eBook Gratuit

APPRENEZ netsuite

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#netsuite

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: netsuite

It is an unofficial and free netsuite ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official netsuite.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec netsuite

Remarques

NetSuite est une plate-forme de gestion ERP, CRM, E-Commerce et Services professionnels basée sur le cloud. Plus de 30 000 entreprises l'utilisent pour gérer l'ensemble de leurs activités.

NetSuite est entièrement personnalisable par les administrateurs et les développeurs, notamment via une API basée sur JavaScript appelée SuiteScript. Les développeurs peuvent écrire des scripts déclenchés par divers événements dans le système NetSuite pour automatiser les processus métier.

Où obtenir de l'aide

1. Rejoignez la communauté NetSuite Professionals Slack, où vous avez un accès instantané à plus de 200 professionnels NetSuite à travers le monde.
2. Utiliser le navigateur d'enregistrements NetSuite pour le schéma de tous les types d'enregistrement
3. Guide de référence JavaScript de Mozilla Developer Network

Versions

Version	Date de sortie
2016.2	2016-09-20

Examples

Configuration de l'IDE Eclipse SuiteCloud

1. Téléchargez et installez le dernier IDE Eclipse
    - Installez Eclipse de deux manières:
        1. Installateur Eclipse
        2. Téléchargez le zip pour votre forfait préféré
    - Si vous ne possédez pas déjà un package Eclipse préféré, *Eclipse for JavaScript Developers* est recommandé
2. Installer le plug-in IDE SuiteCloud
    1. Une fois l'installation terminée, lancez Eclipse
    2. Accédez à *Aide > Installer un nouveau logiciel ...*
    3. Cliquez sur *Ajouter ...* pour ajouter un nouveau site de mise à jour
        - **Nom** : SuiteCloud IDE
        - **Emplacement** : http://system.netsuite.com/download/ide/update_e4
            - **Remarque:** l'emplacement dépend de la version de NetSuite sur laquelle

vous vous trouvez actuellement.
        - **Par exemple** : si vous êtes actuellement à la version 2017.1, vous devez utiliser cette URL à la place: http://system.netsuite.com/download/ide/update_17_1
    4. Sélectionnez le site "SuiteCloud IDE" dans la liste déroulante *Work With*
    5. Passez à travers l'assistant d'installation
    6. Redémarrez Eclipse lorsque vous y êtes invité
3. Configurez le plug-in IDE SuiteCloud
    1. Lorsque Eclipse redémarre, vous serez invité à configurer le plug-in SuiteCloud avec un mot de passe principal et un compte NetSuite par défaut
    2. Une fois cet assistant configuré, accédez à *Préférences* > *NetSuite*
        - Vous trouverez ici toutes les préférences d'IDE SuiteCloud
    3. [Facultatif] Si votre utilisation principale pour Eclipse est le développement de NetSuite, accédez à *Préférences* > *Général* > *Perspectives* et définissez la perspective "NetSuite" par défaut.
4. Créer un nouveau projet NetSuite
    1. Cliquez avec le bouton droit de la souris dans la fenêtre de l' *explorateur NS* et sélectionnez *Nouveau* > *Projet NetSuite.*
    2. Suivez l'assistant pour la configuration du projet de votre choix. Les types de projet sont les suivants:
        1. *Personnalisation du compte* : projet qui *utilise* le *framework de développement SuiteCloud* pour créer des objets, des enregistrements et des scripts personnalisés pour personnaliser un compte NetSuite.
        2. *SuiteScript* : Un projet utilisé exclusivement pour écrire des scripts.
        3. *Application SSP* : application Pages SuiteScript Server, généralement utilisée avec SiteBuilder ou SuiteCommerce pour les applications de commerce électronique basées sur NetSuite.

Bonjour, World 1.0 Script client

1. Créez le fichier source pour votre nouveau script client

    1. Créer un nouveau fichier JavaScript en utilisant votre éditeur ou IDE préféré

    2. Ajoutez le code source suivant à votre fichier (source originale ici)

```
/**
 * A simple "Hello, World!" example of a Client Script. Uses the `pageInit`
 * event to write a message to the console log.
 */

function pageInit(type) {
    console.log("Hello, World from a 1.0 Client Script!");
}
```

    3. Enregistrez le fichier sous le nom `hello-world.js` où vous voulez

2. Utilisez le fichier source que nous venons de créer pour créer un nouvel enregistrement de *script* dans NetSuite

1. Dans votre compte NetSuite, accédez à *Personnalisation* > *Script* > *Scripts* > *Nouveau*
2. Lorsque vous y `hello-world.js` invité, sélectionnez `hello-world.js` comme *fichier de script.*
3. Cliquez sur *Créer un enregistrement de script*
4. Lorsque vous y êtes invité, sélectionnez *Script client en* tant que type de script
5. Nommez votre enregistrement de script *Hello World*
6. `pageInit` la fonction nommée `pageInit` dans notre fichier source à l'événement de script *Page Init* en entrant `pageInit` dans le champ *Fonction de* `pageInit` de *page*
7. Enregistrez votre nouvel enregistrement de script

3. Déployer votre nouveau script dans l'enregistrement de l'employé
    1. Sur votre enregistrement de script nouvellement créé, cliquez sur *Déployer le script*
    2. Dans le champ *S'applique à* , sélectionnez *Employé*
    3. Assurez-vous que le champ *État* est défini sur *Tests*
    4. Cliquez sur *Enregistrer*
4. Voyez votre script en action!
    1. Ouvrez la console développeur / JavaScript de votre navigateur (généralement F12 sur la plupart des navigateurs)
    2. Créez un nouvel employé en accédant à *Listes* > *Employés* > *Employés* > *Nouveau*
    3. Observez votre message "Hello, World" dans la console du navigateur.

Bonjour, script client World 2.0

1. Créez le fichier source pour votre nouveau script client

    1. Créer un nouveau fichier JavaScript en utilisant votre éditeur ou IDE préféré

    2. Ajoutez le code source suivant à votre fichier (source originale ici)

```
define([], function () {
    /**
     * A simple "Hello, World!" example of a Client Script. Uses the `pageInit`
     * event to write a message to the console log.
     *
     * @NApiVersion 2.x
     * @NModuleScope Public
     * @NScriptType ClientScript
     */
    var exports = {};
    function pageInit(context) {
        console.log("Hello, World from a 2.0 Client Script!");
    }
    exports.pageInit = pageInit;
    return exports;
});
```

    3. Enregistrez le fichier sous le nom `hello-world2.js` où vous voulez

2. Utilisez le fichier source que nous venons de créer pour créer un nouvel enregistrement de *script* dans NetSuite
    1. Dans votre compte NetSuite, accédez à *Personnalisation* > *Script* > *Scripts* > *Nouveau*
    2. Lorsque vous y `hello-world2.js` invité, sélectionnez `hello-world2.js` comme *fichier de*

script.
    3. Cliquez sur *Créer un enregistrement de script*
    4. Nommez votre enregistrement de script *Hello World*
    5. Enregistrez votre nouvel enregistrement de script
3. Déployer votre nouveau script dans l'enregistrement de l'employé
    1. Sur votre enregistrement de script nouvellement créé, cliquez sur *Déployer le script*
    2. Dans le champ *S'applique à* , sélectionnez *Employé*
    3. Assurez-vous que le champ *État* est défini sur *Tests*
    4. Cliquez sur *Enregistrer*
4. Voyez votre script en action!
    1. Ouvrez la console développeur / JavaScript de votre navigateur (généralement F12 sur la plupart des navigateurs)
    2. Créez un nouvel employé en accédant à *Listes* > *Employés* > *Employés* > *Nouveau*
    3. Observez votre message "Hello, World" dans la console du navigateur.

Lire Démarrer avec netsuite en ligne: https://riptutorial.com/fr/netsuite/topic/3828/demarrer-avec-netsuite

Chapitre 2: Chargement d'un enregistrement

Examples

SS 1.0

```
var recordType = 'customer'; // The type of record to load. The string internal id.
var recordID = 100; // The specific record instances numeric internal id.
var initializeValues = null;
/* The first two parameters are required but the third --
*  in this case the variable initializeValues -- is optional. */
var loadedRecord = nlapiLoadRecord(recordType, recordID, initializeValues);
```

SS 2.0

Cet exemple suppose que le module d'enregistrement est défini sur la variable RECORDMODULE, comme indiqué ci-dessous.

```
require(['N/record'], function(RECORDMODULE){

    var recordType = RECORDMODULE.Type.SALES_ORDER; //The type of record to load.
    var recordID = 100; //The internal ID of the existing record instance in NetSuite.
    var isDynamic = true; //Determines whether to load the record in dynamic mode.

    var loadedRecord = RECORDMODULE.load({
        type: recordType,
        id: recordID,
        isDynamic: isDynamic,
    });
});
```

Lire Chargement d'un enregistrement en ligne:
https://riptutorial.com/fr/netsuite/topic/4685/chargement-d-un-enregistrement

Chapitre 3: Comprendre les recherches de transactions

Introduction

Une compréhension approfondie de la façon dont la fonction de recherche de transactions est une connaissance cruciale pour chaque développeur NetSuite, mais le comportement par défaut de ces recherches et le contrôle de ce comportement peuvent être assez déroutants au départ.

Remarques

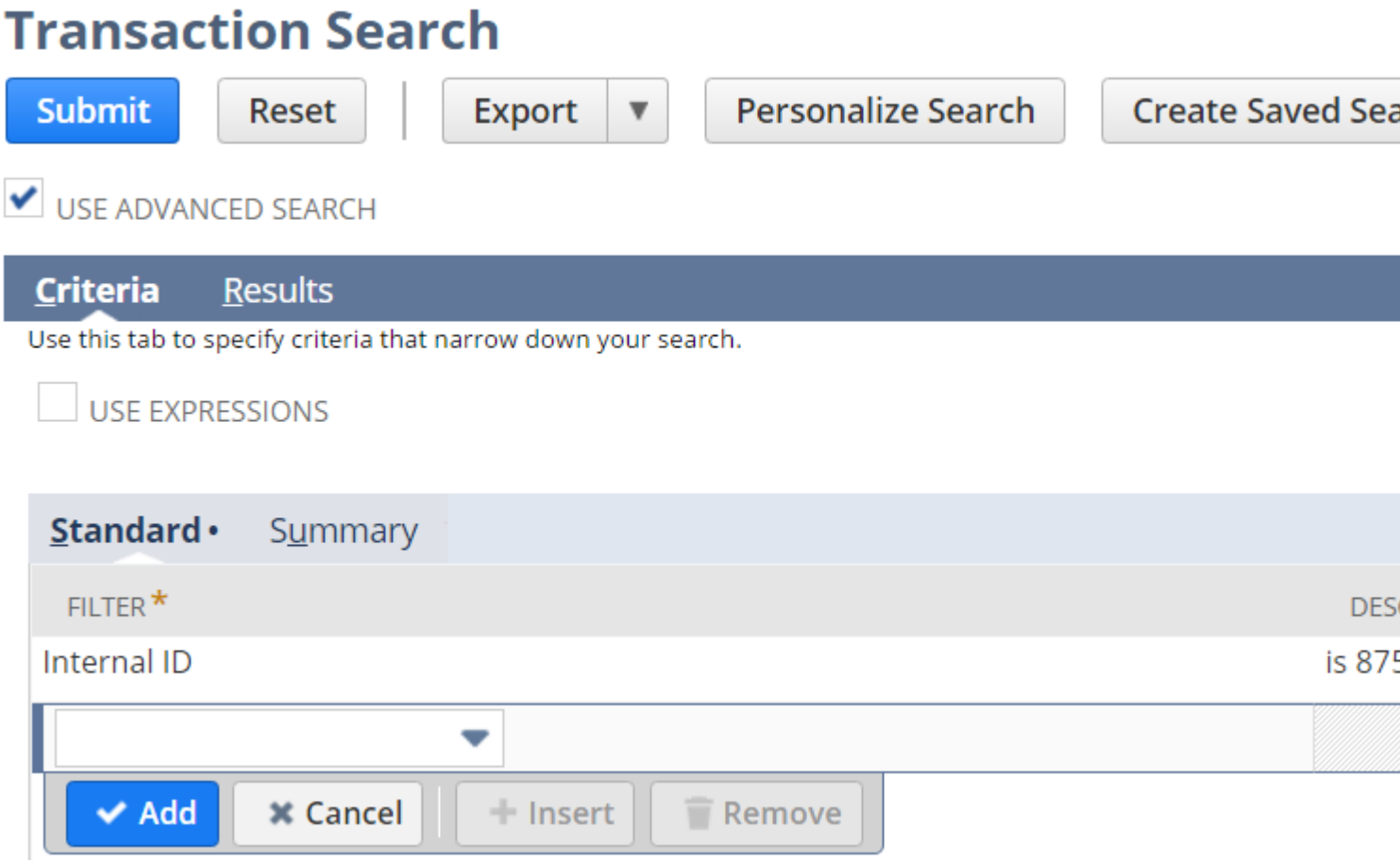
Les références:

