolutif)

Un **script Map/Reduce (M/R)** dans SuiteScript 2.x est conçu pour le traitement robuste et parallèle de grands ensembles de données. Il est idéal pour l'automatisation du rapprochement des factures si vous traitez de **grands volumes de transactions** ou une logique complexe par enregistrement. Le type de script Map/Reduce gérera automatiquement le **yielding de gouvernance** et peut exécuter plusieurs étapes "map" en parallèle, accélérant le traitement de nombreux enregistrements (Source: docs.oracle.com)(Source: docs.oracle.com).

Dans le contexte du rapprochement des factures :

- L'étape **getInputData** pourrait rechercher toutes les factures fournisseurs qui nécessitent un rapprochement (par exemple, toutes les nouvelles factures du dernier jour, ou toutes les factures en attente d'approbation).
- L'étape **map** prendrait chaque enregistrement de facture et effectuerait la logique de rapprochement (charger le bon de commande, charger les réceptions, comparer les champs, etc.). Chaque facture est traitée indépendamment, ce qui correspond au paradigme map.
- L'étape **reduce** pourrait ne pas être fortement nécessaire, sauf si l'on combine les résultats, mais elle pourrait agréger les exceptions ou peut-être regrouper par fournisseur pour un résumé.
- L'étape **summarize** peut rapporter les résultats globaux (par exemple, enregistrer combien ont été approuvés automatiquement et combien d'exceptions).

Avantages : Les scripts Map/Reduce peuvent s'exécuter en parallèle, ce qui signifie que si vous avez des centaines de factures à vérifier, plusieurs factures peuvent être rapprochées simultanément – ce qui se traduit par un débit global plus rapide. Il est important de noter que le framework va automatiquement **céder et replanifier** des segments du travail si les limites de gouvernance sont atteintes, ce qui signifie que vous n'avez pas à écrire manuellement du code pour gérer l'épuisement des unités d'utilisation (Source: docs.oracle.com). La documentation de NetSuite suggère explicitement d'utiliser Map/Reduce plutôt qu'un script planifié lors du traitement de plusieurs enregistrements et lorsque la logique peut être décomposée en plus petits morceaux (Source: docs.oracle.com).

Considérations : Les scripts Map/Reduce sont un peu plus complexes à écrire (plus de code passe-partout pour les étapes). De plus, si la logique de rapprochement est simple et que le volume n'est pas énorme, un script planifié pourrait suffire. Mais compte tenu de leurs avantages, de nombreux développeurs choisissent Map/Reduce pour tout traitement de données non trivial en 2.x. Comme approche hybride, vous pourriez avoir un événement utilisateur qui marque les enregistrements, puis un Map/Reduce (invoqué soit selon un calendrier, soit même déclenché via `N/task` par l'événement utilisateur) gère le traitement lourd du rapprochement de manière asynchrone – donnant à l'utilisateur un enregistrement immédiat, puis effectuant le rapprochement un peu plus tard.

En résumé, **le choix du bon type de script** dépend des exigences :

- Utilisez l'**événement utilisateur afterSubmit** pour un rapprochement quasi instantané et des charges utiles plus petites (typique dans de nombreux cas).
- Utilisez le script **planifié** pour des vérifications en masse périodiques ou si vous préférez un script plus simple et pouvez gérer les volumes.
- Utilisez **map/reduce** pour un volume élevé et lorsque vous souhaitez que le système gère l'exécution concurrente et la gouvernance avec élégance (particulièrement utile si vous rapprochez des centaines de lignes sur de nombreuses factures, car il parallélise et cède automatiquement (Source: docs.oracle.com)(Source: docs.oracle.com)).

Il est à noter que tous ces types de scripts peuvent coexister. Par exemple, vous pourriez :

1. Effectuer un rapprochement rapide à 2 voies dans un événement utilisateur (facture vs bon de commande) et signaler si quelque chose d'évident ne va pas.
2. Laisser un script planifié/MapReduce gérer plus tard le rapprochement complet à 3 voies une fois la réception effectuée, ou pour revérifier et finaliser l'approbation.

Cette approche en couches garantit la réactivité et l'exhaustivité.

Exemples d'extraits de code et de conception SuiteScript

Pour illustrer comment on peut implémenter des parties de cette solution, voici quelques extraits de code SuiteScript 2.x simplifiés. Ces exemples se concentrent sur des tâches clés telles que la liaison d'enregistrements, l'exécution de recherches et la structuration du code pour la réutilisation. Dans un déploiement réel, vous organiseriez ceux-ci en fonctions et éventuellement en modules de bibliothèque séparés.

1. Liaison d'une facture fournisseur à un bon de commande (via SuiteScript)

Lors de la création d'une facture fournisseur via un script (par exemple, si vous automatisez la saisie de données à partir d'un système OCR), il est important de lier la facture au bon de commande afin que NetSuite sache qu'elle est liée. Normalement, lorsque vous facturez manuellement un bon de commande dans NetSuite, le système définit des champs de liaison cachés. Dans SuiteScript, vous y parvenez en définissant l'`orderdoc` (ID interne de la commande) et l'`orderline` (identifiant de ligne) sur chaque ligne d'article de la facture fournisseur (Source: blog.prolecto.com)(Source: blog.prolecto.com):

Copy

```
define(['N/record'], function(record) {
    function createVendorBillFromPO(poId, vendorId) {
        // Charger l'enregistrement de bon de commande pour obtenir les détails des lig
        var poRec = record.load({ type: record.Type.PURCHASE_ORDER, id: poId });

        // Créer un nouvel enregistrement de facture fournisseur
        var billRec = record.create({ type: record.Type.VENDOR_BILL, isDynamic: true }),

        // Définir le fournisseur (entité) sur la facture
        billRec.setValue({ fieldId: 'entity', value: vendorId });

        // Itérer sur les lignes du bon de commande pour les ajouter à la facture
        var lineCount = poRec.getLineCount({ sublistId: 'item' });
        for (var i = 0; i < lineCount; i++) {
            // Sélectionner une nouvelle ligne dans la sous-liste d'articles de la factu
            billRec.selectNewLine({ sublistId: 'item' });

            // Obtenir les valeurs de liaison du bon de commande
            var poLineInternalId = poRec.getSublistValue({ sublistId: 'item', fieldId:
            var poLineItem = poRec.getSublistValue({ sublistId: 'item', fieldId: 'item',
            var poLineQty = poRec.getSublistValue({ sublistId: 'item', fieldId: 'quanti
            var poLineRate = poRec.getSublistValue({ sublistId: 'item', fieldId: 'rate',

            // Définir les champs de liaison requis sur la ligne de facture
            billRec.setCurrentSublistValue({ sublistId: 'item', fieldId: 'orderdoc', val
            billRec.setCurrentSublistValue({ sublistId: 'item', fieldId: 'orderline', va

            // Définir l'article, la quantité, le taux sur la ligne de facture (copie du
            billRec.setCurrentSublistValue({ sublistId: 'item', fieldId: 'item', value:
            billRec.setCurrentSublistValue({ sublistId: 'item', fieldId: 'quantity', val
            billRec.setCurrentSublistValue({ sublistId: 'item', fieldId: 'rate', value:

            billRec.commitLine({ sublistId: 'item' });
        }
```

```
        var billId = billRec.save();
        return billId;
    }

    return { createVendorBillFromPO: createVendorBillFromPO };
});
```

