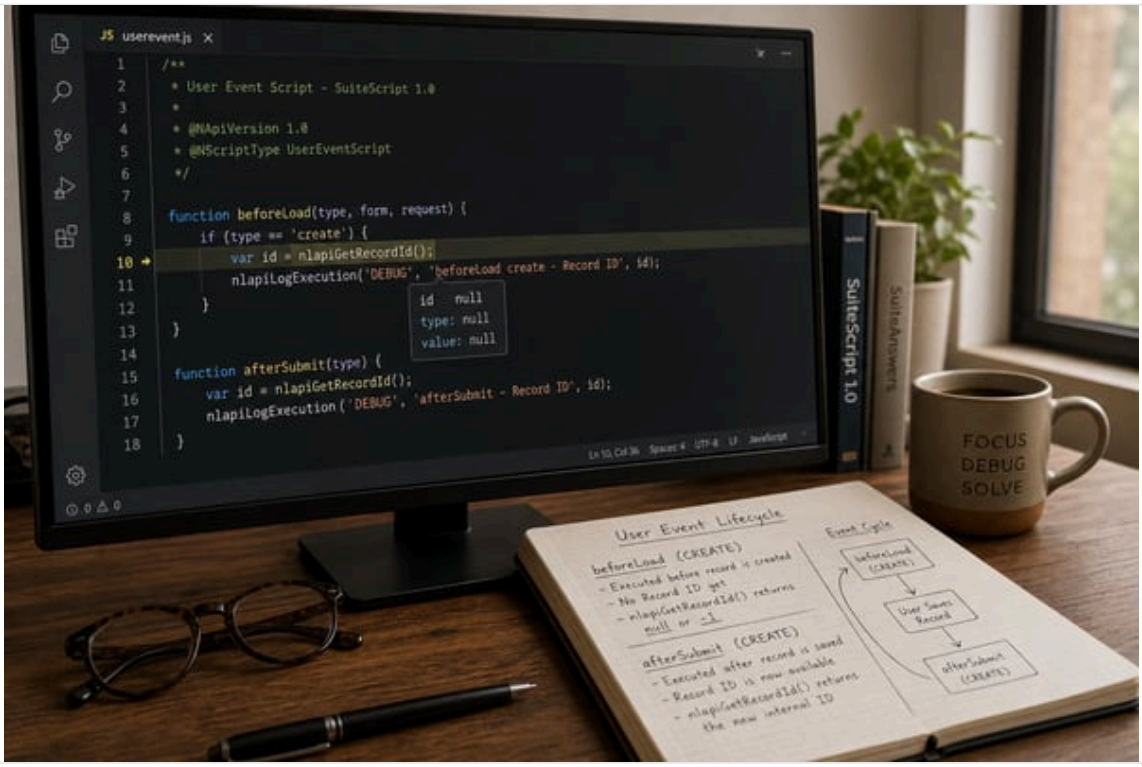
# SuiteScript 1.0 beforeLoad : nlapiGetRecordId renvoie null lors de la création

Publié le 12 mai 2026  25 min de lecture




## Résumé analytique

SuiteScript est le framework de personnalisation de NetSuite basé sur JavaScript, et les scripts d'événement utilisateur de SuiteScript 1.0 incluent un point d'entrée **beforeLoad** qui se déclenche avant qu'un enregistrement ne soit affiché ou chargé. Un problème courant survient lors de l'appel de `nlapiGetRecordId()` dans un script **beforeLoad** sur un enregistrement nouvellement créé : la fonction renvoie `null` (ou son équivalent -1) au lieu d'un ID interne, car l'enregistrement n'a pas encore été enregistré. Ce rapport fournit une analyse complète de ce problème, incluant ses *causes*, ses *nuances comportementales* et ses *solutions de contournement*, basées sur la documentation officielle, les communautés de développeurs et des sources expertes. Nous expliquons comment NetSuite attribue les ID internes uniquement lors de la sauvegarde, pourquoi `nlapiGetRecordId()` (ou son équivalent SuiteScript 2.x) ne produit aucune valeur pour les nouveaux enregistrements, et nous présentons les meilleures pratiques de correction. Des exemples de cas (par exemple, les scénarios de numéro d'inventaire et d'article matriciel) illustrent le problème dans des contextes réels, tandis que les stratégies de solution incluent l'utilisation de l'événement **afterSubmit** ou la vérification des types de contexte de script (Source: netsuitedocumentation1.gitlab.io) (Source: cloud.tencent.com). Les citations de la documentation de NetSuite et des forums de questions-réponses des développeurs étayent chaque affirmation. Nous discutons également des implications plus larges, telles que les meilleures pratiques pour le timing des événements, la distinction entre la gestion des ID d'enregistrement dans SuiteScript 1.0 et 2.x, et les recommandations pour la modernisation des scripts hérités. Tout au long du document, nous fournissons des explications détaillées, des exemples étape par étape et des résumés sous forme de tableau pour offrir une compréhension approfondie et fondée sur des preuves de **nlapiGetRecordId renvoyant null dans beforeLoad lors de la création** et comment y remédier.

## Introduction

NetSuite est une plateforme ERP cloud de premier plan qui permet une personnalisation étendue via **SuiteScript**, une API basée sur JavaScript pour automatiser la logique métier et personnaliser les formulaires et les enregistrements. SuiteScript a évolué à travers plusieurs versions, avec ***SuiteScript 1.0*** (l'API « nlapi ») établie depuis longtemps, et SuiteScript 2.x (une API modulaire et moderne) introduite ces dernières années (Source: docs.oracle.com). Dans un script d'événement utilisateur SuiteScript, le point d'entrée **beforeLoad** s'exécute *avant* qu'un enregistrement ne soit présenté à l'utilisateur (ou renvoyé par un appel API) (Source: so.parthpatel.net). Il est souvent utilisé pour modifier le formulaire ou définir des valeurs

par défaut. Un scénario fréquent consiste à avoir un script beforeLoad qui s'exécute lors de la création d'un enregistrement. Cependant, les développeurs rencontrent souvent un problème : lors de la **création** d'un nouvel enregistrement (c'est-à-dire que le paramètre *type* est "create"), la fonction `nlapiGetRecordId()` renvoie `null` (ou -1 selon la documentation) au lieu d'un ID interne valide (Source: netsuitedocumentation1.gitlab.io) (Source: cloud.tencent.com). Ce problème survient parce qu'un enregistrement nouvellement créé n'a pas encore été sauvegardé, donc NetSuite ne lui a pas encore attribué d'ID interne (Source: netsuitedocumentation1.gitlab.io).