- Page d'aide NetSuite: "Utilisation de la ligne principale dans les critères de recherche de transaction"

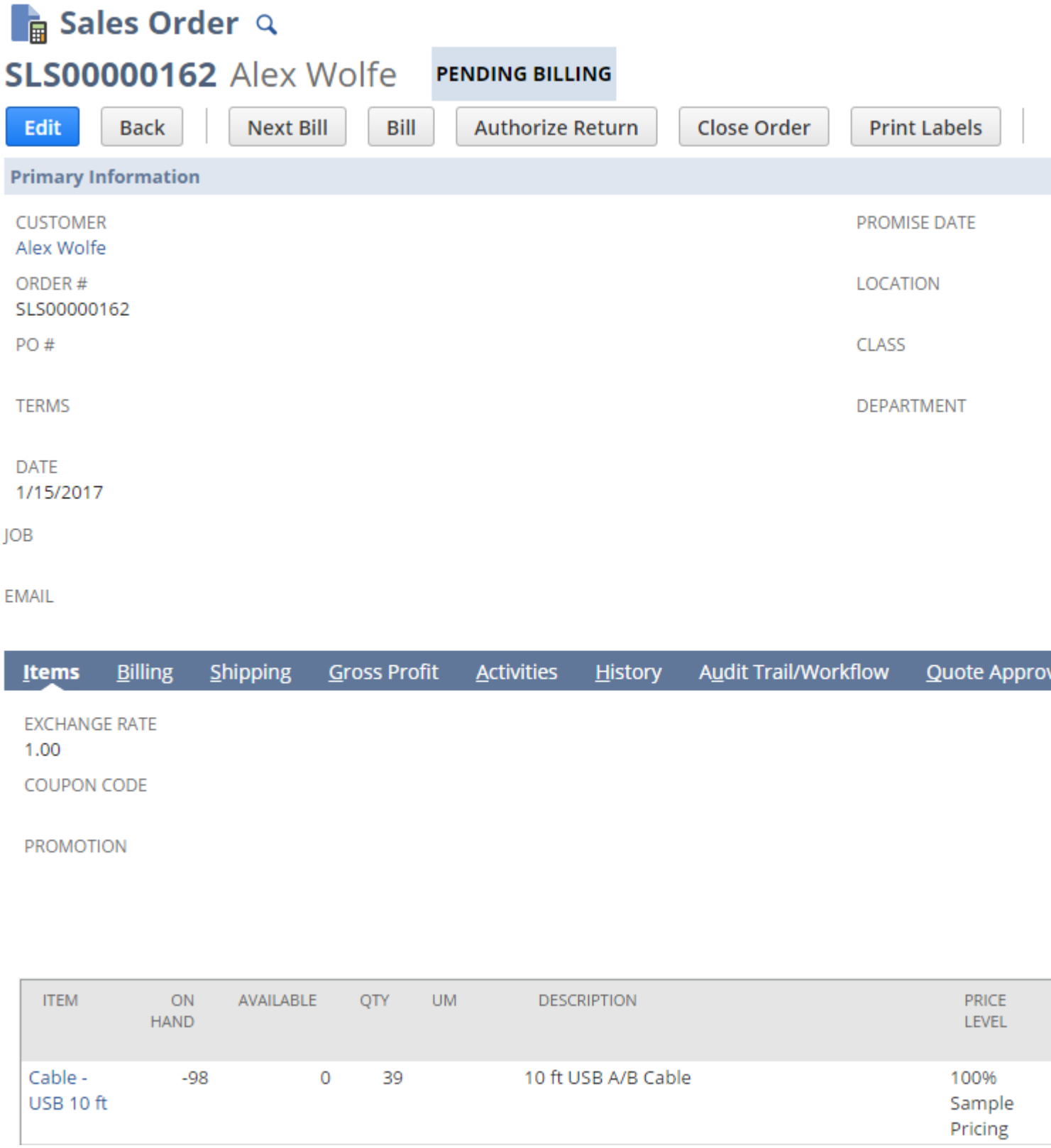
Examples

Filtrage uniquement sur l'ID interne

Explorons un exemple de recherche de transaction où nous définissons un filtre pour l'ID interne d'une transaction unique:

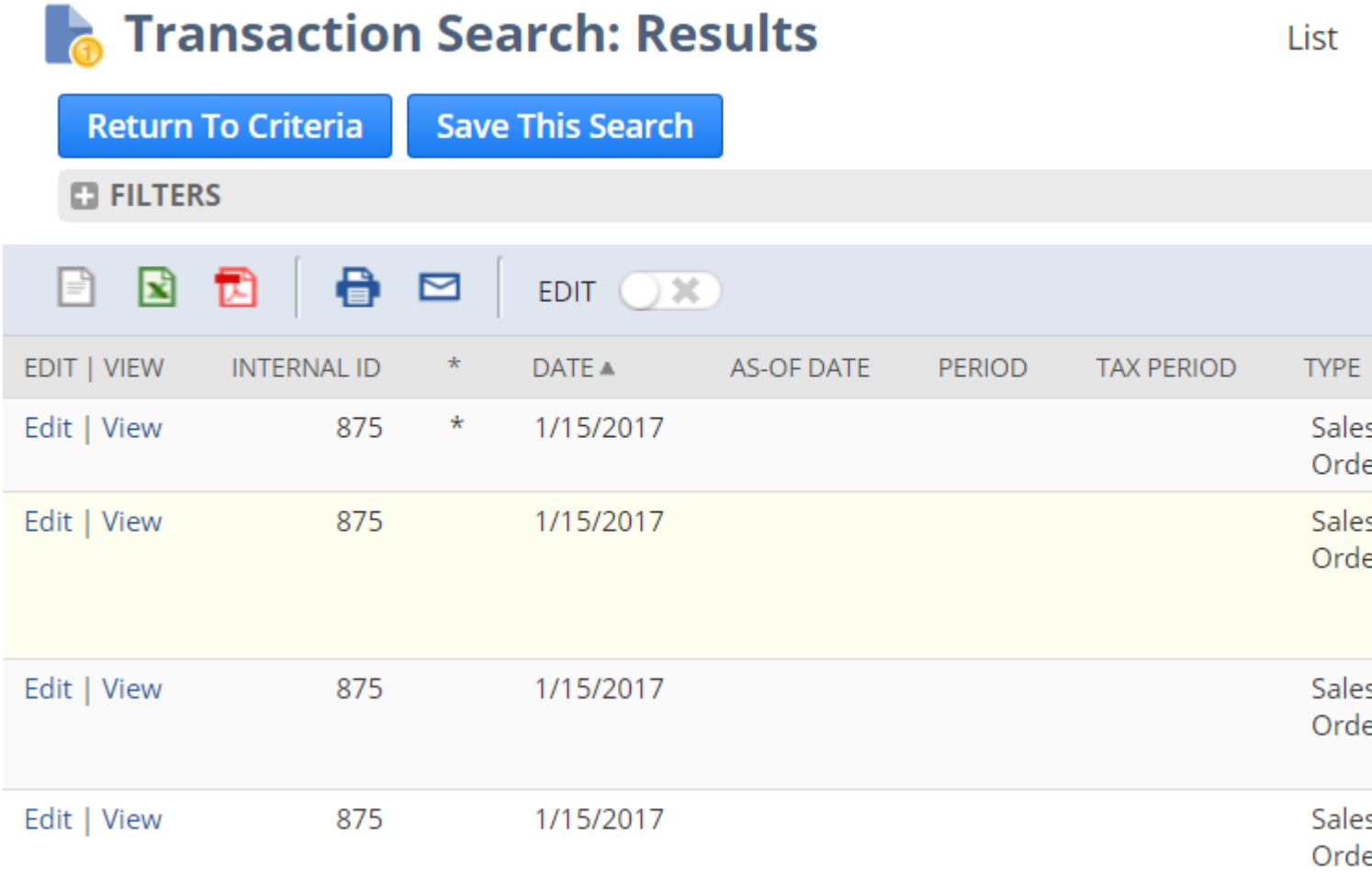


Nous avons spécifié un filtre pour afficher uniquement les résultats de la transaction avec l'ID interne 875; voici cette transaction:



Nous pouvons voir qu'il s'agit d'une commande client avec un seul élément de ligne.

Étant donné que les ID internes sont uniques pour toutes les transactions, nous ne pouvons attendre qu'un seul résultat de recherche pour cette recherche. Voici le résultat de la recherche:



Au lieu du seul résultat attendu, nous obtenons *quatre* résultats. De plus, chaque résultat a exactement le même identifiant interne. Comment est-ce possible?

Pour comprendre ce qui se passe ici, nous devons rappeler que les données stockées dans les enregistrements NetSuite sont divisées en deux catégories:

1. Données de corps: données stockées dans des champs autonomes de l'enregistrement (par exemple, date, représentant commercial, numéro de document, code de coupon)
2. Données de sous-liste: données stockées dans des listes dans chaque enregistrement, généralement affichées sur des sous-onglets de l'interface utilisateur (par exemple, éléments d'une commande client)

Les transactions contiennent plusieurs sous-listes de données, y compris ses:

- éléments de ligne
- Informations sur la livraison
- informations fiscales
- COGS (coût des marchandises vendues) détails

Dans ces résultats de recherche, NetSuite nous montre en fait un résultat pour le corps de la transaction, puis d'autres résultats pour les données des différentes sous-listes de cette même transaction.