Ce que cela fait : Cela crée par programme une facture fournisseur à partir d'un bon de commande (BC). Chaque ligne du BC est ajoutée à la facture avec `orderdoc` défini sur l'ID interne du BC et `orderline` défini sur l'*identifiant de ligne* du BC. Ce lien est crucial pour que NetSuite reconnaisse la facture comme étant liée au BC (permettant le lien BC-Facture et la logique de comptabilisation des écarts). Marty Zigman (expert NetSuite) a documenté cette approche, confirmant que l'utilisation des champs `orderdoc` et `orderline` est la bonne manière de lier une facture autonome à un BC via un script (Source: blog.prolecto.com)(Source: blog.prolecto.com).

En pratique, vous ne pouvez pas toujours ajouter *toutes* les lignes du BC ; vous pourriez facturer partiellement. Le code ci-dessus pourrait être adapté pour ne facturer que certaines lignes ou des quantités partielles si la facture concerne une expédition partielle. Vous ajusteriez `poLineQty` ou ignoreriez des lignes en conséquence.

2. Vérification de la facture par rapport au BC et aux réceptions (Pseudo-code)

Voici un extrait conceptuel (sous forme de pseudo-code) montrant comment on pourrait récupérer un BC et ses réceptions et les comparer aux lignes d'une facture. Cela résiderait probablement dans un événement utilisateur `afterSubmit` ou dans la fonction `map` d'un Map/Reduce :

```
define(['N/record', 'N/search', 'N/runtime', 'N/email'], function(record, search, runtir
    function afterSubmit(context) {
        if (context.type !== context.UserEventType.CREATE && context.type !== context.Us
            return; // Only run on create or edit of Vendor Bill
        }
        var billRec = context.newRecord;
        var billId = billRec.id;
        var vendorId = billRec.getValue({ fieldId: 'entity' });
        var linkedPO = billRec.getValue({ fieldId: 'purchaseorder' });  // assumes stanc
        if (!linkedPO) {
            log.debug('Invoice Matching', 'Bill ' + billId + ' has no linked PO. Skippir
            return;
        }
        // Load the linked Purchase Order
        var poRec = record.load({ type: record.Type.PURCHASE_ORDER, id: linkedPO });
        var poTotalLines = poRec.getLineCount({ sublistId: 'item' });
        // Build a map of PO quantities for quick lookup (item -> quantity ordered, rece
        var poLines = {};
        for (var i = 0; i < poTotalLines; i++) {
            var itemId = poRec.getSublistValue({ sublistId: 'item', fieldId: 'item', lir
            var orderedQty = poRec.getSublistValue({ sublistId: 'item', fieldId: 'quanti
            var orderLineKey = poRec.getSublistValue({ sublistId: 'item', fieldId: 'line
            poLines[orderLineKey] = {
                item: itemId,
                orderedQty: orderedQty,
                receivedQty: 0
            };
        }
        // Search for item receipts associated with this PO to sum received quantities
        var receiptSearch = search.create({
            type: search.Type.ITEM_RECEIPT,
            filters: [
                ['createdfrom', 'anyof', linkedPO], 'AND',
                ['mainline', 'is', 'F']  // we want line-level for summing items (or cou
            ],
            columns: [
                'item',
```

```
            'quantity'
        ]
    });
    receiptSearch.run().each(function(result) {
        var itemId = result.getValue({ name: 'item' });
        var qtyRec = parseFloat(result.getValue({ name: 'quantity' })) || 0;
        // If receipts can be identified by PO line, we might get the orderline in a
        // For simplicity, aggregate by item in this pseudo-code:
        for (var lineKey in poLines) {
            if (poLines[lineKey].item == itemId) {
                poLines[lineKey].receivedQty += qtyRec;
            }
        }
        return true;
    });
    // Tolerance from script parameters
    var pctTolerance = parseFloat(runtime.getCurrentScript().getParameter({ name: '
    var qtyTolerance = parseFloat(runtime.getCurrentScript().getParameter({ name: '
    var hasException = false;
    var exceptionMessages = [];
    // Iterate through vendor bill lines to compare
    var billLineCount = billRec.getLineCount({ sublistId: 'item' });
    for (var j = 0; j < billLineCount; j++) {
        var billItem = billRec.getSublistValue({ sublistId: 'item', fieldId: 'item',
        var billQty = parseFloat(billRec.getSublistValue({ sublistId: 'item', fieldI
        var billRate = parseFloat(billRec.getSublistValue({ sublistId: 'item', fiel
        // Identify the matching PO line via orderline or item
        var orderLineRef = billRec.getSublistValue({ sublistId: 'item', fieldId: 'o
        if (!orderLineRef || !poLines[orderLineRef]) {
            // Fallback: match by item (not ideal if duplicate items on PO)
            log.debug('Matching', 'Bill line ' + j + ' item ' + billItem + ' not mat
            continue;
        }
        var orderedQty = poLines[orderLineRef].orderedQty;
        var receivedQty = poLines[orderLineRef].receivedQty;
        // Compare quantities:
        if (billQty > orderedQty + (qtyTolerance || 0)) {
```

```
                hasException = true;
                exceptionMessages.push('Item ' + billItem + ' billed qty ' + billQty +
            }
            if (billQty > receivedQty + (qtyTolerance || 0)) {
                hasException = true;
                exceptionMessages.push('Item ' + billItem + ' billed qty ' + billQty +
            }
            // Compare amount/price:
            var poRate = parseFloat(poRec.getSublistValue({ sublistId: 'item', fieldId:
            if (poRate && billRate > poRate * (1 + (pctTolerance/100))) {
                hasException = true;
                exceptionMessages.push('Item ' + billItem + ' billed rate $' + billRate
            }
        }
        // Take action based on match result
        if (!hasException) {
            // Auto-approve or mark as matched
            record.submitFields({
                type: record.Type.VENDOR_BILL,
                id: billId,
                values: { custbody_match_status: 'MATCHED' /* custom field indicating ma
            });
            log.audit('Invoice Matching', 'Bill ' + billId + ' auto-approved (matched).
        } else {
            // Flag the bill and notify
            record.submitFields({
                type: record.Type.VENDOR_BILL,
                id: billId,
                values: { custbody_match_status: 'EXCEPTION' }
            });
            email.send({
                author: -5,  // -5 = default system user (or use an employee ID)
                recipients: ['ap@mycompany.com'],
                subject: 'Vendor Bill Match Exception for Bill ' + billRec.getValue({fie
                body: 'Vendor Bill Internal ID ' + billId + ' did not match: \n' + excep
            });
            log.audit('Invoice Matching', 'Bill ' + billId + ' flagged for exceptions:
```

```
            }
        }
        return { afterSubmit: afterSubmit };
});
```

(Note : Le code ci-dessus est à titre d'illustration ; un code de production réel devrait inclure une gestion des erreurs appropriée et éventuellement optimiser certaines recherches.)