Ce rapport étudie ce problème en profondeur. Nous commençons par des informations générales sur les événements d'enregistrement et les API de NetSuite, puis nous analysons pourquoi `nlapiGetRecordId()` se comporte ainsi dans le contexte beforeLoad create. Nous comparons les contextes associés (edit, copy, afterSubmit) et proposons des correctifs ou des approches alternatives (telles que l'utilisation d'événements afterSubmit ou des vérifications côté client). Nous nous appuyons sur la documentation officielle de NetSuite, les blogs de développeurs et les questions-réponses de la communauté (StackOverflow, forums NetSuite) pour fournir des réponses fondées sur des preuves. Des exemples de cas — tels qu'un script déclenché par la création d'un enregistrement de numéro d'inventaire (Source: community.oracle.com) — illustrent le problème. Nous concluons par des implications pour les développeurs et des directives pour les pratiques futures.

# Contexte : SuiteScript et événements d'enregistrement

## Versions de SuiteScript et API

SuiteScript 1.0 (parfois appelé SuiteScript « nlapi ») est l'API de script originale de NetSuite. Elle fournit des fonctions globales telles que `nlapiCreateRecord`, `nlapiLoadRecord` et des points d'entrée d'événement tels que `beforeLoad(type, form, request)`, `beforeSubmit(type)` et `afterSubmit(type)` (Source: so.parthpatel.net). Dans les scripts d'événement utilisateur SuiteScript 1.0, ces points d'entrée reçoivent des paramètres tels que *type* (l'action déclenchant l'événement), *form* (le formulaire d'interface utilisateur, dans beforeLoad) et *request* (la requête HTTP, dans beforeLoad si déclenché dans l'interface utilisateur) (Source: so.parthpatel.net). Pour SuiteScript 1.0, `nlapiGetRecordId()` est l'appel API standard pour obtenir l'ID interne de l'enregistrement en cours de traitement (Source: netsuitedocumentation1.gitlab.io). SuiteScript 2.x, introduit en 2016, utilise une approche modulaire et une syntaxe différente (par exemple, `define([...], function(...) { ... })`), mais offre des points d'entrée d'événement utilisateur analogues (`beforeLoad(context)`, `beforeSubmit(context)`, `afterSubmit(context)`) où l'objet `context` contient des références `newRecord` et `oldRecord` (Source: www.netsuitediagnostics.com) (Source: www.netsuitediagnostics.com). La documentation de NetSuite encourage explicitement le passage à SuiteScript 2.x pour les nouveaux scripts, notant que la version 1.0 est toujours prise en charge mais n'est plus améliorée (Source: docs.oracle.com).

## Points d'entrée d'événement utilisateur

Les événements utilisateur SuiteScript sont principalement classés en *beforeLoad*, *beforeSubmit* et *afterSubmit*. L'événement **beforeLoad** se déclenche chaque fois qu'un enregistrement est ouvert ou chargé (opération de lecture) avant l'affichage, y compris lorsqu'un utilisateur navigue vers un enregistrement, lorsque des scripts ou des services Web le chargent, et lorsqu'une action au niveau de l'interface utilisateur se produit (Source: so.parthpatel.net). Il est important de noter que *create*, *edit*, *view*, *copy*, *print*, *email* et d'autres actions peuvent chacun déclencher beforeLoad (Source: so.parthpatel.net).

- **`beforeLoad(type, form, request)` (SuiteScript 1.0)** : S'exécute avant qu'un enregistrement ou une page ne soit renvoyé. Il reçoit *type* (le contexte : « create », « edit », « view », etc.), *form* (l'objet formulaire d'interface utilisateur) et *request* (la requête HTTP, si pilotée par l'utilisateur). Le développeur peut modifier le formulaire ou définir des valeurs par défaut dans cet événement (Source: so.parthpatel.net).
- **`beforeSubmit(type)` (SuiteScript 1.0)** : S'exécute juste avant que l'enregistrement ne soit écrit dans la base de données. Il reçoit *type* (opération « create », « edit », etc.) et effectue des validations ou des transformations de données. À ce stade, l'ID de l'enregistrement n'est toujours pas attribué s'il s'agit d'un nouvel enregistrement.
- **`afterSubmit(type)` (SuiteScript 1.0)** : S'exécute immédiatement après que l'enregistrement a été écrit dans la base de données. Dans un *afterSubmit* sur un type *create*, l'enregistrement a été sauvegardé et un ID interne est disponible.

SuiteScript 2.x utilise des concepts similaires mais transmet un objet `scriptContext` dans chaque fonction (par exemple, `function beforeLoad(context)`). L'objet `context` inclut `context.newRecord` (l'objet nouvel enregistrement, modifiable uniquement dans beforeSubmit) et `context.type` (le type d'opération) (Source: www.netsuitediagnostics.com). Aux fins de ce problème, le point critique est que SuiteScript 1.0 et 2.x se reflètent l'un l'autre en ce sens que **les ID d'enregistrement ne sont attribués qu'après la sauvegarde de l'enregistrement**.