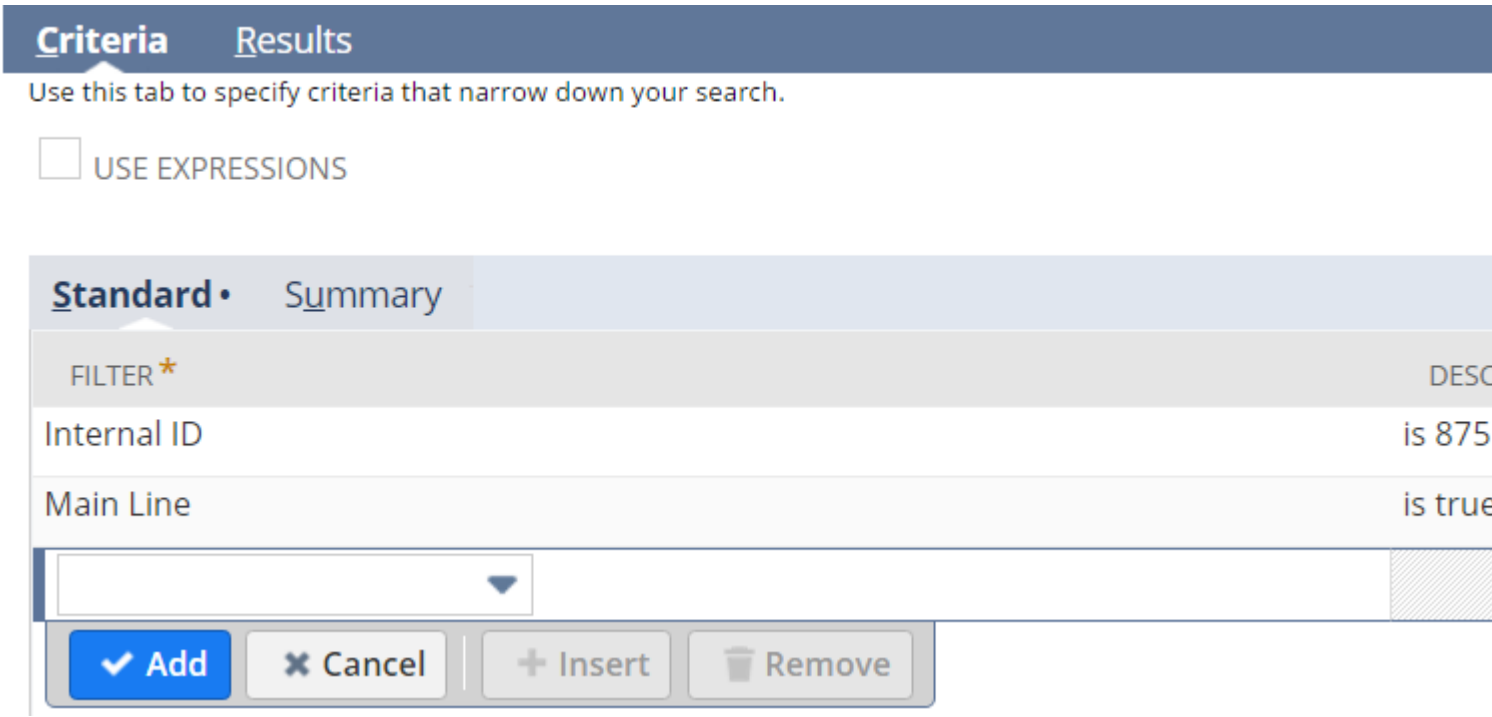
Notez la colonne dans nos résultats de recherche simplement nommée avec un astérisque (*). Notez également que l'un des résultats a un astérisque rempli dans cette colonne alors que le reste est vide. Cette colonne indique quel résultat de recherche représente le corps de la transaction, également appelée ligne principale de la transaction.

Il y a des moments où vous souhaitez que les recherches de transactions affichent uniquement les données de la ligne principale et les heures où vous ne souhaitez que les détails au niveau de la ligne. Les autres exemples montrent comment contrôler ce qui apparaît dans nos résultats.

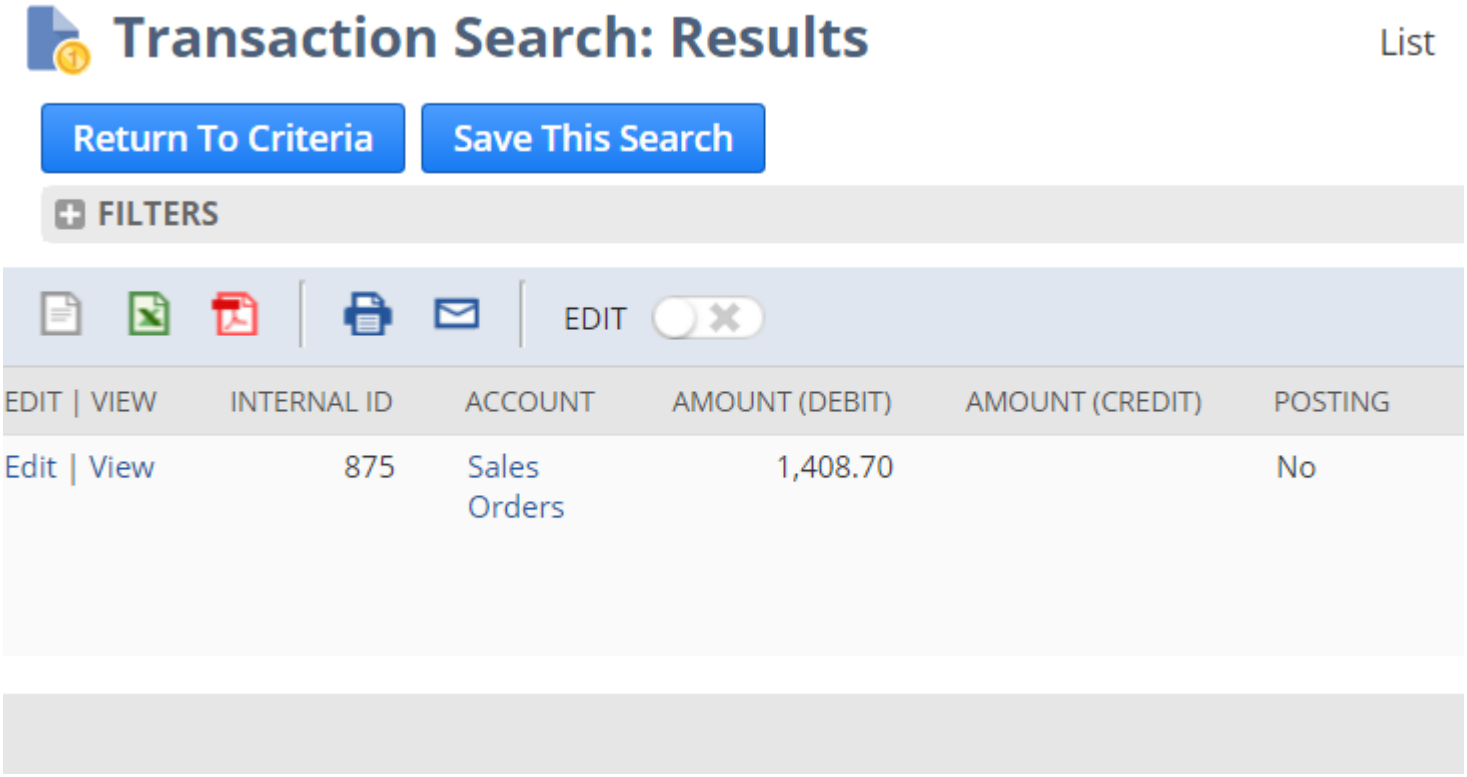
Filtrage avec la ligne principale

Lorsque nous voulons un seul résultat par transaction, cela signifie que nous ne voulons que le corps, ou la ligne principale, de chaque transaction. Pour ce faire, il existe un filtre nommé "Ligne principale".

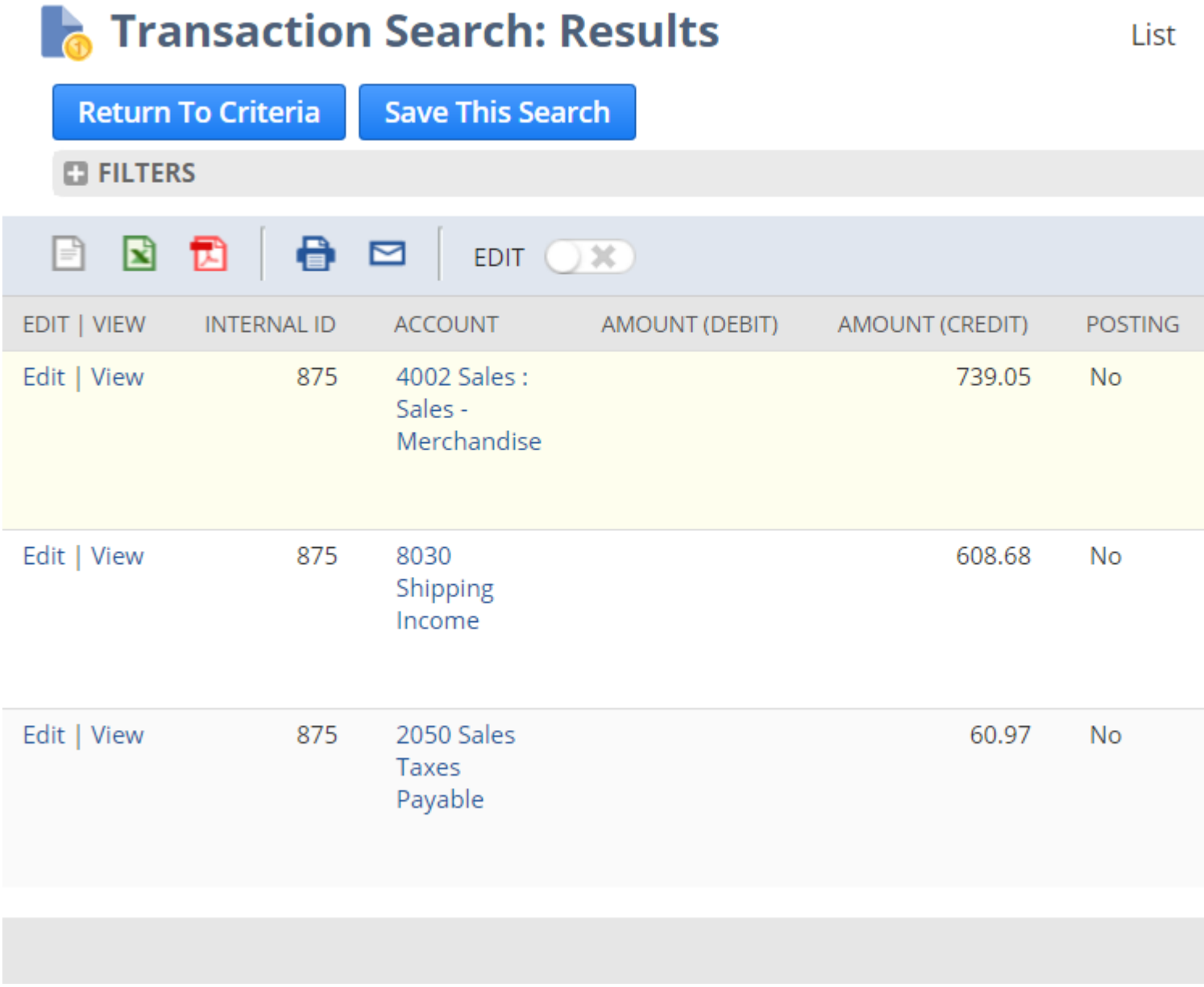
En définissant le filtre *Ligne principale* sur *Oui* dans nos critères de recherche, nous disons essentiellement "Afficher uniquement les données au niveau du corps pour les transactions dans mes résultats":



La modification de nos critères de recherche précédents nous donne désormais le seul résultat attendu:



Si nous inversons notre filtre *Main Line* sur *No* , nous disons "Afficher uniquement les données des sous-listes dans mes résultats":



Pour récapituler *le* comportement de *Main Line* :

- Avec *la ligne principale* définie sur *Oui* , nous avons reçu *un* résultat *uniquement* pour *le corps* de la transaction.
- Avec *la ligne principale* définie sur *Non* , nous avons reçu *trois* résultats *uniquement* pour *les données* de *sous-liste* de la transaction.
- En l'absence de filtre *Main Line* , nous avons reçu *quatre* résultats, essentiellement la combinaison de toutes les données de corps et de sous-liste pour la transaction.

Notez que le filtre Ligne principale n'est pas pris en charge pour les recherches d'entrées de journal.