Dans ce pseudo-code, nous démontrons la logique de base :

- Chargement du BC et préparation d'un ensemble de données des quantités commandées.
- Totalisation des quantités reçues via une recherche (en pratique, on pourrait affiner la recherche pour regrouper par article ou par ligne de BC).
- Récupération des valeurs de tolérance à partir des paramètres du script (en pourcentages ou en nombres absolus).
- Parcourir chaque ligne de facture et la comparer aux données de la ligne de BC correspondante :
  - Si une non-concordance au-delà de la tolérance est trouvée, nous l'enregistrons comme une exception.
- Enfin, nous marquons l'enregistrement comme concordant (et éventuellement le définissons directement au statut **Approuvé** si vous utilisez un processus d'approbation personnalisé ou un état SuiteFlow) ou le marquons comme exception et envoyons une alerte par e-mail.

Cette approche met en évidence une **conception modulaire** : on pourrait séparer la logique de rapprochement dans sa propre fonction (par exemple, `matchInvoice(billId)` renvoie un objet ou lève une exception). Cette fonction pourrait ensuite être invoquée à la fois à partir d'un événement utilisateur et d'un script planifié. Ce faisant, nous adhérons aux principes DRY (Don't Repeat Yourself) – les critères de rapprochement sont définis en un seul endroit. NetSuite permet de créer des modules personnalisés pour un tel code partagé, qui peuvent être référencés via `define` / `require` dans plusieurs scripts.

3. Conception de script modulaire

En tant que bonne pratique, envisagez de structurer votre code SuiteScript en modules réutilisables et en petites fonctions (Source: docs.oracle.com) :

- Un **script de bibliothèque** (par exemple, *InvoiceMatchLib.js*) pourrait contenir des fonctions comme `compareBillToPO(billRecord)` qui renvoie un tableau d'exceptions (ou vide si aucune). Il pourrait également avoir des fonctions d'aide pour des choses comme le chargement de toutes les réceptions pour un BC, ou le calcul des écarts.

- Les scripts d'événement utilisateur et planifiés/MapReduce `requereraient` alors cette bibliothèque. Par exemple, l'`afterSubmit` de l'événement utilisateur appellerait simplement quelque chose comme :

```
var result = InvoiceMatchLib.compareBillToPO(record.load({...})); if(result.exceptions
```

De cette façon, toute la logique détaillée se trouve en un seul endroit.

- L'utilisation d'une approche modulaire facilite la maintenance. Si l'entreprise modifie la tolérance ou souhaite ajouter une nouvelle vérification (par exemple, s'assurer que la **date de réception de l'article** est dans les X jours suivant la date de la facture), vous mettez à jour la bibliothèque et cela affecte tous les points d'entrée du script de manière cohérente.
- De plus, gardez à l'esprit la **gouvernance** : si la fonction de rapprochement peut être utilisée en masse, assurez-vous qu'elle ne consomme pas une utilisation excessive. Par exemple, préférez une seule recherche qui renvoie toutes les réceptions nécessaires plutôt que de faire une recherche par ligne de facture. Dans le code ci-dessus, nous avons effectué une seule `receiptSearch` pour tous les articles du BC, ce qui est plus efficace qu'à l'intérieur de la boucle.

Ces extraits et conseils devraient donner un aperçu de l'implémentation SuiteScript 2.x pour le rapprochement des factures. Dans un scénario réel, le code inclurait plus de vérifications (par exemple, la gestion des **lignes de dépenses** sur les factures en plus des lignes d'articles, la gestion des **BC qui ont plusieurs factures partielles**, etc.), mais la structure reste similaire.

Gestion des cas limites et des exceptions courants

L'automatisation du rapprochement des factures nécessite d'anticiper divers cas limites et de décider comment le script doit les gérer. Voici quelques scénarios courants et les stratégies recommandées :

- **Écarts mineurs (dans la tolérance) :** Il est courant d'autoriser de petites différences dues aux arrondis, aux conversions d'unités ou aux augmentations de prix. Comme mentionné, la mise en œuvre de champs de tolérance (soit comme paramètres de script, soit comme champs personnalisés) vous permet de définir des seuils en pourcentage ou absolus (Source: docs.oracle.com)(Source: docs.oracle.com). Par exemple, une facture qui dépasse de 2 % le montant du BC pourrait toujours être approuvée automatiquement si la tolérance est fixée à 3 %. Le script doit comparer l'écart par rapport à la tolérance et traiter ce qui est dans la tolérance comme une "concordance" (Source: docs.oracle.com). Toute la logique de tolérance doit être centralisée afin d'être facile à ajuster.

- **Quantité reçue inférieure à la quantité facturée (facture avant livraison) :** Cela se produit si les fournisseurs facturent avant la livraison ou si la réception est retardée dans le système. Le script signalera de tels cas (échec du rapprochement à 3 voies). La résolution pourrait être :
  - La facture reste non approuvée jusqu'à la réception des marchandises. Le script planifié peut détecter plus tard que la *quantité reçue* a rattrapé son retard et approuver ensuite la facture.
  - Alternativement, certaines entreprises pourraient choisir de payer partiellement la facture à la quantité reçue ou de contacter le fournisseur pour clarification. En termes d'automatisation, la facture reste généralement en exception jusqu'à ce qu'elle soit résolue. Le workflow de rapprochement à 3 voies de la SuiteApp ne correspond spécifiquement *pas* automatiquement si une réception d'article est manquante ou partielle, nécessitant un examen par un superviseur (Source: docs.oracle.com) (il note même que les réceptions partielles ne sont pas prises en charge pour l'approbation automatique).
  - Si une facturation partielle est attendue, le script pourrait être rendu intelligent : par exemple, si la quantité facturée > quantité reçue, mais que la réception restante arrive plus tard, approuver automatiquement une fois que la réception restante arrive (comme discuté).
- **Quantité facturée supérieure à la quantité commandée :** C'est un signal d'alarme – soit le BC nécessite un ordre de modification, soit la facture porte sur quelque chose qui n'a pas été commandé. Le script doit le détecter à chaque fois. L'action peut être de bloquer l'approbation et d'alerter les achats. Pas d'approbation automatique dans ce cas ; un humain doit décider (peut-être que le BC était erroné, ou que le fournisseur a expédié en trop et a besoin d'un nouveau BC, etc.). Le système pourrait créer une **demande de modification de BC** ou simplement noter l'écart pour un suivi manuel.
- **Écarts de prix :** Si le prix unitaire d'une facture est supérieur à celui du BC, et au-delà de toute tolérance, il doit être signalé. Souvent, les achats ou les comptes fournisseurs contacteront le fournisseur ou vérifieront s'il y a eu un changement de prix convenu non reflété dans le BC. Parfois, de tels écarts peuvent être autorisés mais nécessitent une approbation supplémentaire (par exemple, l'approbation du chef de service si le prix > X % par rapport au BC). Le script pourrait s'intégrer à une matrice d'approbation – par exemple, acheminer la facture vers un approbateur supérieur si l'écart > seuil. Pour simplifier, notre automatisation approuve automatiquement dans la tolérance ou la signale au-delà de la tolérance.
- **Frais supplémentaires non inclus dans le BC :** Les fournisseurs peuvent ajouter des frais de transport, des taxes ou des frais divers qui ne figuraient pas sur le BC original. Si votre processus les attend, vous pouvez les gérer :
  - Si le transport ou la taxe figure sur la facture mais pas sur le BC, cela peut être acceptable. Vous pourriez configurer le script pour ignorer certains frais non liés au BC (ou les gérer séparément).