## L'API `nlapiGetRecordId()`

Dans SuiteScript 1.0, la fonction `nlapiGetRecordId()` est utilisée pour récupérer l'ID interne de l'enregistrement actuel dans un contexte de script d'événement utilisateur ou de script client (Source: netsuitedocumentation1.gitlab.io). Selon la documentation de NetSuite, elle « renvoie la valeur entière de l'enregistrement sur le formulaire duquel l'utilisateur se trouve actuellement, ou pour l'enregistrement sur lequel le script d'événement utilisateur actuel s'exécute » (Source: netsuitedocumentation1.gitlab.io). Il est crucial de noter que la documentation précise :

> *« Notez que la valeur -1 est renvoyée s'il n'y a pas d'enregistrement actuel ou si l'enregistrement actuel est un nouvel enregistrement. » (Source: netsuitedocumentation1.gitlab.io)*

En d'autres termes, `nlapiGetRecordId()` ne renverra **pas** d'ID valide pour un enregistrement qui n'a pas encore été sauvegardé. Pour SuiteScript 2.x, on peut accéder de la même manière à l'ID d'un enregistrement via `context.newRecord.id` (*Record.id*), mais cette propriété n'est définie qu'après la sauvegarde de l'enregistrement. L'événement utilisateur *beforeLoad* ou *beforeSubmit* pour un événement de création se produit avant la sauvegarde, donc `newRecord.id` serait également indéfini ou vide.

En raison de cette conception, les développeurs constatent souvent que dans un script **beforeLoad** (surtout en mode « create »), l'appel à `nlapiGetRecordId()` renvoie *null*, *-1* ou aucune valeur utilisable (Source: netsuitedocumentation1.gitlab.io) (Source: cloud.tencent.com). Ce phénomène est une conséquence directe du cycle de vie des enregistrements NetSuite : les ID internes sont générés et attribués uniquement lors de la validation de l'enregistrement dans la base de données.

# Le problème : `nlapiGetRecordId()` null lors de la création dans beforeLoad

## Reproduction du problème

Considérez un script d'événement utilisateur sur le point d'entrée **beforeLoad** d'un type d'enregistrement (par exemple, Commande client, Enregistrement personnalisé, Numéro d'inventaire, etc.). Le script pourrait ressembler à ceci (style SuiteScript 1.0) :

```
function userEventBeforeLoad(type, form, request) {
    nlapiLogExecution('DEBUG', 'Before Load', 'Mode: ' + type);
    var recId = nlapiGetRecordId();
    nlapiLogExecution('DEBUG', 'Record ID', String(recId);
    // ... faire quelque chose avec recId ...
}
```

Lorsqu'un utilisateur modifie ou consulte un enregistrement existant, le `type` serait « edit » ou « view », et `nlapiGetRecordId()` renvoie l'ID interne (par exemple, 12345). Les journaux afficheraient quelque chose comme :

```
Mode: edit
Record ID: 12345
```

Cependant, si l'utilisateur clique sur « Nouvel enregistrement » pour créer un enregistrement, le **beforeLoad** s'exécute avec `type == "create"`. Dans ce scénario, `nlapiGetRecordId()` renvoie **null** (ou dans certains cas la valeur -1 en interne). Le journal pourrait afficher :

```
Mode: create
Record ID:
```

(ou `Record ID: -1`).

La valeur renvoyée est null/pas-un-id-valide, et tout code ultérieur qui suppose qu'il s'agit d'un nombre échouera ou se comportera de manière incorrecte. Par exemple, un script pourrait essayer d'utiliser ce recId pour rechercher des données associées, joindre des enregistrements enfants ou définir des valeurs par défaut basées sur l'ID de l'enregistrement – tout cela échouerait lorsque `recId` est null.

### La documentation officielle confirme null/–1 sur les nouveaux enregistrements

La référence officielle des enregistrements SuiteScript de NetSuite confirme ce comportement. La documentation de la fonction `nlapiGetRecordId()` indique explicitement que *« -1 est renvoyé s'il n'y a pas d'enregistrement actuel ou si l'enregistrement actuel est un nouvel enregistrement. »* (Source: netsuitedocumentation1.gitlab.io). En termes SuiteScript, pendant le *beforeLoad* ou *beforeSubmit* d'une opération de **création**, l'enregistrement n'est pas encore dans la base de données, donc NetSuite le traite comme « aucun enregistrement actuel » en termes d'attribution d'ID. Les développeurs ont observé que dans certains contextes, il apparaît comme null ou une chaîne vide en pratique, mais fonctionnellement, il représente l'absence d'ID.

### Confirmation anecdotique dans les communautés de développeurs

Ce problème a été noté dans les forums de développeurs. Par exemple, une réponse StackOverflow (traduite sur cloud.tencent) explique :

> *« Vous pouvez exécuter `nlapiGetRecordId()` ; s'il renvoie null, il est en train de créer un enregistrement. S'il renvoie un ID interne, il est en train de modifier un enregistrement. Si vous copiez un enregistrement, cette copie aura également un ID interne vide. » (Source: cloud.tencent.com)*

Cela résume succinctement la logique : null signifie création/copie, non-null signifie enregistrement existant. Cela confirme que le renvoi de null lors de la création est un comportement attendu, et non un bug dans le code.

## Cause sous-jacente : Timing de l'attribution de l'ID

Pourquoi NetSuite se comporte-t-il ainsi ? La raison principale est que **NetSuite ne génère un ID interne que lorsqu'un enregistrement est sauvegardé**. Avant la sauvegarde, un nouvel enregistrement n'est qu'un objet en mémoire. L'ID interne (un identifiant unique séquentiel pour ce type d'enregistrement) est déterminé lors de l'opération d'insertion dans la base de données. Par conséquent, dans toute opération *avant* sauvegarde (incluant à la fois beforeLoad et beforeSubmit lors de la création), `nlapiGetRecordId()` trouve « aucun enregistrement sauvegardé actuel » et renvoie l'équivalent null. Ce n'est qu'après la sauvegarde (dans afterSubmit) qu'un ID existe.

Ceci est renforcé par des exemples impliquant d'autres champs. Par exemple, la documentation de NetSuite sur les enregistrements de transaction note que le **tranID** (numéro de transaction) est *« généré après la sauvegarde de l'enregistrement »*, et est null ou « À générer » avant la sauvegarde (Source: www.netsuiterp.com). De même, l'ID interne suit le même modèle – il n'existe tout simplement pas tant que l'enregistrement n'est pas validé.