Filtrage de sous-listes spécifiques

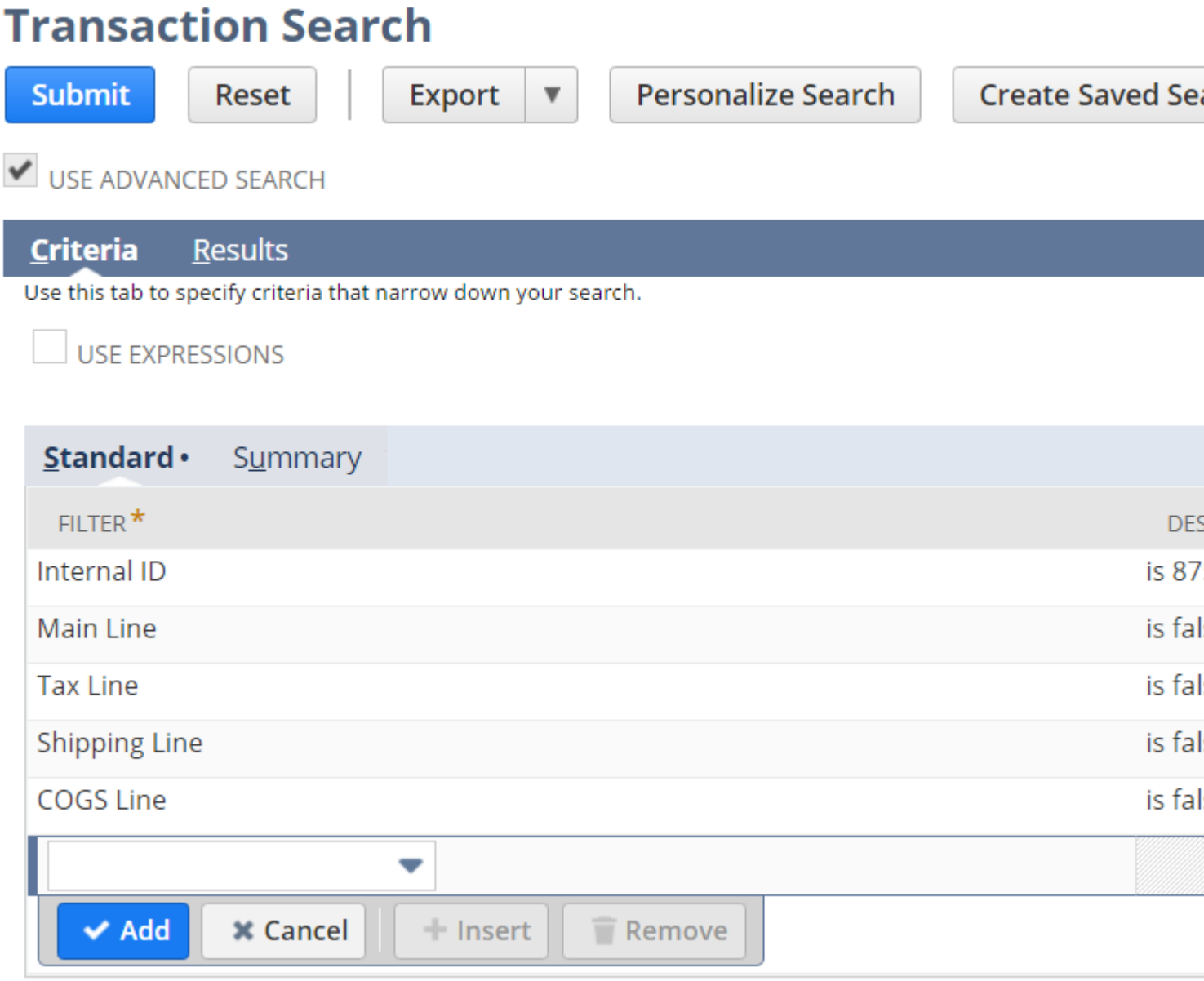
Rappelons que chaque transaction contient plusieurs sous-listes de données. Maintenant que nous ne pouvons afficher que les données de sous-liste à l'aide de *Main Line* , nous pouvons affiner davantage nos résultats de recherche pour obtenir des données de sous-liste spécifiques.

La plupart des sous-listes incluses dans les résultats de la transaction ont un filtre de recherche correspondant pour indiquer si elles sont incluses dans vos résultats:

- Utilisez le filtre *Ligne d'expédition* pour contrôler les données de la sous-liste Expédition
- Utilisez le filtre *Ligne de taxe* pour contrôler les données de la sous-liste de taxe
- Utilisez le filtre *Ligne COGS* pour contrôler les données de la sous-liste COGS

Chacun de ces filtres se comporte comme *Main Line* ou tout autre filtre de case à cocher: *Oui* pour inclure ces données, *Non* pour les exclure de vos résultats.

Notez qu'il n'y a pas de filtre pour *Item Line* pour contrôler les données de la sous-liste Item. Essentiellement, pour pouvoir dire "Afficher uniquement les données de la sous-liste des éléments", nous devons spécifier tous ces filtres comme *non* dans nos critères:



Avec ces critères, votre recherche renverra un résultat par ligne d'article sur chaque transaction correspondante.

À mon avis, ce filtre manquant est une lacune majeure dans la fonctionnalité de recherche qui

devrait être corrigée; il serait beaucoup plus facile et plus cohérent de simplement filtrer une *ligne de poste* . D'ici là, vous devez spécifier que vous souhaitez uniquement les données d'article dans vos résultats de transaction.

Lire Comprendre les recherches de transactions en ligne:
https://riptutorial.com/fr/netsuite/topic/9012/comprendre-les-recherches-de-transactions

Chapitre 4: Créer un enregistrement

Examples

Créer une nouvelle tâche

```
var record = nlapiCreateRecord('task');
record.setFieldValue('title', taskTitle);
var id = nlapiSubmitRecord(record, true);
```

Créer un enregistrement en mode dynamique

var record = nlapiCreateRecord ('customrecord_ennveeitissuetracker', {recordmode: 'dynamic'}); nlapiLogExecution ('DEBUG', 'record', record); record.setFieldValue ('custrecord_name1', name); record.setFieldValue ('custrecord_empid', id); record.setFieldValue ('custrecord_contactno', contactno); record.setFieldValue ('custrecord_email', email); record.setFieldValue ('custrecord_location', loc); record.setFieldValue ('custrecord_incidentdate', date d'incident); record.setFieldValue ('custrecord_issuedescription', desc); // record.setFieldValue ('custrecord_reportedby', rapport); record.setFieldValue ('custrecord_issuetype', issuetype); record.setFieldValue ('custrecord_priority', priority); // record.setFieldValue ('custrecord_replacement fourniture', repl); record.setFieldValue ('custrecord_issuestatus', issuestatus); // record.setFieldValue ('custrecord_resolvedby', résolu par); record.setFieldValue ('custrecord_remarks', remarques); record.setFieldValue ('custrecord_resolvedby', résolu par); record.setFieldValue ('custrecord_updatedstatus', updatedstatus); var id = nlapiSubmitRecord (enregistrement, true); var recordId = nlapiGetRecordId (); record = nlapiLoadRecord ('customrecord_ennveeitissuetracker', id);

Lire Créer un enregistrement en ligne: https://riptutorial.com/fr/netsuite/topic/5127/creer-un-enregistrement

Chapitre 5: Demande de customField, customFieldList & customSearchJoin avec PHP API Advanced Search

Introduction

Celles-ci où certaines des choses les plus difficiles à faire (et dont on parle le moins) avec la recherche avancée de l'API PHP (où vous spécifiez quels champs).

Je suis en train de migrer vers la bibliothèque rest_suite github qui utilise RESTLET, et de contourner la limite de concurrence de 1 utilisateur de l'API PHP.

Mais avant que je supprime mon ancien code im le poster ici. Des exemples de spécifications pour ces champs peuvent être trouvés ici:
http://www.netsuite.com/help/helpcenter/en_US/srbrowser/Browser2016_1/schema/search/transactionsea

Examples

Utilisation de customField et customFieldList

```
$service = new NetSuiteService();
$search = new TransactionSearchAdvanced();
$internalId = '123';//transaction internalId

$search->criteria->basic->internalIdNumber->searchValue = $internalId;
$search->criteria->basic->internalIdNumber->operator = "equalTo";

$field = new SearchColumnSelectCustomField();
$field->scriptId = 'custbody_os_freight_company';//this is specific to you & found in netsuite
$search->columns->basic->customFieldList->customField[] = $field;

$field = new SearchColumnStringCustomField();
$field->scriptId = 'custbody_os_warehouse_instructions';//this is specific to you & found in
netsuite
$search->columns->basic->customFieldList->customField[] = $field;

//and so on, you can keep adding to the customField array the custom fields you want

$request = new SearchRequest();

$request->searchRecord = $search;

$searchResponse = $service->search($request);
```

CustomSearchJoin Utilisation

```
$service = new NetSuiteService();
$search = new TransactionSearchAdvanced();
```

```
$internalId = '123';//transaction internalId

$search->criteria->basic->internalIdNumber->searchValue = $internalId;
$search->criteria->basic->internalIdNumber->operator = "equalTo";

$CustomSearchRowBasic = new CustomSearchRowBasic();
$CustomSearchRowBasic->customizationRef->scriptId = 'custbody_os_entered_by';//this is
specific to you & found in netsuite
$CustomSearchRowBasic->searchRowBasic = new EmployeeSearchRowBasic();
$CustomSearchRowBasic->searchRowBasic->entityId = new SearchColumnStringField();

$search->columns->customSearchJoin[] = $CustomSearchRowBasic;
//and so on, you can keep adding to the customSearchJoin array the custom fields you want


$request = new SearchRequest();

$request->searchRecord = $search;

$searchResponse = $service->search($request);
```

Lire Demande de customField, customFieldList & customSearchJoin avec PHP API Advanced Search en ligne: https://riptutorial.com/fr/netsuite/topic/9799/demande-de-customfield--customfieldlist--amp--customsearchjoin-avec-php-api-advanced-search

Chapitre 6: Edition en ligne avec SuiteScript

Introduction

La modification en ligne permet aux utilisateurs de modifier et de mettre à jour très rapidement les données d'un enregistrement particulier sans avoir à charger l'intégralité de l'enregistrement sur une page, à modifier le formulaire, puis à enregistrer l'enregistrement.

Les développeurs NetSuite ont une fonctionnalité correspondante appelée `submitFields` . La fonctionnalité `submitFields` est fournie par la fonction globale `nlapiSubmitField` dans SuiteScript 1.0 et la méthode `N/record#submitFields` dans SuiteScript 2.0.

Syntaxe

- nlapiSubmitField (recordType, recordId, fieldId, fieldValue);
- nlapiSubmitField (recordType, recordId, fieldIds, fieldValues);
- nlapiSubmitField (recordType, recordId, fieldId, fieldValue, doSourcing);

Paramètres

Paramètre	Détails
recordType	`String` - L'ID interne du type d'enregistrement mis à jour
recordId	`String` ou `Number` - L'ID interne de l'enregistrement en cours de mise à jour
fieldIds	`String` ou `String[]` - ID interne du ou des champs mis à jour
fieldValues	`any` ou `any[]` - Les valeurs correspondantes à définir dans les champs donnés
faireSourcing	`Boolean` - `Boolean` si les valeurs dépendantes doivent être entrées lors de la soumission de l'enregistrement. Le défaut est `false`

Remarques

La fonctionnalité `submitFields` est une fonctionnalité complémentaire de la fonctionnalité `lookupFields` .

Performance et limites

`submitFields` fonctionne beaucoup plus rapidement et utilise moins de gouvernance que les mêmes modifications en chargeant et en soumettant l'enregistrement complet.

Plusieurs champs peuvent être mis à jour à la fois pour le même coût que la mise à jour d'un seul champ. La mise à jour de davantage de champs avec `submitFields` *n'entraîne pas de* coûts de gouvernance plus élevés.

Toutefois, vous devez savoir que seuls certains champs de chaque type d'enregistrement sont modifiables en ligne et que les économies de performances *ne* s'appliquent qu'à ces champs modifiables en ligne. Si vous utilisez la fonction `submitFields` sur un champ non éditable en ligne, le champ *sera* mis à jour correctement, mais en coulisse, NetSuite chargera et soumettra l'enregistrement, ce qui prendra plus de temps et utilisera plus de gouvernance. Vous pouvez déterminer si un champ est modifiable en ligne en vous référant à la colonne "nlapiSubmitField" du navigateur d'enregistrements .