NetSuite pourrait traiter le transport comme une ligne de dépense ; le script peut être conçu pour ignifier le rapprochement sur des types de lignes ou des articles spécifiques (par exemple, un article d'expédition).

- Si des frais complètement inattendus apparaissent, cela devrait être une exception. Le script peut signaler toute ligne de facture qui ne correspond pas à une ligne de BC (en vérifiant si `orderline` est manquant ou si une ligne de dépense n'a pas de référence de BC associée) et la lister dans les messages d'exception.

- **Factures en double :** Payer deux fois la facture d'un fournisseur est une erreur classique des comptes fournisseurs à éviter. NetSuite a une préférence "Facture fournisseur - Appliquer une référence unique" qui, si elle est activée, empêchera automatiquement l'enregistrement d'une facture fournisseur en double avec le même fournisseur et le même numéro de facture (référence). Si cela n'est pas activé ou si vous voulez une sécurité supplémentaire, le script peut effectuer une vérification des doublons. En utilisant `N/search`, vous pourriez rechercher les factures fournisseurs existantes avec le même fournisseur (`entity`) et le même numéro de facture (`tranid` ou `custbody_vendor_invoice_num` si vous utilisez un champ personnalisé). Si trouvé, le script ne doit pas approuver la nouvelle facture ; au lieu de cela, la marquer comme une exception de doublon. Cela pourrait empêcher les doublons accidentels (ou signaler les doublons intentionnels, par exemple, si un fournisseur a renvoyé une facture, les comptes fournisseurs peuvent alors en annuler une). L'enregistrement des doublons dans un tableau séparé ou l'envoi d'alertes immédiates peut aider à les résoudre rapidement. De nombreux outils d'automatisation des comptes fournisseurs tiers incluent également une détection robuste des doublons dans le cadre de la capture des factures.

- **Pas de bon de commande (factures sans BC) :** Certaines factures n'ont pas de BC correspondant (par exemple, factures de services publics, services non couverts par un BC, etc.). Dans de tels cas, le "rapprochement" pourrait ne pas s'appliquer. Vous pouvez décider :

  - Si votre processus exige que chaque facture ait un BC, alors toute facture sans BC est une exception – acheminez-la aux achats ou à la direction pour décider de l'approuver ou de la rejeter.
  - Si certaines factures sont censées être sans BC, vous pourriez ignorer la routine de rapprochement pour celles-ci. Le script ci-dessus montre un exemple où, si aucun BC lié n'est trouvé, il enregistre et ignore (Source: stampli.com). Alternativement, vous pourriez implémenter un rapprochement à 2 voies par rapport à un *contrat fournisseur* ou un budget pour celles-ci, mais cela dépasse la portée typique.
  - Il est important de communiquer aux utilisateurs : les factures sans BC ne seront pas approuvées automatiquement ; elles suivront le processus d'approbation normal (ou nécessiteront toujours une approbation manuelle).

- **Plusieurs BC pour une seule facture :** Occasionnellement, un fournisseur peut envoyer une seule facture combinée pour des articles qui ont été commandés sur des BC distincts. L'interface utilisateur de NetSuite ne permet pas directement à une facture fournisseur de se lier à plusieurs BC, mais via SuiteScript (ou les services web SOAP), il est possible de référencer plusieurs BC lors de la création d'une facture (Source: nanonets.com)(Source: nanonets.com). Si votre processus consolide de cette manière, votre logique de rapprochement doit le gérer :
  - Dans l'extrait de création que nous avons montré, vous pourriez passer plusieurs références de BC et de lignes. Le script de rapprochement itérerait alors peut-être à travers tous les BC liés.
  - Généralement, une approche plus simple consiste à diviser ces factures en plusieurs factures (une par BC) dans NetSuite pour le rapprochement, mais si ce n'est pas souhaité, votre script peut le faire. Cela ajoute simplement de la complexité – vous devriez collecter plusieurs BC et leurs réceptions et les comparer collectivement à la facture unique.
  - Les solutions tierces prétendent souvent gérer automatiquement les factures multi-BC (Source: nanonets.com) (en créant plusieurs factures en arrière-plan ou en utilisant des enregistrements personnalisés). Pour un script personnalisé, c'est faisable mais une tenue de registres minutieuse est nécessaire.
- **Facturation partielle / Plusieurs factures par BC :** C'est très courant (un BC peut être facturé en plusieurs parties par plusieurs factures). Le script doit pouvoir gérer le fait qu'une ligne de BC puisse être répartie sur plusieurs factures. Le lien intégré de NetSuite (`orderline`/`orderdoc`) garantit que chaque facture sait à quel BC et à quelle ligne elle est liée, et le BC garde une trace de la *quantité facturée* en interne. Notre script peut considérer que :
  - Pour le rapprochement des quantités, assurez-vous que la quantité de la facture plus toute autre quantité facturée ne dépasse pas la quantité commandée (si nécessaire, on peut additionner les quantités facturées existantes des autres factures).
- NetSuite stocke la *quantité facturée* sur la ligne de commande (visible dans l'onglet secondaire Facturation de la commande dans l'interface utilisateur, bien que les factures partielles multiples ne soient pas toutes listées par défaut, sauf si un lien approprié est utilisé comme indiqué). Si le lien approprié est utilisé, la **quantité facturée** d'une ligne de commande est mise à jour après chaque facture fournisseur (sauf lors de l'utilisation de l'approche de transformation une seule fois). Ainsi, le script pourrait même récupérer `poRec.getSublistValue({sublistId:'item', fieldId:'quantitybilled'})` pour voir combien a déjà été facturé. Alors `(quantitéDéjàFacturée + quantitéFactureActuelle <= quantitéCommandée)` est la condition à vérifier.
  - Si quelqu'un tente de sur-facturer une ligne (par exemple, 5 sur 10 déjà facturés, et essaie maintenant de facturer 6 de plus), le script le signale.