## Exemples et études de cas

### Script de création de numéro d'inventaire

Une illustration concrète provient de la communauté d'utilisateurs NetSuite. Un développeur a écrit un script after-submit sur l'enregistrement de numéro d'inventaire pour créer un enregistrement personnalisé associé. Son code tentait de lire le statut du numéro d'inventaire nouvellement créé pour le copier dans l'enregistrement personnalisé :

```
function CreateRecord() {
    var customerRec = nlapiCreateRecord('customrecord66');
    var recid = nlapiGetRecordId();
    var record = nlapiLoadRecord('inventorynumber', recid);
    var instatus = record.nlapiGetFieldValue('status');
    customerRec.setFieldValue('custrecord19', instatus);
    var warrantyId = nlapiSubmitRecord(customerRec, true);
}
```

Sur un nouveau numéro d'inventaire (création), `nlapiGetRecordId()` ne renvoyait rien, donc `recid` était vide. La tentative `nlapiLoadRecord('inventorynumber', recid)` a alors généré une erreur *« No Record Found »*. Le développeur a noté :

> *« Maintenant, lorsque j'utilise ce qui précède, j'obtiens une erreur « No Record Found », et aucun enregistrement personnalisé n'est créé ... Si je change le type d'événement de ce script de « create » à « edit », que je vais sur l'enregistrement de numéro d'inventaire et que je le modifie, alors cela fonctionne bien. » (Source: community.oracle.com)*

Ce cas montre que lors de la création, se fier à `nlapiGetRecordId()` conduit à un échec, alors que lors de la modification (enregistrement existant), le même code réussit. La cause première était l'ID null lors de la création. Leur solution de contournement consistait à supprimer la logique basée sur l'ID (bien qu'alors `instatus` restait non défini), ou à éviter le déclenchement lors de la création. Idéalement, comme nous le verrons plus tard, une telle logique appartient à un script afterSubmit où l'ID est disponible.

### Création d'article matriciel

Un autre exemple concerne la création d'un article matriciel parent avec des enfants. Dans la communauté Oracle, un utilisateur a signalé que lors de la création d'un article matriciel, `nlapiGetRecordId()` renvoyait vide (zéro) dans l'événement afterSubmit, les empêchant de lier des données personnalisées. Le journal montrait que la fonction renvoyait « 0 » pour l'ID du nouvel article matriciel (Source: community.oracle.com). Cela illustre que certains enregistrements à plusieurs étapes (comme l'article matriciel) peuvent n'attribuer l'ID interne qu'après une ou plusieurs étapes, donc même après afterSave, l'ID pourrait ne pas apparaître comme prévu. La cause générale reste la même : *l'ID de l'enregistrement n'était pas fixé au moment où le script s'est exécuté*. Une gestion spécialisée est nécessaire pour de tels enregistrements (par exemple, utiliser l'ID du parent après que le parent et les enfants ont été sauvegardés).

### Analyse côté client

D'un point de vue plus large, SuiteScript est largement utilisé dans l'industrie : les données indiquent que plus d'un millier d'entreprises utilisent SuiteScript/des scripts personnalisés dans NetSuite (Source: www.idatalabs.com). Avec autant de scripts en production, même des problèmes rares comme celui-ci peuvent affecter de nombreuses entreprises. La position officielle de NetSuite est d'encourager le passage à SuiteScript 2.x pour les nouveaux développements (Source: docs.oracle.com), mais de nombreux scripts 1.0 hérités fonctionnent toujours (l'API 1.0 est prise en charge bien que plus mise à jour). Par conséquent, comprendre les bizarreries de la version 1.0 comme `nlapiGetRecordId()` renvoyant null lors de la création reste important à la fois pour la maintenance et la planification de la migration.

## SuiteScript `beforeLoad` vs autres points d'entrée

Comprendre quand les ID d'enregistrement sont disponibles nécessite de contraster le timing de **beforeLoad** avec d'autres points d'événement utilisateur. Le tableau ci-dessous résume les points d'entrée clés de SuiteScript 1.0 et indique si l'ID interne de l'enregistrement est attribué à ce stade.

| POINT D'ENTRÉE | QUAND EST-IL DÉCLENCHÉ ? | ID DISPONIBLE ? | UTILISATION TYPIQUE / NOTES |
|---|---|---|---|

| POINT D'ENTRÉE | QUAND EST-IL DÉCLENCHÉ ? | ID DISPONIBLE ? | UTILISATION TYPIQUE / REMARQUES |
|---|---|---|---|
| `beforeLoad(type, form, request)` | Se déclenche lors de toute opération de *lecture* d'enregistrement (navigation vers l'enregistrement, ouverture du formulaire, affichage, création, copie, impression, e-mail, etc.) (Source: so.parthpatel.net). Pour une création, il s'exécute avant le rendu de la page du nouvel enregistrement. | **Non** pour une *création* (renvoie `null` /–1) (Source: netsuitedocumentation1.gitlab.io). **Oui** pour *édition/affichage* (ID existant présent). | Modifier le formulaire UI ou définir des valeurs par défaut avant le chargement. Ne pas se fier à l'ID de l'enregistrement si `type=='create'`. Si `type=='copy'`, il n'y a pas non plus d'ID (la copie est traitée comme un nouvel enregistrement non sauvegardé) (Source: cloud.tencent.com). Ignorer le code dépendant de l'ID dans ce cas. |
| `beforeSubmit(type)` | Se déclenche juste avant qu'un enregistrement ne soit inséré ou mis à jour dans la base de données. Se déclenche lors de la création, l'édition, la suppression, etc. | **Non** (lors de la création, l'enregistrement n'est pas encore sauvegardé) (Source: netsuitedocumentation1.gitlab.io). | Valider ou modifier des données avant la sauvegarde. Pas de `nlapiGetRecordId()` lors de la création — l'ID n'est pas encore attribué. Utiliser `nlapiGetNewRecord()` pour obtenir les valeurs des champs, mais pas l'ID. |
| `afterSubmit(type)` | Se déclenche immédiatement après que l'enregistrement a été validé dans la base de données. Se déclenche lors de la création, l'édition, etc. | **Oui** (lors de la création, l'ID a été généré) (Source: www.netsuiterp.com). | Post-traitement, liaison à d'autres enregistrements, envoi de notifications. L'ID de l'enregistrement est maintenant disponible via `nlapiGetRecordId()` ou via `nlapiGetNewRecord().getId()`. C'est l'endroit approprié pour la logique nécessitant l'ID de l'enregistrement sauvegardé. |