`submitFields` fonctionnalité `submitFields` est également limitée aux champs de *corps* d'un enregistrement. Si vous avez besoin de modifier les données de sous-liste, vous devrez charger l'enregistrement pour effectuer vos modifications, puis soumettre le document.

Les références:

- NetSuite Help: "Édition en ligne et présentation de SuiteScript"
- NetSuite Help: "Edition en ligne avec nlapiSubmitField"
- NetSuite Help: "Conséquences de l'utilisation de nlapiSubmitField sur des champs modifiables non intégrés"
- NetSuite Help: "Champ API"
- NetSuite Help: "record.submitFields (options)"

Examples

[1.0] Soumettre un champ unique

```
/**
 * A SuiteScript 1.0 example of using nlapiSubmitField to update a single field on a related
record
 */

// From a Sales Order, get the Customer ID
var customerId = nlapiGetFieldValue("entity");

// Set a comment on the Customer record
nlapiSubmitField("customer", customerId, "comments", "This is a comment added by inline
editing with SuiteScript.");
```

[1.0] Soumettre plusieurs champs

```
/**
 * A SuiteScript 1.0 example of using nlapiSubmitField to update multiple fields on a related
record
 */
```

```
// From a Sales Order, get the Customer ID
var customerId = nlapiGetFieldValue("entity");

// Set a Comment and update the Budget Approved field on the Customer record
nlapiSubmitField("customer", customerId,
    ["comments", "isbudgetapproved"],
    ["The budget has been approved.", "T"]);
```

[2.0] Soumettre un champ unique

```
/**
 * A SuiteScript 2.0 example of using N/record#submitFields to update a single field on a
related record
 */

require(["N/record", "N/currentRecord"], function (r, cr) {

    // From a Sales Order, get the Customer ID
    var customerId = cr.get().getValue({"fieldId": "entity"});

    // Set a Comment on the Customer record
    r.submitFields({
        "type": r.Type.CUSTOMER,
        "id": customerId,
        "values": {
            "comments": "This is a comment added by inline editing with SuiteScript."
        }
    });
});
```

[2.0] Soumettre plusieurs champs

```
/**
 * A SuiteScript 2.0 example of using N/record#submitFields to update multiple fields on a
related record
 */

require(["N/record", "N/currentRecord"], function (r, cr) {

    // From a Sales Order, get the Customer ID
    var customerId = cr.get().getValue({"fieldId": "entity"});

    // Set a Comment and check the Budget Approved box on the Customer record
    r.submitFields({
        "type": r.Type.CUSTOMER,
        "id": customerId,
        "values": {
            "comments": "The budget has been approved.",
            "isbudgetapproved": true
        }
    });
});
```

Lire Edition en ligne avec SuiteScript en ligne: https://riptutorial.com/fr/netsuite/topic/9082/edition-en-ligne-avec-suitescript

Chapitre 7: Enregistrements de déploiement de script et de script

Introduction

Pour que NetSuite sache utiliser notre code source, nous devons pouvoir lui indiquer quelles fonctions appeler, quand les appeler et à qui les appeler. Nous accomplissons tout cela avec les enregistrements de *déploiement de script* et de *script* .

Examples

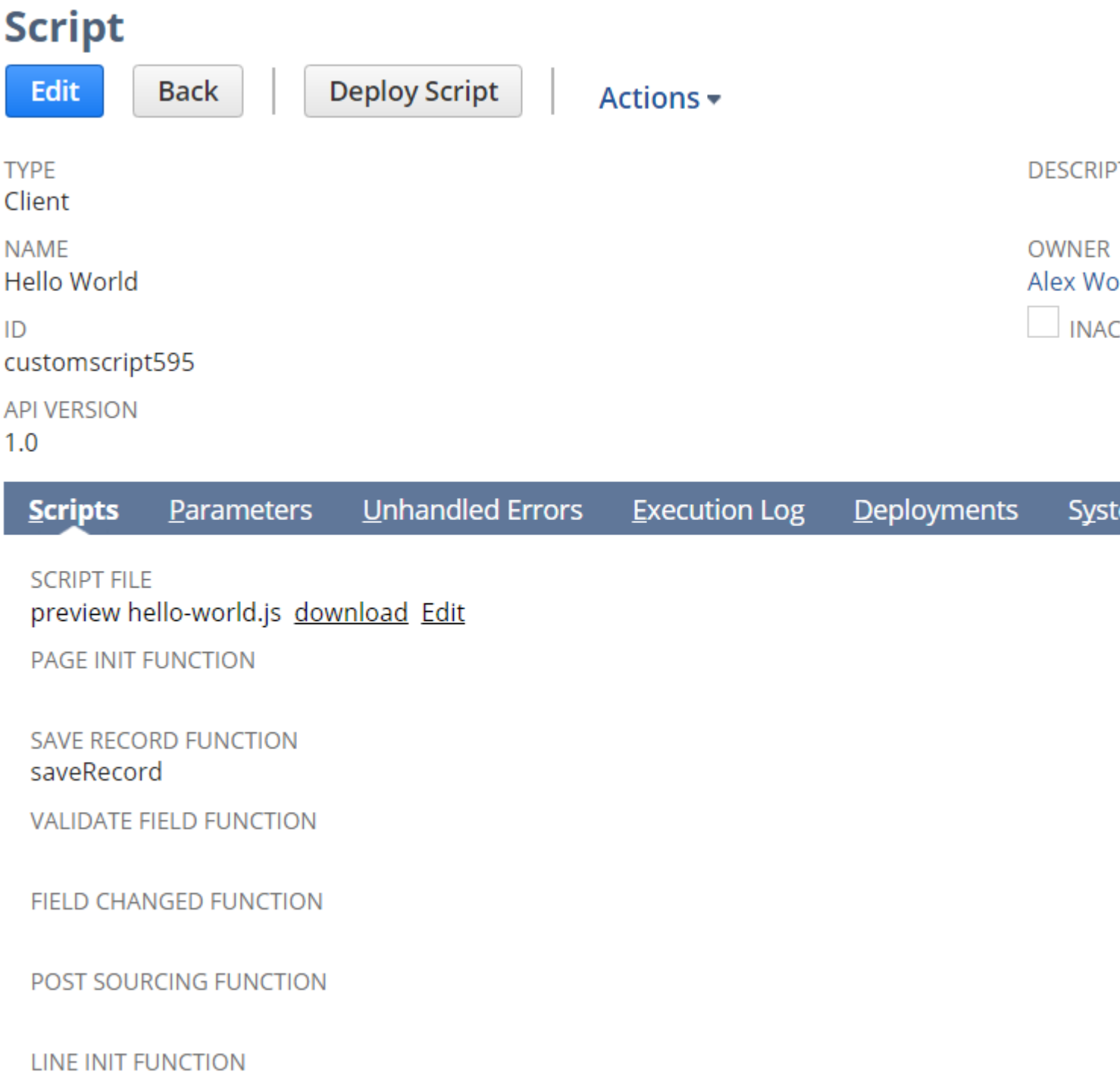
Enregistrements de script

NetSuite utilise l'enregistrement de *script* pour mapper les fonctions de votre fichier source à des événements spécifiques du système. Par exemple, si une logique métier doit être exécutée lorsqu'un formulaire est enregistré dans l'interface utilisateur, l'enregistrement de script indique à NetSuite quelle fonction appeler lorsque l'événement `Save Record` se produit.

Vous pouvez considérer l'enregistrement de *script* comme définissant le *moment où* notre code source doit être exécuté. il définit essentiellement quelque chose qui s'apparente à:

> "Lorsqu'un enregistrement est enregistré, appelez la fonction saveRecord dans hello-world.js."

Voici un exemple de ce à quoi ressemblerait l'enregistrement de script:



Enregistrements de déploiement de script

Une fois que nous avons créé un enregistrement de *script* , nous devons déployer ce script dans le système. Bien que l'enregistrement de *script* indique à NetSuite les fonctions à appeler depuis notre fichier source, l'enregistrement de *déploiement de script* permet à NetSuite de savoir quels enregistrements et quels utilisateurs notre script doit exécuter.

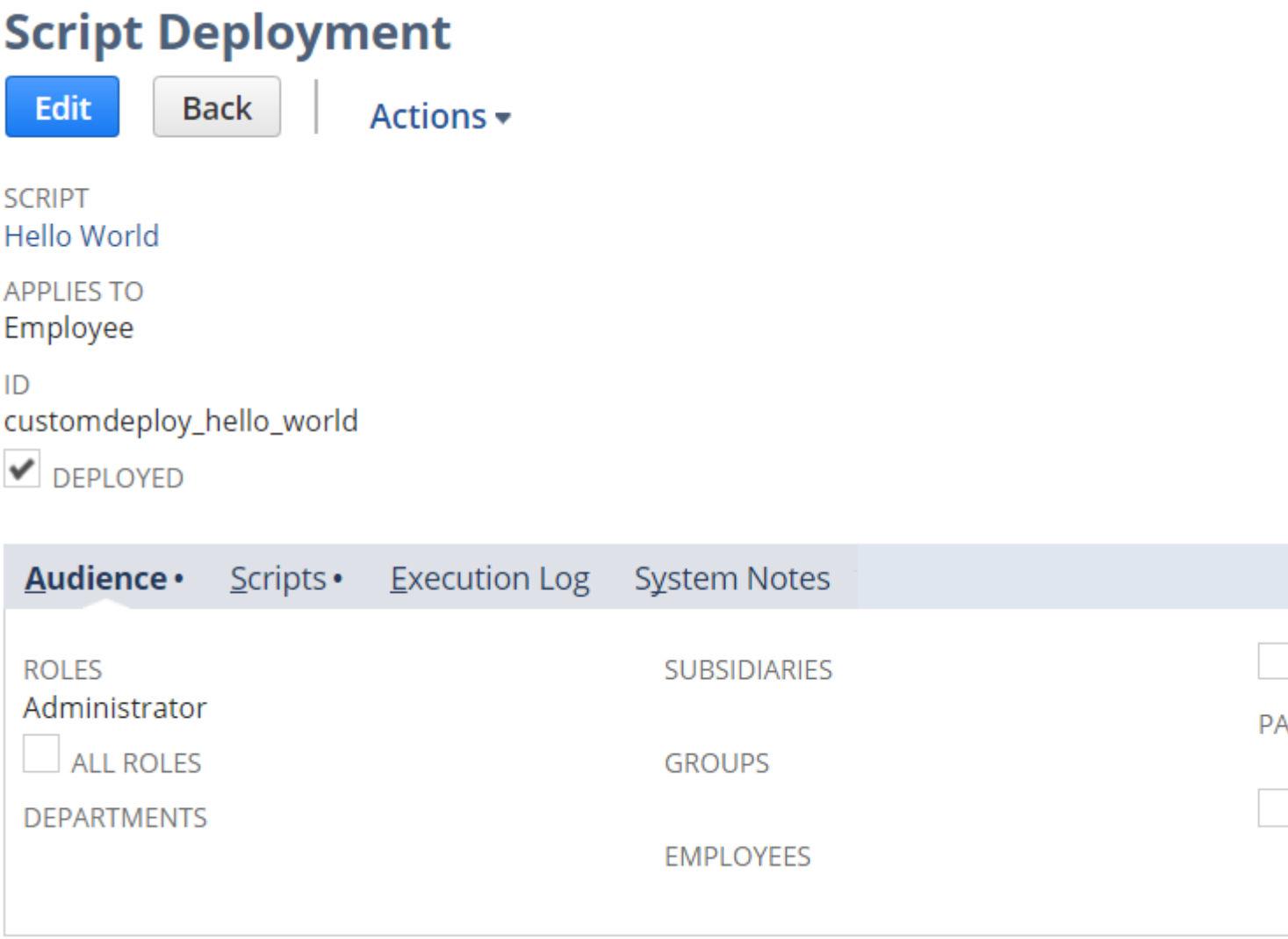
Bien que l'enregistrement de *script* définisse *quand* notre code source doit être exécuté, le *script de déploiement* définit *où* et *qui* peut exécuter notre script. Si nous avons un enregistrement de *script* qui dit:

> "Lorsqu'un enregistrement est enregistré, appelez la fonction saveRecord dans hello-world.js."

alors notre *déploiement de script* pour cet enregistrement peut modifier légèrement cela pour:

> "Lorsqu'un enregistrement Employé est enregistré, appelez la fonction saveRecord dans hello-world.js, mais uniquement pour les utilisateurs du groupe Administrateurs."