- **Désaccord sur les conditions ou d'autres champs :** Le workflow de rapprochement à trois voies d'Oracle vérifie également si les conditions de paiement ou l'emplacement diffèrent entre la commande et la facture (Source: docs.oracle.com). Ces éléments peuvent ne pas être bloquants, mais ils pourraient indiquer une incohérence de saisie de données. Par exemple, si une commande était pour l'emplacement A mais que la facture est codée pour l'emplacement B, il s'agit peut-être simplement d'une erreur de saisie. Le script pourrait éventuellement avertir ou même corriger automatiquement certaines choses (avec prudence). L'entreprise pourrait ne pas se soucier d'un désaccord sur les conditions si elle négocie délibérément des conditions différentes sur la facture, mais c'est quelque chose à prendre en compte. Généralement, notre script se concentre sur les quantités et les montants, car ceux-ci ont un impact sur les finances.

Lors de la gestion de toute exception, la meilleure pratique est de la **rendre visible et assignable** :

- Définir un statut ou un champ clair sur la facture fournisseur (afin que quiconque la consulte puisse voir qu'elle est en cours d'examen en raison de problèmes de rapprochement).
- Enregistrer les détails (pourrait même joindre une note ou un sous-enregistrement listant les écarts trouvés).
- Notifier la partie appropriée (e-mail ou une recherche enregistrée sur un tableau de bord pour les factures non rapprochées).

En gérant de manière proactive ces cas limites, la solution automatisée garantit qu'elle ne gère pas seulement le « chemin heureux », mais qu'elle soutient également efficacement le personnel des comptes fournisseurs dans les « chemins malheureux » en les détectant tôt et en les acheminant correctement.

Intégration avec l'OCR et l'automatisation des comptes fournisseurs tiers (facultatif)

Bien que SuiteScript puisse automatiser la logique de rapprochement au sein de NetSuite, de nombreuses organisations exploitent également l'**OCR (Reconnaissance Optique de Caractères)** et des outils d'automatisation des comptes fournisseurs pour importer les factures dans NetSuite. L'intégration de ces solutions peut rationaliser davantage le processus :

- **NetSuite Bill Capture (OCR native) :** NetSuite offre une fonctionnalité de capture de factures intégrée appelée *Bill Capture*, qui utilise l'OCR et l'apprentissage automatique pour extraire des données des fichiers de factures (Source: stampli.com). Avec Bill Capture, les fournisseurs peuvent envoyer des factures par e-mail à une adresse désignée ou les utilisateurs peuvent télécharger des

numérisations ; le système crée ensuite une « Facture Fournisseur Numérisée » pour examen (Source: stampli.com). Lorsque l'utilisateur examine et crée l'enregistrement de facture fournisseur réel, une grande partie des données (fournisseur, articles, montants, et surtout la référence de la commande si trouvée sur la facture) est pré-remplie par l'OCR. Bill Capture tente de rapprocher la facture des commandes existantes dans le cadre de son processus de suggestion (Source: stampli.com) – par exemple, si la facture mentionne un numéro de commande existant, elle se liera à cette commande et remplira même automatiquement les lignes si possible.

Une fois que l'utilisateur approuve les suggestions de l'OCR, une facture fournisseur est créée dans NetSuite. À ce stade, notre automatisation de rapprochement SuiteScript intervient pour valider formellement la facture par rapport aux données de la commande. Essentiellement, Bill Capture élimine la saisie de données et fournit une première passe de liaison, tandis que SuiteScript garantit l'application des règles de rapprochement et des tolérances détaillées. La fonction Bill Capture de NetSuite, associée à un script de rapprochement personnalisé, peut fournir une solution de comptes fournisseurs de bout en bout : de la numérisation à la validation en passant par l'approbation.

Remarque : Bill Capture est un module complémentaire qui fournit l'extraction de données et un indicateur d'interface utilisateur de rapprochement à 3 voies de base, mais un routage complexe ou des tolérances personnalisées pourraient toujours nécessiter un script ou un workflow personnalisé. La combinaison de Bill Capture pour la saisie et de SuiteScript pour la logique de rapprochement personnalisée peut être très puissante.

- **Plateformes d'automatisation des comptes fournisseurs tierces :** De nombreux fournisseurs de SuiteApp (Stampli, Tipalti, Nanonets, MineralTree, etc.) proposent une automatisation avancée des comptes fournisseurs. Ceux-ci gèrent généralement la capture de factures (OCR), le codage, les workflows d'approbation, puis se synchronisent avec NetSuite. Par exemple, **Stampli** ou **Nanonets** peuvent extraire les données des factures et même effectuer un rapprochement à 2 ou 3 voies de leur côté, ne poussant dans NetSuite que les factures entièrement codées et validées (Source: nanonets.com). Si vous utilisez une telle plateforme, une partie de la logique de rapprochement pourrait se produire en dehors de NetSuite. Cependant, vous pourriez toujours implémenter SuiteScript pour vérifier ou gérer les cas limites lorsque les données arrivent.

  Certaines solutions tierces fournissent leurs propres bundles SuiteScript pour effectuer le rapprochement. Par exemple, il existe des SuiteApps spécifiquement conçues pour le rapprochement à 3 voies qui lient automatiquement les commandes et les réceptions aux factures en temps réel (un exemple est le moteur 3-Way Match de SquareWorks, ou la SuiteApp d'automatisation des comptes fournisseurs d'EchoVera (Source: squareworks.com)(Source: echovera.ca)). Si ceux-ci sont installés, un script personnalisé pourrait ne pas être nécessaire, ou votre script devra coexister avec eux.

- **Intégration OCR via RESTlet :** Si vous construisez une intégration OCR personnalisée, vous pourriez configurer un RESTlet dans NetSuite. Le système OCR externe appellerait ce RESTlet avec les données de la facture (fournisseur, date, montants, et peut-être une référence de pièce jointe PDF). Le RESTlet (qui utilise SuiteScript) pourrait alors créer une facture fournisseur (comme montré précédemment avec `record.create`) et effectuer la vérification de rapprochement immédiatement. Si vous le souhaitez, il pourrait même refuser de créer la facture en cas de désaccord complet, ou la créer avec un statut d'approbation en attente et les exceptions notées. Cette approche nécessite un développement des deux côtés mais vous donne un contrôle total.

En résumé, l'intégration d'outils d'OCR et d'automatisation des comptes fournisseurs peut considérablement réduire l'effort manuel d'importation des factures *dans* NetSuite. L'automatisation du rapprochement SuiteScript complète cela en garantissant qu'une fois la facture dans le système, elle est rigoureusement validée avant le paiement. Les entreprises visant des « comptes fournisseurs sans contact » pourraient combiner ces éléments : une facture arrive via OCR, est automatiquement saisie et rapprochée, et si tout est conforme, elle passe directement à la planification du paiement sans aucune intervention humaine. Seules les exceptions nécessiteraient une attention – ce qui est le résultat idéal de l'automatisation des comptes fournisseurs.