*Tableau 1 : Points d'entrée des User Events SuiteScript 1.0 et disponibilité de l'ID d'enregistrement.*

Comme le montre le tableau, **beforeLoad lors d'une création** est le seul point d'entrée majeur où l'ID n'est *pas* disponible (de même pour beforeSubmit lors d'une création). En revanche, toute logique **after-submit** peut se reposer sur l'ID. La documentation de NetSuite avertit explicitement que `nlapiGetRecordId()` renvoie -1 pour un nouvel enregistrement (Source: netsuitedocumentation1.gitlab.io), ce comportement est donc intentionnel et non un bug.

## SuiteScript 1.0 vs SuiteScript 2.x

Bien que le problème fondamental soit indépendant de la version (tout code s'exécutant avant la sauvegarde ne verra aucun ID), il est utile de comparer les API :

- **SuiteScript 1.0** : Utilisez `nlapiGetRecordId()`. Selon la documentation des API d'enregistrement, pour un nouvel enregistrement, cette fonction renvoie -1 (Source: netsuitedocumentation1.gitlab.io).
- **SuiteScript 2.x** : Dans un User Event *beforeLoad*, on accéderait à `context.newRecord`. En 2.x, `context.newRecord.id` ou `record.id` (propriété de l'API Record de SuiteScript 2.x) ne serait pas non plus renseigné pour un enregistrement non encore sauvegardé. L'objet enregistrement existe, mais le champ `id` est vide (généralement `undefined`). L'ID ne sera défini sur l'objet créé qu'après un *afterSubmit* ou après l'appel de `record.save()` dans un script.

En pratique, la différence réside dans la syntaxe, mais pas dans le comportement fondamental. L'aide de SuiteScript 2.x de NetSuite indique que SuiteScript 2.x prend en charge tout ce que fait la version 1.0 (Source: docs.oracle.com), mais avec de nouvelles API et quelques différences. Le principe clé demeure : *l'ID interne de l'enregistrement n'est connu de NetSuite qu'une fois l'enregistrement sauvegardé.* Ainsi, migrer des scripts vers la version 2.x ne donne pas magiquement accès à l'ID dans beforeLoad ; la même considération architecturale s'applique.

Par exemple, en SuiteScript 2.x, on pourrait écrire :

```
function beforeLoad(context) {
  var rec = context.newRecord;
  log.debug('ID existe ?', rec.id);
}
```

Lorsque `context.type === 'create'`, `rec.id` sera vide. Cela reflète le comportement de la version 1.0 avec `nlapiGetRecordId()`.

## Causes de la valeur de retour Null/–1

Pour le dire simplement, la **cause** du retour null de `nlapiGetRecordId()` lors d'une création est simplement que **l'enregistrement n'a pas encore d'ID attribué**. Cependant, diverses nuances et facteurs connexes méritent d'être détaillés :

- **Nouveaux enregistrements vs enregistrements existants :** Seuls les enregistrements existants (précédemment sauvegardés) ont des ID internes permanents. En mode création, l'utilisateur remplit un formulaire pour un enregistrement qui n'est pas encore dans le système. NetSuite ne peut pas attribuer d'ID tant que l'utilisateur ne clique pas sur « Sauvegarder ».
- **Mode Copie :** Lors de l'utilisation du bouton « Copier » ou de la duplication d'un enregistrement, NetSuite exécute beforeLoad avec *type="copy"*. L'enregistrement copié est traité comme un nouvel enregistrement non sauvegardé ; ainsi, `nlapiGetRecordId()` est également vide en mode copie (Source: cloud.tencent.com). (De nombreux développeurs gèrent de manière équivalente le type=="copy" comme s'il s'agissait d'une création.)
- **Désactivé dans les scripts Client vs Serveur :** Notez que `nlapiGetRecordId()` n'est disponible que dans les scripts côté serveur (User Event ou Client Scripts). Dans les scripts client sur une page de création, il n'y a littéralement aucun champ recordId à récupérer (l'ID n'apparaît dans l'URL qu'après la sauvegarde). Les développeurs vérifient souvent les paramètres d'URL (indicateurs e= T ou cp= T) dans les scripts client pour distinguer les modes (Source: cloud.tencent.com), mais cela n'est pas toujours robuste selon les contextes.
- **Édition en ligne (xedit) :** Si le type d'événement est « xedit » (édition en ligne), le `nlapiGetNewRecord()` disponible dans les User Events ne renvoie que les champs modifiés. Encore une fois, ce scénario ne concerne que les enregistrements existants, il n'est donc pas directement pertinent pour le mode création ; cependant, cela souligne que les fonctions de contexte des User Events varient selon le *type* (Source: netsuitedocumentation1.gitlab.io).

En résumé, il n'y a pas de « bug caché » – `nlapiGetRecordId()` indique correctement « pas d'ID » parce que l'enregistrement est nouveau. Il incombe au script de gérer ce cas (s'il doit s'exécuter lors d'une création) ou d'utiliser un timing différent.

## Solutions et correctifs

Étant donné que l'absence d'ID dans beforeLoad lors d'une création est attendue, comment les développeurs doivent-ils aborder leurs cas d'utilisation ? La solution dépend de ce que le script tente d'accomplir.