Encore une fois, voici un exemple de ce à quoi ressemblerait le *déploiement de scripts* :



Un *script* peut être associé à plusieurs *déploiements de script* . Cela nous permet de déployer la même logique métier sur plusieurs types d'enregistrements différents avec des audiences différentes.

Lire Enregistrements de déploiement de script et de script en ligne: https://riptutorial.com/fr/netsuite/topic/8835/enregistrements-de-deploiement-de-script-et-de-script

Chapitre 8: Evénement utilisateur: Evénement avant chargement

Paramètres

Paramètre	Détails
SuiteScript 2.0	-
`scriptContext`	{ `Object` }
`scriptContext.newRecord`	{ `N/record.Record` } Une référence à l'enregistrement chargé depuis la base de données
`scriptContext.type`	{ `UserEventType` } Le type d'action qui a déclenché cet événement utilisateur
`scriptContext.form`	{ `N/ui/serverWidget.Form` } Une référence au formulaire d'interface utilisateur qui sera rendu
SuiteScript 1.0	-
`type`	{ `Object` } Le type d'action qui a déclenché cet événement utilisateur
`form`	{ `nlobjForm` } Une référence au formulaire d'interface utilisateur qui sera rendu
`request`	{ `nlobjRequest` } la requête HTTP GET; disponible uniquement lorsqu'il est déclenché par des requêtes du navigateur

Remarques

beforeLoad

L'événement `Before Load` est déclenché par toute opération de lecture sur un enregistrement. Chaque fois qu'un utilisateur, un script, une importation CSV ou une demande de service Web tente de lire un enregistrement de la base de données, l'événement `Before Load` est déclenché.

Enregistrez les actions qui déclenchent un événement `beforeLoad` :

- Créer
- modifier
- Afficher / charger
- Copie
- Impression

- Email
- Aperçu rapide

Cas d'utilisation typiques pour `beforeLoad`

- Modifier le formulaire de l'interface utilisateur avant que l'utilisateur ne le voit
- Définir les valeurs de champs par défaut
- Prétraitement des données

Les événements utilisateur ne sont pas enchaînés

Le code écrit dans les événements utilisateur ne déclenche aucun événement utilisateur sur d' *autres* enregistrements. Par exemple, le chargement de l'enregistrement Client associé à l' `beforeLoad` - `beforeLoad` d'un enregistrement de commande client *ne déclenchera pas* le `beforeLoad` l'enregistrement `beforeLoad` . Même si vous chargez un autre enregistrement Transaction, ses événements utilisateur ne seront pas déclenchés.

NetSuite fait cela pour éviter que des événements utilisateur se déclenchent dans une boucle infinie. Si vous *avez* besoin d' événements utilisateur pour tirer dans une séquence enchaînée, d' autres types de script (par exemple RESTlets, Suitelets, Scripts programmés) devront être injectés entre les événements.

Le gestionnaire d'événements renvoie un `void`

Le type de retour du gestionnaire d'événements `beforeLoad` est `void` . Toute donnée renvoyée par notre gestionnaire d'événement n'a aucun effet sur le système. Nous n'avons pas besoin de renvoyer quelque chose de notre fonction de gestionnaire car nous ne pouvons rien faire avec sa valeur renvoyée.

Examples

Minimal: enregistrer un message sur Before Load

```
// 1.0
function beforeLoad(type, form, request) {
    nlapiLogExecution("DEBUG", "Before Load", "type=" + type);
}

// 2.0
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 * @NModuleScope SameAccount
```

```
 */
define(["N/log"], function (log) {
    function beforeLoad(context) {
        log.debug({
            "title": "Before Load",
            "details": "type=" + context.type
        });
    }

    return {
        "beforeLoad": beforeLoad
    };
});
```

Modifier le formulaire d'interface utilisateur

```
// 1.0
// Revealing Module pattern, structures 1.0 similar to 2.0
var myNamespace = myNamespace || {};
myNamespace.example = (function () {

    /** @appliedtorecord employee */
    var exports = {};

    function beforeLoad(type, form, request) {
        showBonusEligibility(form);
    }

    function showBonusEligibility(form) {
        var field = form.addField("custpage_is_bonus_eligible",
            "checkbox", "Eligible for Bonus?");
        field.setDefaultValue(isEligibleForBonus(nlapiGetNewRecord()) ? "T" : "F");
    }

    function isEligibleForBonus(rec) {
        // Implement actual business rules for bonus eligibility here
        return true;
    }

    exports.beforeLoad = beforeLoad;
    return exports;
})();

// 2.0
/**
 * @appliedtorecord employee
 * @NScriptType UserEventScript
 * @NApiVersion 2.x
 */
define(["N/log", "N/ui/serverWidget"], function (log, ui) {
    var exports = {};

    function beforeLoad(context) {
        showBonusEligibility(context.form);
    }

    function showBonusEligibility(form) {
        var field = form.addField({
            "id": "custpage_is_bonus_eligible",
```

```
            "label": "Eligible for Bonus?",
            "type": ui.FieldType.CHECKBOX
        });
        field.defaultValue = (isEligibleForBonus() ? "T" : "F");
    }

    function isEligibleForBonus(rec) {
        // Implement actual business rules for bonus eligibility here
        return true;
    }

    exports.beforeLoad = beforeLoad;
    return exports;
});
```

Restreindre l'exécution en fonction de l'action qui a déclenché l'événement utilisateur

```
// 1.0
// Utilize the type argument and raw Strings to filter your
// execution by the action
function beforeLoad(type, form, request) {
    // Don't do anything on APPROVE
    // Note that `type` is an Object, so we must use ==, not ===
    if (type == "approve") {
        return;
    }

    // Continue with normal business logic...
}

// 2.0
/**
 * @appliedtorecord employee
 * @NScriptType UserEventScript
 * @NApiVersion 2.x
 */
define([], function () {
    var exports = {};

    // Utilize context.type value and context.UserEventType enumeration
    // to filter your execution by the action
    function beforeLoad(context) {
        // Don't do anything on APPROVE
        if (context.type === context.UserEventType.APPROVE) {
            return;
        }

        // Continue with normal business logic...
    }

    exports.beforeLoad = beforeLoad;
    return exports;
});
```

Restreindre l'exécution en fonction du contexte qui a déclenché l'événement utilisateur

Dans SuiteScript 1.0, nous récupérons le contexte d'exécution actuel à l'aide de `nlapiGetContext().getExecutionContext()` , puis nous comparons le résultat aux chaînes brutes appropriées.

```
// 1.0 in Revealing Module pattern
var myNamespace = myNamespace || {};
myNamespace.example = (function () {
    var exports = {};

    function beforeLoad(type, form, request) {
        showBonusEligibility(form);
    }

    function showBonusEligibility(form) {
        // Doesn't make sense to modify UI form when the request
        // did not come from the UI
        var currentContext = nlapiGetContext().getExecutionContext();
        if (!wasTriggeredFromUi(currentContext)) {
            return;
        }

        // Continue with form modification...
    }

    function wasTriggeredFromUi(context) {
        // Current context must be compared to raw Strings
        return (context === "userinterface");
    }

    function isEligibleForBonus() {
        return true;
    }

    exports.beforeLoad = beforeLoad;
    return exports;
})();
```

Dans SuiteScript 2.0, nous obtenons le contexte d'exécution actuel en important le module `N/runtime` et en inspectant sa propriété `executionContext` . Nous pouvons ensuite comparer sa valeur aux valeurs de l'énumération `runtime.ContextType` plutôt qu'aux chaînes brutes.

```
// 2.0
/**
 * @NScriptType UserEventScript
 * @NApiVersion 2.x
 */
define(["N/ui/serverWidget", "N/runtime"], function (ui, runtime) {
    var exports = {};

    function beforeLoad(scriptContext) {
        showBonusEligibility(scriptContext.form);
    }

    function showBonusEligibility(form) {
        // Doesn't make sense to modify the form if the
        if (!wasTriggeredFromUi(runtime.executionContext)) {
            return;
        }
```

```
        // Continue with form modification...
    }

    function wasTriggeredFromUi(context) {
        // Context can be compared to enumeration from runtime module
        return (context === runtime.ContextType.USER_INTERFACE);
    }

    exports.beforeLoad = beforeLoad;
    return exports;
});
```

Lire Evénement utilisateur: Evénement avant chargement en ligne: https://riptutorial.com/fr/netsuite/topic/7119/evenement-utilisateur--evenement-avant-chargement

Chapitre 9: Evénement utilisateur: événements avant et après la soumission

Syntaxe

- beforeSubmit (type) // Before Submit, 1.0
- beforeSubmit (scriptContext) // Avant la soumission, 2.0
- afterSubmit (type) // Après soumission, 1.0
- afterSubmit (scriptContext) // Après soumission, 2.0

Paramètres

Paramètre	Détails
SuiteScript 2.0	-
`scriptContext`	`{Object}`
`scriptContext.newRecord`	`{N/record.Record}` Une référence à l'enregistrement en cours de lecture dans la base de données. Nous pouvons l'utiliser pour modifier les valeurs de champ sur l'enregistrement
`scriptContext.oldRecord`	`{N/record.Record}` Une référence en lecture seule à l'état précédent de l'enregistrement. Nous pouvons l'utiliser pour comparer aux nouvelles valeurs
`scriptContext.type`	`{UserEventType}` Une énumération du type d'action d'écriture en cours d'exécution
SuiteScript 1.0	-
`type`	`{String}` Le type d'action d'écriture en cours d'exécution

Remarques

`beforeSubmit` et `afterSubmit`

Ces deux événements sont déclenchés par toute opération d'écriture de base de données sur un enregistrement. Chaque fois qu'un utilisateur, un script, une importation CSV ou une demande de service Web tente d'écrire un enregistrement dans la base de données, les événements Submit sont déclenchés.

Enregistrez les actions qui déclenchent les *deux* événements Submit:

- Créer
- modifier
- Effacer
- XEdit (modification en ligne)
- Approuver
- Rejeter
- Annuler
- Pack
- Navire

Enregistrez les actions qui déclenchent avant la `beforeSubmit` uniquement:

- Mark Complete
- Réassigner (cas de support)
- Modifier les prévisions

Enregistrez les actions qui déclenchent `afterSubmit` uniquement:

- Dropship
- Commande spéciale
- Items commandés
- Payer les factures

Cas d'utilisation typiques pour `beforeSubmit`

- Valider l'enregistrement avant qu'il ne soit engagé dans la base de données
- Vérification des autorisations et des restrictions
- Modifications de dernière minute avant la validation de la base de données
- Tirez les mises à jour des systèmes externes

Cas d'utilisation typiques pour `afterSubmit`

- Notification par email des modifications d'enregistrement
- Redirection du navigateur
- Créer / mettre à jour des enregistrements dépendants
- Poussez les changements sur les systèmes externes

Les événements utilisateur ne sont pas enchaînés

Le code écrit dans les événements utilisateur ne déclenche aucun événement utilisateur sur d'*autres* enregistrements. Par exemple, la modification de l'enregistrement client associé à l'`beforeSubmit` d'un enregistrement de commande client *ne déclenchera pas* les événements de

soumission de l'enregistrement client.