*(Aparté facultatif : Si vous implémentez une telle solution de bout en bout, assurez-vous également d'intégrer le côté **paiement** de manière appropriée – par exemple, utilisez SuiteScript ou SuiteFlow pour retenir la facture du paiement jusqu'à son approbation, et une fois approuvée, déclenchez peut-être le processus de paiement ou intégrez-vous à des solutions EFT/ACH.)*

Meilleures pratiques de journalisation des erreurs et de gestion des exceptions

Même avec un code bien écrit, des erreurs peuvent survenir – peut-être un enregistrement manquant, un champ contenant des données inattendues, ou une limite de gouvernance atteinte. Il est crucial que notre automatisation SuiteScript dispose d'une gestion des erreurs robuste pour éviter les échecs silencieux ou les transactions bloquées :

- **Blocs Try/Catch :** Enveloppez la logique principale dans des instructions try/catch pour intercepter toute exception d'exécution. Par exemple, lors du chargement d'un enregistrement avec `record.load`, si l'ID d'enregistrement est invalide, cela déclenchera une erreur – l'intercepter vous permet de la gérer (peut-être enregistrer un message spécifique « Commande non trouvée pour la facture X » et passer, plutôt que de faire planter tout le script). Dans un script Map/Reduce, les

erreurs dans l'étape de mappage pour un enregistrement n'arrêtent pas les autres, mais elles seront signalées dans le résumé. Vous pouvez également intercepter et enregistrer dans chaque mappage pour plus de clarté.

- **Utiliser N/error pour les exceptions personnalisées :** Le module `N/error` peut être utilisé pour créer des objets d'erreur significatifs (Source: docs.oracle.com). Par exemple, si une configuration cruciale est manquante (par exemple, aucun paramètre de tolérance défini), vous pourriez lever une erreur personnalisée : `throw error.create({name:'MATCH_CONFIG_ERROR', message:'Tolerance parameters not set', notifyOff: false});`. Le fait de la lever arrêtera le script (dans un script planifié ou UE) et indiquera clairement dans le journal pourquoi il s'est arrêté. Dans un Map/Reduce, lever une erreur dans une entrée de mappage pourrait marquer cette entrée comme ayant échoué mais continuer les autres.

- **Journalisation avec contexte :** Pendant que le script traite les factures, utilisez généreusement `log.debug` ou `log.audit` pour enregistrer ce qui se passe (Source: docs.oracle.com). Par exemple, après avoir rapproché une facture, enregistrez un audit : « Facture 1234 : Rapprochée OK » ou « Facture 1235 : Exception – écart de quantité ». Ces journaux aident au dépannage et servent également de trace historique dans les journaux d'exécution du script. Utilisez `log.error` pour enregistrer les exceptions interceptées ou les conditions inattendues. Étant donné que ces journaux peuvent être filtrés par niveau de journalisation, envisagez d'utiliser `DEBUG` pour les informations de routine et `AUDIT` pour les informations récapitulatives importantes.

- **Éviter les boucles infinies ou les redéclenchements :** Faites attention si votre script d'événement utilisateur met à jour l'enregistrement sur lequel il s'exécute (nous avons utilisé `submitFields` dans le pseudo-code pour mettre à jour un champ personnalisé). Cette mise à jour déclenchera, par défaut, à nouveau l'événement utilisateur (afterSubmit se déclenchera lors de la mise à jour du champ). Cela peut provoquer une boucle infinie. Pour éviter cela, les stratégies incluent :

  - Utiliser un indicateur statique : par exemple, définir un champ `custbody_matched_processed = true` après le traitement, et faire en sorte que le script vérifie `if(custbody_matched_processed) return;` au début.

  - Ou utilisez `context.executionContext` pour détecter si le script s'exécute dans l'interface utilisateur, en mode planifié, en tant que suitelet, etc. Si vous concevez le flux de manière à ce que l'événement utilisateur définisse un indicateur et que le script planifié le supprime, etc., vous pouvez éviter la récursion.

  - NetSuite dispose également d'un mécanisme dans SuiteScript 2.x où `context.newRecord` sur afterSubmit pour une opération `submitFields` pourrait ne pas contenir le champ (car ce n'est pas un chargement complet d'enregistrement). Cela devient délicat, donc souvent le plus simple est un indicateur de champ personnalisé ou un paramètre de déploiement temporaire.

- **Gestion des limites de gouvernance :** Dans les scripts planifiés, si vous prévoyez d'atteindre la limite de gouvernance (par exemple, vous devez traiter 1000 factures et chacune prend une certaine gouvernance), vous pourriez céder manuellement :
  - Vous pouvez utiliser `N/task.TaskType.SCHEDULED_SCRIPT` pour replanifier le même script et transmettre des paramètres indiquant où continuer (comme un index ou un ID interne pour reprendre).
  - Ou diviser le travail par pagination des résultats de recherche, en traitant par blocs à chaque invocation.
  - Cependant, comme discuté, l'utilisation de Map/Reduce rend cela largement inutile car il se replanifiera automatiquement si nécessaire (Source: docs.oracle.com).
- **Gouvernance des transactions :** Un script d'événement utilisateur s'exécute dans le contexte d'une transaction de sauvegarde d'enregistrement. S'il prend trop de temps ou échoue, il pourrait potentiellement annuler cette transaction (ce qui signifie que la facture fournisseur ne serait pas enregistrée). Il est généralement préférable que le script intercepte les exceptions et ne les laisse pas remonter dans un événement utilisateur, sinon les utilisateurs verront une fenêtre contextuelle d'erreur et la facture ne sera pas créée. Au lieu de cela, gérez les erreurs avec élégance : peut-être définir un champ « Erreur de rapprochement » avec le message d'erreur afin que l'utilisateur sache que quelque chose s'est mal passé, mais permettre tout de même l'enregistrement de la facture. Alors les comptes fournisseurs peuvent y remédier. Ce n'est que dans des cas vraiment critiques (comme un risque de corruption de données) que vous devriez lever une erreur non gérée dans un événement utilisateur.
- **Notifications d'échecs :** Si le script planifié ou map/reduce rencontre une erreur grave et ne peut pas terminer son exécution, assurez-vous que cela ne passe pas inaperçu. Vous pouvez configurer les déploiements de scripts pour envoyer une notification par e-mail en cas d'échec (il y a une case à cocher pour l'e-mail et les destinataires dans les paramètres de déploiement). De plus, dans les blocs catch, utilisez `N/email` pour alerter un administrateur si nécessaire. Par exemple, « Script de rapprochement des factures annulé en raison d'une erreur : ... ».
- **Tests et journalisation dans le bac à sable :** Avant de déployer en production, testez avec une variété de scénarios dans un environnement de bac à sable ou de prévisualisation de version. Utilisez la journalisation pour vérifier que le script identifie correctement les correspondances et les non-correspondances. Les tests devraient couvrir les cas limites : sur-facturation, sous-réception, limite de seuil de tolérance, plusieurs factures par commande, etc. À chaque fois, vérifiez que le résultat du script (approbation ou exception) est celui que vous attendez. Cela permettra de détecter toute erreur logique dans le script.