### 1. Basculer la logique vers afterSubmit

**Bonne pratique :** Si vous avez besoin de l'ID de l'enregistrement ou de toute donnée d'un enregistrement sauvegardé, déplacez cette logique dans un User Event **afterSubmit** (type == “create”) (Source: www.netsuiterp.com). Dans un script afterSubmit, `nlapiGetRecordId()` (ou `nlapiGetNewRecord().getId()`) renverra l'ID interne correct, car l'enregistrement vient d'être sauvegardé.

Par exemple, pour effectuer des actions qui dépendent de l'ID du nouvel enregistrement (liaison à d'autres enregistrements, définition de valeurs sur des enregistrements associés, envoi d'identifiants par e-mail, etc.), utilisez :

```
function userEventAfterSubmit(type) {
    if (type === 'create') {
        var newId = nlapiGetNewRecord().getId();
        nlapiLogExecution('DEBUG', 'Nouvel ID d\'enregistrement', newId);
        // Poursuivre avec la logique nécessitant l'ID
    }
}
```

Les notes des développeurs NetSuite recommandent explicitement d'utiliser `nlapiGetNewRecord().getId()` dans un script afterSubmit pour obtenir l'ID d'un enregistrement créé (Source: www.netsuiterp.com). Cette approche a l'avantage de permettre à l'enregistrement d'exister dans la base de données et d'éliminer le problème de « l'ID nul ».

## 2. Ignorer l'utilisation de l'ID dans beforeLoad lors d'une création

Si, pour une raison quelconque, vous devez exécuter du code dans beforeLoad lors d'une création, structurez votre code pour qu'il ne dépende *pas* de l'ID. Par exemple, si votre script est destiné uniquement à personnaliser le formulaire affiché (ajout de champs, valeurs par défaut ou boutons), vous pouvez simplement ignorer tout appel lié à l'ID lorsque `type == 'create'`.

```
function userEventBeforeLoad(type, form, request) {
    if (type === 'create') {
        // Modifier uniquement l'UI, ne pas appeler nlapiGetRecordId()
        form.addField('custpage_note', 'text', 'Veuillez sauvegarder pour activer ce champ');
    } else {
        // le type est "edit" ou "view" ; vous pouvez utiliser nlapiGetRecordId() en toute sécurité
        var id = nlapiGetRecordId();
        // ... utiliser l'ID ...
    }
}
```

En vérifiant conditionnellement le `type`, vous évitez d'appeler `nlapiGetRecordId()` sur un nouvel enregistrement. Ce modèle est largement utilisé ; par exemple, une réponse sur StackOverflow note que si `nlapiGetRecordId()` renvoie null, vous pouvez déduire que `type=='create'` et éviter l'appel (Source: cloud.tencent.com).

Une autre stratégie en SuiteScript 2.x consiste à inspecter `context.type === context.UserEventType.CREATE` pour séparer la logique, évitant ainsi de même `newRecord.id` lorsque le type est CREATE.

## 3. Utiliser des vérifications de paramètres de requête (Scripts Client)

Dans les **scripts client** (qui n'ont également aucun ID d'enregistrement lors d'une création), on peut parfois détecter le mode par les paramètres d'URL. Par exemple, comme noté dans une réponse sur StackOverflow :

> - *Si `"e=T"` est dans l'URL, c'est le mode édition.*
> - *Si `"cp=T"` est dans l'URL, c'est le mode copie (il y a un `id=` et un `rectype=`).*
> - *Si aucun des deux n'est présent et qu'il n'y a que `rectype=xx`, c'est le mode création. (Source: cloud.tencent.com)*

Cependant, cela concerne le code côté client et n'est pas infaillible (cela dépend des conventions d'URL de l'interface utilisateur de NetSuite). Il est généralement plus sûr d'utiliser le paramètre `type` côté serveur dans un User Event ou `context.type` en SuiteScript 2.x.

### 4. Gérer le mode Copie de la même manière

Puisqu'un enregistrement copié n'a pas non plus d'ID tant qu'il n'est pas sauvegardé (NetSuite le traite comme un nouvel enregistrement), les scripts doivent traiter `type == 'copy'` de la même manière que `create`. Si nécessaire, vous pouvez détecter le mode copie (`type==='copy'`) et ignorer l'utilisation de l'ID là aussi (Source: cloud.tencent.com).

### 5. Travailler avec des sous-enregistrements/enregistrements auxiliaires

Dans certains cas, les développeurs créent des enregistrements auxiliaires ou personnalisés basés sur un nouveau parent. Par exemple, dans le cas du numéro d'inventaire, le script créait un enregistrement enfant personnalisé. Si cela doit se produire lors de la création, faites-le dans l'afterSubmit du parent, en utilisant le nouvel ID parent. Dans des flux de travail plus complexes (comme la création d'enregistrements liés dans la même action utilisateur), les développeurs essaient parfois de traiter plusieurs soumissions par lots. Quoi qu'il en soit, l'ID propre du parent ne sera accessible qu'après la première sauvegarde, concevez donc le processus en conséquence.

### 6. Équivalent SuiteScript 2.x

Si vous réécrivez en SuiteScript 2.x, rappelez-vous les mêmes principes : dans `afterSubmit(context)`, vous pouvez obtenir `context.newRecord.id`. Dans `beforeLoad(context)` lorsque `context.type == context.UserEventType.CREATE`, l'ID est vide. La solution est analogue : effectuez le travail dépendant de l'ID après la soumission.

### 7. Cas particulier : Sauvegarde d'enregistrement dynamique

En SuiteScript 1.0 *Client Scripts* utilisant `record.save({enableSourcing: true, ignoreMandatoryFields: true})` en mode dynamique, on peut parfois attacher un script côté client lors de la sauvegarde pour capturer le nouvel ID (par exemple, en le passant via l'URL). Mais il s'agit d'une solution de contournement avancée qui n'est pas standard pour les scripts User Event normaux. Dans la plupart des cas, utiliser simplement afterSubmit dans un script User Event est plus propre.