NetSuite fait cela pour éviter que des événements utilisateur se déclenchent dans une boucle infinie. Si vous *avez* besoin d' événements utilisateur pour tirer dans une séquence enchaînée, d' autres types de script (par exemple RESTlets, Suitelets, Scripts programmés) devront être injectés entre les événements.

Les gestionnaires d'événements renvoient un `void`

Le type de retour des gestionnaires d'événements Submit est `void` . Toute donnée renvoyée par notre gestionnaire d'événement n'a aucun effet sur le système. Nous n'avons pas besoin de renvoyer quelque chose de notre fonction de gestionnaire car nous ne pouvons rien faire avec sa valeur renvoyée.

!! MISE EN GARDE !!

Soyez très prudent lorsque vous comparez les valeurs entre les anciens et les nouveaux enregistrements. Les champs vides de l' *ancien* enregistrement sont renvoyés comme `null` , tandis que les champs vides du *nouvel* enregistrement sont renvoyés sous forme de chaîne vide. Cela signifie que vous ne pouvez pas simplement comparer l'ancien avec le nouveau ou que vous obtenez des faux positifs. Toute logique que vous écrivez doit gérer le cas où l'un est `null` et l'autre est une chaîne vide appropriée.

Examples

Minimal: enregistrer un message

```
// 1.0, Revealing Module pattern
var myNamespace = myNamespace || {};

myNamespace.example = (function () {

    /**
     * User Event 1.0 example detailing usage of the Submit events
     *
     * @appliedtorecord employee
     */
    var exports = {};

    function beforeSubmit(type) {
        nlapiLogExecution("DEBUG", "Before Submit", "action=" + type);
    }

    function afterSubmit(type) {
        nlapiLogExecution("DEBUG", "After Submit", "action=" + type);
    }

    exports.beforeSubmit = beforeSubmit;
```

```
    exports.afterSubmit = afterSubmit;
    return exports;
})();

// 2.0
define(["N/log"], function (log) {

    /**
     * User Event 2.0 example showing usage of the Submit events
     *
     * @NApiVersion 2.x
     * @NModuleScope SameAccount
     * @NScriptType UserEventScript
     * @appliedtorecord employee
     */
    var exports = {};

    function beforeSubmit(scriptContext) {
        log.debug({
            "title": "Before Submit",
            "details": "action=" + scriptContext.type
        });
    }

    function afterSubmit(scriptContext) {
        log.debug({
            "title": "After Submit",
            "details": "action=" + scriptContext.type
        });
    }

    exports.beforeSubmit = beforeSubmit;
    exports.afterSubmit = afterSubmit;
    return exports;
});
```

Avant de soumettre: Valider l'enregistrement avant qu'il ne soit engagé dans la base de données

Pour cet exemple, nous voulons nous assurer que tous les employés marqués comme *ressource de projet ont* également un *coût de main-d'œuvre* approprié défini.

```
// 1.0, Revealing Module pattern
var myNamespace = myNamespace || {};
myNamespace.example = (function () {

    /**
     * User Event 1.0 example detailing usage of the Submit events
     *
     * @appliedtorecord employee
     */
    var exports = {};

    function beforeSubmit(type) {
        if (!isEmployeeValid(nlapiGetNewRecord())) {
            throw nlapiCreateError("STOIC_ERR_INVALID_DATA", "Employee data is not valid",
true);
        }
```

```
    }

    function isEmployeeValid(employee) {
        return (!isProjectResource(employee) || hasValidLaborCost(employee));
    }

    function isProjectResource(employee) {
        return (employee.getFieldValue("isjobresource") === "T");
    }

    function hasValidLaborCost(employee) {
        var laborCost = parseFloat(employee.getFieldValue("laborcost"));

        return (Boolean(laborCost) && (laborCost > 0));
    }

    exports.beforeSubmit = beforeSubmit;
    return exports;
})();

// 2.0
define(["N/error"], function (err) {

    var exports = {};

    /**
     * User Event 2.0 example detailing usage of the Submit events
     *
     * @NApiVersion 2.x
     * @NModuleScope SameAccount
     * @NScriptType UserEventScript
     * @appliedtorecord employee
     */
    function beforeSubmit(scriptContext) {
        if (!isEmployeeValid(scriptContext)) {
            throw err.create({
                "name": "STOIC_ERR_INVALID_DATA",
                "message": "Employee data is not valid",
                "notifyOff": true
            });
        }
    }

    function isEmployeeValid(scriptContext) {
        return (!isProjectResource(scriptContext.newRecord) ||
hasValidLaborCost(scriptContext.newRecord));
    }

    function isProjectResource(employee) {
        return (employee.getValue({"fieldId" : "isjobresource"}));
    }

    function hasValidLaborCost(employee) {
        var laborCost = employee.getValue({"fieldId" : "laborcost"});

        return (Boolean(laborCost) && (laborCost > 0));
    }

    exports.beforeSubmit = beforeSubmit;
    return exports;
});
```

Notez que nous transmettons des références au *nouvel* enregistrement dans notre validation car nous ne nous soucions pas des valeurs utilisées auparavant. Nous nous intéressons uniquement aux valeurs sur le point d'être écrites dans la base de données. En 2.0, nous le faisons via la référence `scriptContext.newRecord` , et en 1.0, nous appelons la fonction globale `nlapiGetNewRecord` .

Lorsque les données soumises ne sont pas valides, nous créons et générons une erreur. Dans un événement `beforeSubmit` , afin d'éviter que les modifications ne soient écrites dans la base de données, votre fonction doit `throw` une exception. Souvent, les développeurs essaient de `return false` fonction, en s'attendant à ce que cela soit suffisant, mais cela ne suffit pas. Les objets d'erreur sont créés en 2.0 en utilisant le module `N/error` et en 1.0 en utilisant la fonction globale `nlapiCreateError` ; nous soulevons ensuite une exception en utilisant notre objet d'erreur créé avec le mot-clé `throw` .

Après envoi: détermine si un champ a été modifié

Une fois que l'enregistrement est stocké dans la base de données, nous voulons inspecter ce qui a été modifié dans l'enregistrement. Nous effectuerons cette inspection en comparant les valeurs entre l'ancienne et la nouvelle instance d'enregistrement.

```
// 1.0, Revealing Module pattern
var myNamespace = myNamespace || {};
myNamespace.example = (function () {

    /**
     * User Event 1.0 example detailing usage of the Submit events
     *
     * @appliedtorecord employee
     */
    var exports = {};

    function afterSubmit(type) {
        notifySupervisor();
    }

    function notifySupervisor() {
        // Old and New record instances are retrieved from global functions
        var employee = nlapiGetNewRecord();
        var prevEmployee = nlapiGetOldRecord();

        // If Employee Status didn't change, there's nothing to do
        if (!didStatusChange(employee, prevEmployee)) {
            return;
        }

        // Otherwise, continue with business logic...
    }

    function didStatusChange(employee, prevEmployee) {
        var status = employee.getFieldValue("employeestatus");
        var prevStatus = prevEmployee.getFieldValue("employeestatus");

        /* !! Caution !!
         * Empty fields from the Old record come back as `null`
         * Empty fields from the New record come back as an empty String
         * This means  you cannot simply compare the old and new
```

```
         */
        return ((prevStatus || status) && (status !== prevStatus));
    }

    exports.afterSubmit = afterSubmit;
    return exports;
})();

// 2.0
define(["N/runtime"], function (runtime) {

    /**
     * User Event 2.0 example detailing usage of the Submit events
     *
     * @NApiVersion 2.x
     * @NModuleScope SameAccount
     * @NScriptType UserEventScript
     * @appliedtorecord employee
     */
    var exports = {};

    function afterSubmit(scriptContext) {
        notifySupervisor(scriptContext);
    }

    function notifySupervisor(scriptContext) {
        // Old and New records are simply properties on scriptContext
        var employee = scriptContext.newRecord;
        var prevEmployee = scriptContext.oldRecord;

        // If Employee Status didn't change, there's nothing to do
        if (!didStatusChange(employee, prevEmployee)) {
            return;
        }

        // Otherwise, continue with business logic...
    }

    function didStatusChange(employee, prevEmployee) {
        var status = employee.getValue({"fieldId" : "employeestatus"});
        var prevStatus = prevEmployee.getValue({"fieldId" : "employeestatus"});

        /* !! Caution !!
         * Empty fields from the Old record come back as `null`
         * Empty fields from the New record come back as an empty String
         * This means  you cannot simply compare the old and new
         */
        return ((prevStatus || status) && (status !== prevStatus));
    }

    exports.afterSubmit = afterSubmit;
    return exports;
});
```

Soyez très prudent lorsque vous comparez les valeurs entre les anciens et les nouveaux enregistrements. Les champs vides de l' *ancien* enregistrement sont renvoyés comme `null` , tandis que les champs vides du *nouvel* enregistrement sont renvoyés sous forme de chaîne vide. Cela signifie que vous ne pouvez pas simplement comparer l'ancien avec le nouveau ou que vous obtenez des faux positifs. Toute logique que vous écrivez doit gérer le cas où l'un est `null` et

l'autre est une chaîne vide appropriée.

Lire Evénement utilisateur: événements avant et après la soumission en ligne: https://riptutorial.com/fr/netsuite/topic/7200/evenement-utilisateur--evenements-avant-et-apres-la-soumission

Chapitre 10: Exécuter une recherche

Examples

SS 2.0 Ad Hoc Search

```
require(['N/search'], function(SEARCHMODULE){

    var type = 'transaction';
    var columns = [];
    columns.push(SEARCHMODULE.createColumn({
        name: 'internalid'
    }));
    columns.push(SEARCHMODULE.createColumn({
        name: 'formulanumeric',
        formula: '{quantity}-{quantityshiprecv}'
    }));

    var salesOrdersArray = [123,456,789];
    var filters = [];
    filters.push(['type', 'anyof', 'SalesOrd']);
    filters.push('and');
    filters.push(['mainline', 'is', 'F']);
    filters.push('and');
    filters.push(['internalid', 'anyof', salesOrdersArray]);

    var mySearchObj = {};
    mySearchObj.type = type;
    mySearchObj.columns = columns;
    mySearchObj.filters = filters;

    var mySearch = SEARCHMODULE.create(mySearchObj);
    var resultset = mySearch.run();
    var results = resultset.getRange(0, 1000);
    for(var i in results){
        var result = results[i];
        var row = {};
        for(var k in result.columns){
            log.debug('Result is ' + result.getValue(result.columns[k])); //Access result from
here
        }
    }
});
```

SS 2.0 à partir de la recherche enregistrée

```
require(['N/search'], function(SEARCHMODULE){
    var savedSearchId = 'customsearch_mySavedSearch';
    var mySearch = SEARCHMODULE.load(savedSearchId);
    var resultset = mySearch.run();
    var results = resultset.getRange(0, 1000);
    for(var i in results){
        var result = results[i];
        for(var k in result.columns){
            log.debug('Result is ' + result.getValue(result.columns[k])); //Access result from
```

```
here
        }
    }
});
```

Lire Exécuter une recherche en ligne: https://riptutorial.com/fr/netsuite/topic/6081/executer-une-recherche

Chapitre 11: Exécuter une recherche

Examples

SS 2.0 à partir de la recherche enregistrée

```
require(['N/search'], function(SEARCHMODULE){
    var savedSearchId = 'customsearch_mySavedSearch';
    var mySearch = SEARCHMODULE.load(savedSearchId);
    var resultset = mySearch.run();
    var results = resultset.getRange(0, 1000);
    for(var i in results){
        var result = results[i];
        for(var k in result.columns){
            log.debug('Result is ' + result.getValue(result.columns[k])); //Access result from
here
        }
    }
});
```