En suivant ces pratiques, vous garantissez que l'automatisation est fiable et transparente. Si quelque chose ne va pas, vous en aurez une trace et pourrez y remédier sans que les fournisseurs ne soient impayés de manière inattendue ou, inversement, que les factures ne soient payées sans rapprochement en raison d'une défaillance silencieuse du script. La gestion des exceptions consiste à anticiper ce qui peut mal tourner et à s'assurer que cela est géré de manière contrôlée – exactement ce qu'un bon système de rapprochement des factures devrait faire (après tout, l'objectif principal est de bien gérer les « exceptions » !).

Considérations de déploiement et de test

Le déploiement d'une solution de rapprochement des factures basée sur SuiteScript dans NetSuite nécessite une planification minutieuse pour minimiser les perturbations et garantir la précision. Voici quelques considérations finales pour le déploiement et les tests :

- **Activation des fonctionnalités :** Assurez-vous que les fonctionnalités NetSuite pertinentes sont activées. Pour le rapprochement à trois voies, le compte doit disposer de la fonctionnalité **Réception avancée** si vous utilisez les réceptions d'articles et la fonctionnalité *Rapprocher la facture de la réception* (si vous prévoyez de l'utiliser). Si vous utilisez les Approbations NetSuite (SuiteApp de workflow de rapprochement à 3 voies), décidez si vous l'utiliserez en tandem avec vos scripts ou si vous la désactiverez en faveur de la solution personnalisée pour éviter les conflits.
- **Tests en bac à sable :** Testez toujours les scripts dans un environnement de bac à sable avec une copie de données réelles. Simulez des scénarios réels :
    - Créez une commande test avec plusieurs lignes, recevez certains ou tous les articles, puis créez des factures fournisseurs qui correspondent, sur-facturent, sous-facturent, etc., et voyez comment le script réagit.
    - Testez les cas limites tels que : facture sans commande, facture avec un numéro de commande erroné (si votre script essaie de trouver des commandes par numéro, voyez comment il gère l'absence de correspondance), plusieurs factures pour une ligne de commande, etc.
    - Si possible, impliquez les utilisateurs finaux réels (comptables fournisseurs) dans les tests, afin qu'ils puissent fournir des commentaires sur le processus (par exemple, les notifications d'exception sont-elles pertinentes, les informations fournies sont-elles suffisantes pour agir, etc.).
- **Considérations de performance :** Pendant les tests, mesurez le temps que prennent les scripts. Si un script d'événement utilisateur ralentit de manière significative l'enregistrement d'une facture fournisseur (par exemple, en prenant plusieurs secondes), envisagez l'optimisation ou le

déchargement. Utilisez la SuiteApp Application Performance Management (APM) de NetSuite ou les journaux d'exécution du script pour voir le temps et si une partie du code est particulièrement lente. Assurez-vous que toute *Recherche Enregistrée* utilisée est optimisée (sélectionnez uniquement les champs nécessaires, etc.). Si la performance est un problème, vous pourriez réduire la portée de l'événement utilisateur et transférer davantage vers un script planifié.

- **Moment du déploiement :** Déployez le script planifié/MapReduce à un moment où il n'interférera pas avec la clôture financière ou le traitement quotidien intensif. NetSuite suggère de planifier les processus par lots en dehors des heures de pointe pour de meilleures performances (Source: docs.oracle.com). De plus, coordonnez-vous avec l'équipe financière – elle pourrait ne pas vouloir qu'un nouveau processus d'auto-approbation soit mis en service juste à la fin du mois. Idéalement, déployez après la fin d'une période, afin d'avoir le temps de surveiller et d'ajuster avant les cycles critiques.

- **Sécurité et autorisations :** Le déploiement du script s'exécutera sous un rôle spécifique (généralement le paramètre **Exécuter en tant que rôle** est Administrateur ou un autre rôle avec des autorisations complètes pour la facture fournisseur, la commande, la réception d'article, etc.). Assurez-vous que le rôle du script peut accéder à tous les types d'enregistrements et champs nécessaires (y compris les champs personnalisés pour les tolérances ou les indicateurs). Si vous prévoyez d'autoriser le personnel non-administrateur à initier ou à surveiller le processus, assurez-vous qu'il dispose des autorisations pour tous les enregistrements que le script touche (ou créez un tableau de bord/portlet personnalisé pour eux qui affiche les statuts).

- **Interaction avec les workflows/approbations :** Si SuiteApprovals est activé pour les factures fournisseurs, ou si un workflow est déjà en place pour les approbations, décidez comment le script s'intègre :

  - Une approche consiste à laisser le script *définir le champ de statut d'approbation* (qui est disponible si SuiteApprovals est activé). Par exemple, si le script définit le statut d'une facture fournisseur sur « Approuvée » (et que vous avez configuré le processus d'approbation en conséquence), il contourne efficacement l'approbation manuelle. Assurez-vous que cela est acceptable dans la politique de votre entreprise.

  - Alternativement, utilisez des champs personnalisés (comme Statut de rapprochement) et ajustez le workflow d'approbation pour approuver automatiquement si Statut de rapprochement = « Rapproché » et ne nécessiter une approbation manuelle que si « Exception ». La SuiteApp de rapprochement à 3 voies de NetSuite fonctionne de manière similaire, approuvant automatiquement les factures sans exception et ne routant que celles présentant des écarts (Source: docs.oracle.com).

  - Si vous utilisez la SuiteApp de rapprochement à 3 voies de NetSuite, notez ses **limites** (par exemple, il ne prend pas en charge les réceptions partielles, selon les notes d'Oracle (Source: docs.oracle.com)). Vous pouvez décider de ne pas l'installer et d'utiliser plutôt votre approche personnalisée pour éviter toute confusion. Ou si elle est installée pour d'autres raisons, désactivez éventuellement son action et utilisez uniquement la logique du script pour la cohérence.
- **Déploiement par phases :** Il pourrait être judicieux de déployer l'automatisation par phases. Par exemple, exécutez-le initialement en **mode journalisation uniquement** – où il n'approuve rien en réalité, mais enregistre ce qu'il *ferait* (et peut-être définit un champ non critique). Cela peut être fait en ne modifiant pas réellement les statuts, mais en enregistrant simplement les résultats. Laissez-le fonctionner pendant une semaine pour voir s'il se comporte comme prévu (par exemple, voir combien il approuverait automatiquement ou signalerait, et vérifier chaque décision manuellement). Après avoir confirmé qu'il prend les bonnes décisions, vous pouvez activer les actions d'auto-approbation. Cela réduit les risques.