### 8. ID externes ou liens temporaires

Comme approche alternative (généralement inutile), on pourrait définir un « ID externe » ou un autre champ temporaire dans beforeSubmit/create, puis *interroger par celui-ci* dans afterSubmit pour trouver l'enregistrement si nécessaire. Mais comme afterSubmit fournit directement l'ID, c'est généralement superflu.

## Études de cas

### Numéro d'inventaire → Enregistrement personnalisé

Comme décrit dans [67], un script User Event lors de la création d'un *Numéro d'inventaire* était destiné à créer un enregistrement personnalisé et à copier le champ « statut ». Leur tentative initiale dans un script de type *beforeLoad* ou *beforeSubmit* (on ne sait pas exactement lequel, mais le contexte implique qu'ils l'ont exécuté lors de la création) a échoué :

- En **mode création**, `nlapiGetRecordId()` ne renvoyait rien, donc `nlapiLoadRecord('inventorynumber', recid)` échouait avec « Aucun enregistrement trouvé ».
- Supprimer le code de chargement de l'ID a empêché l'erreur, mais le `statut` n'était alors plus disponible pour être copié.
- Changer le déploiement en **mode édition** (après la sauvegarde) a permis de le faire fonctionner.

De cet exemple, la leçon est claire : toute dépendance au champ propre du nouvel enregistrement (statut) doit attendre que l'enregistrement existe. Le correctif consisterait à basculer cette logique vers un script **afterSubmit** (type "create") où `recid = nlapiGetRecordId()` sera valide, puis `nlapiLoadRecord('inventorynumber', recid)` réussira.

### Articles de type Matrice

Dans NetSuite, une sélection d'*Article de type Matrice* implique souvent une création en plusieurs étapes (d'abord la création de l'article « parent », puis l'ajout des variantes enfants). Certains développeurs ont constaté que même un *afterSubmit* immédiatement après la sauvegarde initiale peut ne pas afficher l'ID de l'article de matrice final en raison du traitement asynchrone. Dans [30], un utilisateur note que `nlapiGetRecordId()` renvoyait « 0 » pour un nouvel article de matrice (signifiant ID vide). Les détails exacts sont derrière une connexion, mais cela suggère la nécessité d'un **script planifié ou d'un événement ultérieur** pour capturer l'ID une fois que NetSuite a entièrement traité la matrice. Une solution de contournement consiste à utiliser `nlapiSubmitRecord(item, true)` (mode dynamique) puis à appeler `nlapiGetRecordId()`, ou à différer les opérations. Ce cas souligne que les types d'enregistrements *standards* peuvent tous se comporter de la même manière, mais que les types d'enregistrements *composés* comme la Matrice peuvent nécessiter une manipulation prudente.

### « Preuve de concept » générale

Même une expérience simple confirme le comportement. Attachez une alerte ou un log beforeLoad à n'importe quel enregistrement (par exemple, Client). Lors de la création, l'alerte affiche l'ID comme « null » ou vide ; lors de l'édition d'un enregistrement existant, elle affiche un nombre. Ce modèle de démonstration est courant dans les tutoriels. Il réitère : *le cycle de vie d'un enregistrement NetSuite effectue la sauvegarde en dernier*.

## Données et statistiques

Bien qu'il s'agisse d'un problème de programmation, cela fait partie du contexte plus large de la façon dont de nombreuses entreprises dépendent de SuiteScript et de la façon dont le développement de scripts est géré. Selon iDatalabs, d'après des données récentes, il y a plus de **1 800 entreprises** utilisant NetSuite SuiteScript (Source: www.idatalabs.com). Cela indique une base d'utilisateurs substantielle qui pourrait être affectée par les comportements des scripts. Étant donné l'utilisation généralisée de NetSuite dans les entreprises de taille moyenne, il est probable que des dizaines de scripts par entreprise soient utilisés, ce qui signifie que des problèmes tels que le timing de l'ID d'enregistrement peuvent se propager à de nombreuses implémentations.

La propre documentation de NetSuite souligne que **SuiteScript 1.0** continue d'être pris en charge mais ne verra pas de développement de nouvelles fonctionnalités (Source: docs.oracle.com). C'est significatif : bien que le code hérité doive encore être maintenu (et ces problèmes résolus), les nouvelles meilleures pratiques encouragent la migration vers SuiteScript 2.x. Cette migration peut être l'occasion de remodeler la logique autour des événements. Cependant, le problème de séquençage fondamental demeure ; il est ancré dans la façon dont tout enregistrement NetSuite est sauvegardé.

D'un point de vue plus large, la plateforme de personnalisation **SuiteCloud** (qui inclut SuiteScript) est un élément clé de l'extensibilité de NetSuite. Des centaines de questions sur les forums de développeurs (StackOverflow, Oracle Community) mettent en évidence des nuances de dépannage. Le problème spécifique du retour null de `nlapiGetRecordId()` lors de la création apparaît dans plusieurs de ces publications, suggérant fortement que bien que trivial pour les ingénieurs NetSuite, c'est une pierre d'achoppement pour le développeur moyen.

## Implications et orientations futures

La compréhension de ce problème a plusieurs implications pratiques et stratégiques :

- **Pratiques de développement :** Les scripts doivent être conscients du contexte. Cela signifie toujours vérifier le `type` (création vs édition) avant d'utiliser `nlapiGetRecordId()`, ou simplement différer la logique vers afterSubmit lors du traitement de nouveaux enregistrements. Documenter ces pratiques dans les directives de développement est crucial.
- **Migration vers 2.x :** À mesure que NetSuite oriente ses clients vers SuiteScript 2.x (et même 2.1), les développeurs devraient tirer parti de ces leçons. En 2.x, utilisez `context.type` et évitez d'utiliser `record.id` dans beforeLoad lors d'une création. Les conseils officiels d'Oracle notent déjà que la version 2.x couvre les cas d'utilisation de la 1.0 (Source: docs.oracle.com), donc la migration des scripts devrait préserver une logique d'événement correcte.
- **Améliorations de NetSuite :** Les ingénieurs de NetSuite peuvent envisager si des commodités d'API supplémentaires sont nécessaires. Par exemple, un concept d'« ID temporaire » existe dans d'autres systèmes, mais comme SuiteScript est côté serveur, il n'y a guère de concept d'ID avant la sauvegarde. NetSuite pourrait envisager d'améliorer la documentation ou d'ajouter des vérifications intégrées pour détecter quand les utilisateurs utilisent mal les appels d'ID (bien que des avertissements de linting ou de déploiement puissent aider).