- **Sauvegarde et récupération :** Puisqu'il s'agit d'une automatisation autour des transactions financières, ayez toujours un plan de contingence. Si le script échoue ou produit des résultats incorrects, ayez un plan pour le désactiver rapidement (décocher le déploiement, etc.) et revenir temporairement au processus manuel. Les déploiements de scripts NetSuite permettent de désactiver rapidement un script si nécessaire. De plus, maintenez un contrôle de version de votre code de script (au cas où un changement introduirait un bug, vous pouvez revenir en arrière).

- **Audit et journalisation en production :** Une fois en production, surveillez régulièrement les journaux du script, surtout pendant les premières semaines. Soyez attentif à toute exception inattendue ou à un nombre élevé de factures signalées – cela pourrait indiquer des problèmes de données ou que les paramètres de tolérance doivent être ajustés. Assurez-vous que toutes les exceptions signalées sont bien légitimes (pas de faux positifs). Cette période de réglage fin est cruciale pour instaurer la confiance dans l'automatisation.

- **Formation des utilisateurs :** Même s'il s'agit d'une « automatisation », les utilisateurs finaux (service des comptes fournisseurs) doivent être conscients de ce que fait le script. Formez-les aux nouveaux champs ou statuts sur la facture fournisseur (comme un champ Statut de correspondance), à ce que signifie l'approbation automatique d'une facture et à la manière dont ils seront informés des exceptions. Les utilisateurs doivent savoir que, par exemple, ils n'ont plus besoin de faire correspondre manuellement chaque facture au bon de commande (le système s'en charge), mais qu'ils doivent prêter attention aux e-mails d'exception ou à une recherche enregistrée des factures non appariées. Cela garantit une adoption en douceur.

- **Amélioration continue :** Après le déploiement, recueillez les commentaires. Le personnel des comptes fournisseurs pourrait dire : « Nous avons souvent une variance de prix de 5 % due aux fluctuations monétaires – pouvons-nous l'intégrer comme tolérance ? » ou « Nous aimerions que l'e-mail d'exception inclue également le numéro de bon de commande et le nom du fournisseur pour plus de clarté. » SuiteScript est flexible, alors intégrez ces commentaires pour affiner la solution. Restez également informé des nouvelles versions de NetSuite – de nouvelles fonctionnalités pourraient apparaître (par exemple, des améliorations de Bill Capture ou SuiteApprovals) qui pourraient compléter ou affecter votre script.

En couvrant ces considérations, vous contribuez à garantir que l'automatisation fonctionne non seulement en théorie, mais aussi en pratique, offrant un processus de rapprochement des factures fiable et efficace au sein de NetSuite.

Conclusion

L'automatisation du rapprochement des factures fournisseurs dans NetSuite à l'aide de SuiteScript 2.x peut considérablement rationaliser le processus des comptes fournisseurs en effectuant automatiquement des vérifications de rapprochement à deux et trois voies qui, autrement, nécessiteraient un effort manuel. Nous avons commencé par un aperçu des principes fondamentaux du rapprochement des factures – en distinguant le rapprochement à deux voies du rapprochement à trois voies et en soulignant pourquoi l'automatisation est précieuse pour réduire les erreurs et les coûts. Nous avons ensuite esquissé une architecture de solution où les scripts d'événements utilisateur et les scripts planifiés/MapReduce de SuiteScript orchestrent le flux de travail de rapprochement : chargement des bons de commande et des réceptions, comparaison avec les factures fournisseurs, et approbation automatique ou signalement des exceptions selon les règles métier.

Tout au long de ce rapport, nous avons approfondi les détails techniques précieux pour les développeurs NetSuite et les consultants ERP : quels modules SuiteScript utiliser et comment (de `N/record` pour la manipulation des enregistrements (Source: docs.oracle.com), à `N/search` pour trouver les enregistrements pertinents (Source: docs.oracle.com), à `N/runtime` pour les paramètres de configuration (Source: docs.oracle.com), et `N/email` pour les notifications (Source: docs.oracle.com)), et quels types de scripts conviennent le mieux aux différentes parties du processus. Nous avons fourni des extraits de code illustrant les principaux modèles d'implémentation – tels que la liaison d'une facture fournisseur à un bon de commande via un script (Source: blog.prolecto.com), et du pseudo-code pour la logique de rapprochement couvrant les comparaisons de base et la gestion des tolérances. Nous avons également discuté des stratégies pour écrire un code modulaire et maintenable, comme l'utilisation de modules de bibliothèque et la structuration du script pour la réutilisabilité (Source: docs.oracle.com).

De manière cruciale, nous avons exploré comment gérer les cas limites courants : des écarts de quantité et de prix aux réceptions manquantes et aux factures en double, garantissant que l'automatisation est robuste et couvre les scénarios du monde réel. Nous avons abordé l'intégration de l'OCR et des outils d'automatisation des comptes fournisseurs tiers, reconnaissant que la capture des factures est une étape en amont importante qui peut alimenter notre solution de rapprochement SuiteScript – que ce soit via le propre OCR Bill Capture de NetSuite (Source: stampli.com) ou des plateformes externes.

Enfin, nous avons abordé les meilleures pratiques en matière de gestion des erreurs (afin que l'automatisation échoue de manière élégante et transparente) et fourni des conseils sur le déploiement et le test de la solution dans un environnement NetSuite – en mettant l'accent sur les tests en sandbox, l'optimisation des performances, la formation des utilisateurs et l'amélioration itérative.

Avec une telle automatisation en place, une organisation peut atteindre un processus de comptes fournisseurs plus *« sans contact »* : les factures qui correspondent aux bons de commande et aux réceptions dans les tolérances autorisées sont automatiquement approuvées et peuvent passer au paiement, tandis que seuls les cas exceptionnels nécessitent une intervention manuelle (Source: netsuite.com). Cela conduit à un traitement plus rapide, moins d'erreurs et une gestion plus contrôlée des flux de trésorerie (Source: netsuite.com)(Source: netsuite.com).

Références : Les informations et exemples de ce rapport proviennent d'une combinaison de la documentation officielle et du centre d'aide d'Oracle NetSuite (pour l'utilisation de l'API SuiteScript et le flux de travail standard de rapprochement à 3 voies) (Source: docs.oracle.com)(Source: docs.oracle.com), des discussions NetSuite SuiteAnswers et de la Communauté (fournissant un contexte sur des scénarios commerciaux réels et des solutions SuiteScript), et de blogs et articles d'experts de la communauté NetSuite (illustrant des modèles de code et les meilleures pratiques en SuiteScript) (Source: blog.prolecto.com)(Source: docs.oracle.com). Ces sources sont citées tout au long du rapport pour offrir des lectures complémentaires et valider les approches discutées.

Étiquettes: netsuite, suitescript, rapprochement-factures, comptes-fournisseurs, automatisation, rapprochement-2-voies, rapprochement-3-voies, erp

---

À propos de 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.

AVERTISSEMENT