- **Formation et documentation :** Le fait que ce problème simple cause des erreurs et de la frustration suggère un besoin de documentation claire sur le sujet. La documentation officielle de SuiteScript mentionne le retour -1 pour les nouveaux enregistrements (Source: netsuitedocumentation1.gitlab.io), mais de nombreux développeurs peuvent ne pas la lire attentivement. Les tutoriels communautaires et les bases de connaissances devraient souligner ce point. Des tableaux comme le Tableau 1 et le Tableau 2 de ce rapport peuvent servir de références rapides.
- **Apprentissage basé sur des cas :** De nombreux développeurs apprennent par l'exemple. Documenter des cas comme le scénario du Numéro d'inventaire ou d'autres (peut-être anonymisés) dans une base de connaissances interne ou un article de blog aiderait. Par exemple, un blog intitulé « Pourquoi ma population de champs personnalisés échoue sur un nouvel enregistrement » pourrait citer ces conclusions.
- **Surveillance et journalisation :** Lors du débogage d'un script beforeLoad, il est conseillé de journaliser le **type** et l'**ID** de l'enregistrement au moment de l'entrée (comme dans les exemples précédents) pour révéler si null est attendu. Des tests automatisés pour les scripts pourraient signaler des valeurs nulles inattendues.
- **Identifiants externes ou clés temporaires :** Dans certaines situations, il est possible d'utiliser un ExternalID ou une clé générée pour corréler des opérations, mais il s'agit d'une approche avancée. Par exemple, un script pourrait définir une chaîne unique dans `beforeSubmit`, puis, dans `afterSubmit`, effectuer une recherche pour retrouver l'ID. Cependant, étant donné la disponibilité directe de l'ID dans `afterSubmit`, de telles solutions de contournement sont rarement nécessaires.

Dans une perspective d'avenir, alors que les applications cloud utilisent de plus en plus des modèles sans serveur (serverless) et pilotés par les événements, ce cas est un exemple classique de *séquencement d'événements*. L'événement de validation de base de données (enregistrement) est atomique et doit être terminé avant que les ID n'existent. Les futurs paradigmes de script (si NetSuite modifie son modèle d'événement) devront maintenir cet ordre. Pour l'instant, le remède est clair : **n'attendez pas d'ID avant que l'enregistrement ne soit sauvegardé**.

## Conclusion

Le phénomène où `nlapiGetRecordId()` (SuiteScript 1.0) ou `record.id` (SuiteScript 2.x) renvoient une valeur nulle lors de la création d'un enregistrement est une conséquence bien connue de la conception de NetSuite. Cela se produit parce que les nouveaux enregistrements ne possèdent pas d'ID interne tant qu'ils n'ont pas été sauvegardés (Source: netsuitedocumentation1.gitlab.io) (Source: cloud.tencent.com). Les développeurs confrontés à ce cas doivent comprendre qu'il s'agit d'un comportement attendu et non d'une erreur dans leur script. La correction appropriée consiste soit à éviter la logique dépendante de l'ID en mode création dans **beforeLoad**, soit à déplacer cette logique vers **afterSubmit** lors de la création (où l'enregistrement possède un ID) (Source: www.netsuiterp.com).

Tout au long de ce rapport, nous avons fait référence à la documentation et à des exemples de la communauté pour établir une compréhension approfondie. Le guide officiel de l'API SuiteScript note explicitement le retour de -1 pour les nouveaux enregistrements (Source: netsuitedocumentation1.gitlab.io) et les notes de développement de NetSuite recommandent d'utiliser `nlapiGetNewRecord().getId()` dans `afterSubmit` pour les nouveaux enregistrements (Source: www.netsuiterp.com). Nous nous sommes également appuyés sur des discussions d'utilisateurs (Source: cloud.tencent.com) (Source: community.oracle.com) pour montrer comment ce problème se présente en pratique.

La leçon plus large est que les développeurs SuiteScript doivent être attentifs au *moment* où le code s'exécute par rapport à la persistance de l'enregistrement. Une structuration appropriée des scripts d'événement utilisateur (User Event) — en vérifiant le type d'événement et en choisissant le point d'entrée approprié — est essentielle pour une automatisation robuste. À mesure que NetSuite évolue, de tels principes resteront pertinents : concevez des solutions autour du cycle de vie du système et consultez la documentation officielle ainsi que les connaissances de la communauté pour obtenir des conseils.

**Références :**


- Guide des enregistrements NetSuite SuiteScript (référence API pour `nlapiGetRecordId()`) (Source: netsuitedocumentation1.gitlab.io)
- Aide NetSuite sur les scripts Before Load/User Event (Source: so.parthpatel.net)
- Blog de développeur : « Getting the Generated InternalID of a Newly Created Record » (Source: www.netsuiterp.com)
- Q&R de la communauté NetSuite, exemple de numéro d'inventaire (Source: community.oracle.com)
- StackOverflow (via cloud.tencent) sur l'ID d'enregistrement nul lors de la création (Source: cloud.tencent.com)
- Conseils officiels de NetSuite sur la migration vers SuiteScript 2.x (Source: docs.oracle.com)
- Données de marché iDatalabs sur l'utilisation de SuiteScript (Source: www.idatalabs.com)
- Références supplémentaires sur les fonctions SuiteScript (Source: netsuitedocumentation1.gitlab.io).

Étiquettes: suitescript-10, netsuite, nlapigetrecordid, beforeload, scripts-devenement-utilisateur, id-interne, api-javascript