SS 2.0 Ad Hoc Search

```
require(['N/search'], function(SEARCHMODULE){

    var type = 'transaction';
    var columns = [];
    columns.push(SEARCHMODULE.createColumn({
        name: 'internalid'
    }));
    columns.push(SEARCHMODULE.createColumn({
        name: 'formulanumeric',
        formula: '{quantity}-{quantityshiprecv}'
    }));

    var salesOrdersArray = [123,456,789];
    var filters = [];
    filters.push(['type', 'anyof', 'SalesOrd']);
    filters.push('and');
    filters.push(['mainline', 'is', 'F']);
    filters.push('and');
    filters.push(['internalid', 'anyof', salesOrdersArray]);

    var mySearchObj = {};
    mySearchObj.type = type;
    mySearchObj.columns = columns;
    mySearchObj.filters = filters;

    var mySearch = SEARCHMODULE.create(mySearchObj);
    var resultset = mySearch.run();
    var results = resultset.getRange(0, 1000);
    for(var i in results){
        var result = results[i];
        var row = {};
        for(var k in result.columns){
            log.debug('Result is ' + result.getValue(result.columns[k])); //Access result from
```

```
here
        }
    }
});
```

Effectuer une recherche résumée

```
// Assuming N/search is imported as `s`
var mySalesOrderSearch = s.create({
    type: 'salesorder'
    // Use the summary property of a Column to perform grouping/summarizing
    columns: [{
        name: 'salesrep',
        summary: s.Summary.GROUP
    },{
        name: 'internalid',
        summary: s.Summary.COUNT
    }],
    filters: [{
        name: 'mainline',
        operator: 'is',
        values: ['T']
    }]
});

mySalesOrderSearch.run().each(function (result) {
    var repId = result.getValue({
        "name": "salesrep",
        "summary": s.Summary.GROUP
    });
    var repName = result.getText({
        "name": "salesrep",
        "summary": s.Summary.GROUP
    });
    var orderCount = parseInt(result.getValue({
            "name": "internalid",
            "summary": s.Summary.COUNT
    }), 10);

    log.debug({
        "title": "Order Count by Sales Rep",
        "details": repName + " has sold " + orderCount + " orders."
    });
});
```

Lire Exécuter une recherche en ligne: https://riptutorial.com/fr/netsuite/topic/6359/executer-une-recherche

Chapitre 12: Exploitation des colonnes de formules dans les recherches enregistrées

Introduction

Les colonnes de formule dans les recherches enregistrées peuvent exploiter de nombreuses fonctionnalités d'Oracle SQL et HTML. Les exemples montrent comment ces fonctionnalités peuvent être utilisées, ainsi que les pièges à éviter.

Examples

Instruction Oracle SQL CASE dans une formule Netsuite

En utilisant une instruction CASE, affichez de manière conditionnelle une expression dans la colonne en fonction des valeurs trouvées dans une autre colonne, appelée «mon royaume pour un OR». Dans l'exemple, le résultat est obtenu lorsque le statut de la transaction est `Pending Fulfillment` ou `Partially Fulfilled` :

```
CASE DECODE( {status}, 'Pending Fulfillment', 1, 'Partially Fulfilled', 1, 0 )
WHEN 1 THEN expression-1
END
```

Analyse d'un nom d'enregistrement hiérarchique à l'aide d'une expression régulière

À l'aide d'une expression régulière, analysez un nom d'enregistrement qui pourrait être hiérarchique. L'expression recherche le deux-points final dans le nom. Il retourne ce qui suit les deux points, ou le nom entier si aucun:

```
regexp_substr( {name} , '[^:]*$' )
```

Construire une chaîne complexe en concaténant plusieurs champs

L'exemple génère une chaîne à partir du nom de l'enregistrement parent, du nom de cet enregistrement et du mémo de cet enregistrement.

```
{createdfrom} || ' ' || {name} || ' ' || {memo}
```

Personnaliser le CSS (feuille de style) pour une colonne en insérant un élément DIV

```
'<div style="font-size:11pt">' || expression || '</div>'
```

Protégez les formules de chaînes contre la corruption et les attaques par

injection

Dans un champ de formule de chaîne, considérez que certaines valeurs peuvent contenir des sous-chaînes qui ressemblent au navigateur HTML. À moins que cela ne soit intentionnel, il est important de protéger les valeurs de la corruption. Ceci est utile pour éviter les attaques par injection: il empêche une personne d'entrer du code HTML dans un champ de commentaire dans une commande Web qui sera interprétée ultérieurement sur le bureau du représentant du service clientèle.

`htf.escape_sc(` *expression* `)`

Protéger les valeurs de champ contre la corruption lors du passage par une URL

`utl_url.escape(` *expression* `)`

Tester la valeur de `mainline` dans une instruction SQL CASE

Dans une formule de recherche sauvegardée, les valeurs possibles de `mainline` sont conçues pour être utiles dans un contexte HTML. Lorsque `mainline` est true, la valeur de `{mainline}` est la chaîne de 1 caractère `*` (astérisque). Lorsque la `mainline` est fausse, la valeur de `{mainline}` est la chaîne de 6 caractères ` ` (espace insécable, HTML codé comme référence d'entité de caractère). Ces valeurs de chaîne peuvent être comparées avec des littéraux de chaîne dans un contexte SQL.

```
CASE
WHEN {mainline} = '*' THEN expression-when-true
WHEN {mainline} = ' ' THEN expression-when-false
END
```

Exemple complexe et réaliste

L'exemple suivant combine plusieurs des techniques abordées ici. Il place un lien hypertexte dans une colonne formatée personnalisée qui, une fois cliquée, ouvre l'enregistrement de commande client associé à une ligne. Le lien hypertexte est conçu pour ouvrir l'enregistrement dans une nouvelle fenêtre ou un nouvel onglet lorsque vous cliquez dessus, et pour afficher une info-bulle lorsque vous survolez. Le champ `internalid` utilisé dans l'URL est protégé du codage URL. Le nom du client, lorsqu'il est disponible, est affiché dans la même colonne, à l'abri de l'encodage HTML.

```
'<div style="font-size:11pt">'
 ||
CASE {mainline}
WHEN '*' THEN '<br>' || htf.escape_sc( regexp_substr( {name} , '[^:]*$' ) ) || '<br>'
END
 ||
'<a alt="" title="Open the order associated with this line." '
 ||
'href="javascript:void(0);" onClick="window.open('''
```

```
 ||
'https://system.na1.netsuite.com/app/accounting/transactions/transaction.nl?id='
 ||
utl_url.escape( {internalid} )
 ||
''' , ''_blank'' )">'
 ||
{number}
 ||
'</a>'
 ||
'</div>'
```

Compter les enregistrements avec avec et sans valeur fournie dans un champ (compter les valeurs manquantes et non manquantes)

A l'aide de la fonction `NVL2()` Oracle SQL, vous pouvez créer une colonne d'affichage contenant une valeur si un champ contient des données et une autre valeur si un champ ne contient pas de données. Par exemple, dans une recherche par entité, transformez la présence d'une adresse de messagerie principale en une colonne d'affichage de texte:

```
NVL2( {email} , 'YES' , 'NO' )
```

Cela vous permet de compter les enregistrements sous-totalisés par la présence ou l'absence d'une adresse électronique:

```
Field: Internal ID
Summary Type: Count

Field: Formula (Text)
Summary Type: Group
Formula: NVL2( {email} , 'YES' , 'NO' )
```

Lire Exploitation des colonnes de formules dans les recherches enregistrées en ligne: https://riptutorial.com/fr/netsuite/topic/8298/exploitation-des-colonnes-de-formules-dans-les-recherches-enregistrees

Chapitre 13: Gouvernance

Remarques

Gouvernance

La «gouvernance» est le nom donné au système de NetSuite pour détecter et arrêter les scripts de longue durée, emballés ou nécessitant beaucoup de ressources.

Chaque type de script a des limites de gouvernance qu'il ne peut pas dépasser et il existe quatre types de limites de gouvernance pour chaque type de script.

- Limite d'utilisation de l'API
- Limite du nombre d'instructions
- Limite de temporisation
- Limite d'utilisation de la mémoire

Si un script dépasse sa limite de gouvernance dans **l'un** de ces quatre domaines, NetSuite **lance une exception *insaisissable*** et termine immédiatement le script.

Limite d'utilisation de l'API

NetSuite limite l'utilisation de l'API de vos scripts avec un système basé sur des "unités d'utilisation". Certains appels d'API NetSuite, en particulier ceux qui exécutent une action de lecture ou d'écriture sur la base de données, coûtent un nombre spécifique d'unités à chaque appel. Chaque type de script dispose alors d'un nombre maximal d'unités pouvant être utilisé lors de chaque exécution du script.

Si un script dépasse la limite d'utilisation de l'API, NetSuite termine le script en `SSS_USAGE_LIMIT_EXCEEDED` *une erreur* `SSS_USAGE_LIMIT_EXCEEDED` *.*

Vous trouverez ci-dessous quelques exemples de coûts unitaires pour les opérations courantes. Pour obtenir une liste exhaustive des coûts de gouvernance, consultez l'article intitulé «Gouvernance des API» dans l'aide de NetSuite.

Opération	Coût unitaire
Chargement d'une recherche enregistrée	`5`
Récupération des résultats de recherche	`10`
Planification d'une tâche	`10`
Demander une URL	`10`

Opération	Coût unitaire
Envoi d'un email	10
Créer un enregistrement personnalisé	2
Création d'un enregistrement d'employé	5
Création d'un enregistrement de commande client	10
Enregistrement d'un enregistrement personnalisé	4
Enregistrement d'un enregistrement de contact	10
Enregistrement d'un enregistrement de commande d'achat	20

Différentes opérations utilisent des quantités d'unités différentes et certaines opérations coûtent un montant différent en fonction du type d'enregistrement utilisé. Plus le nombre d'unités par fonction est élevé, plus le délai d'exécution est généralement long.

Les transactions sont les plus importantes des types d'enregistrements. Travailler avec elles coûte donc le plus grand nombre d'unités. À l'inverse, les enregistrements personnalisés sont très légers et ne coûtent donc pas beaucoup d'unités. Les enregistrements NetSuite standard qui *ne* sont *pas des* transactions, tels que les clients, les employés ou les contacts, se situent entre les deux en termes de coût.

Voici les limites d'utilisation par type de script:

Type de script	Limite d'utilisation
Client	1000
Événement utilisateur	1000
Suitelet	1000
Portlet	1000
Action de workflow	1000
RESTlet	5000
Prévu	10 000
Carte / Réduire	10 000
Installation groupée	10 000
Mise à jour de masse	10 000 par enregistrement

Limites de délai d'expiration et d'instruction

NetSuite utilise également le système de gouvernance pour détecter et arrêter les scripts en cours d'exécution en utilisant un mécanisme de temporisation et un compteur d'instructions.

Si l'exécution d'un script prend trop de temps, NetSuite l'arrêtera en lançant une erreur `SSS_TIME_LIMIT_EXCEEDED` .

En outre, les scripts d'exécution peuvent être détectés et arrêtés en fonction de leur «nombre d'instructions». Si les limites du nombre d'instructions définies sont dépassées, NetSuite arrête le script en `SSS_INSTRUCTION_COUNT_EXCEEDED` une erreur `SSS_INSTRUCTION_COUNT_EXCEEDED` .

Il n'y a malheureusement **pas de** documentation d'aide définissant:

- le délai d'attente pour chaque type de script
- les limites de comptage des instructions pour chaque type de script
- ce qui constitue une seule "instruction"

Il est simplement important de savoir que si vous rencontrez l'erreur `SSS_TIME_LIMIT_EXCEEDED` ou l'erreur `SSS_INSTRUCTION_COUNT_EXCEEDED` dans l'un de vos scripts, le traitement prend trop de temps. Focalisez votre enquête sur vos structures de boucle pour déterminer les optimisations possibles.

Limite d'utilisation de la mémoire

Si votre script dépasse la limite d'utilisation de la mémoire, NetSuite mettra fin à votre script en `SSS_MEMORY_USAGE_EXCEEDED` une erreur `SSS_MEMORY_USAGE_EXCEEDED` .

Chaque variable déclarée, chaque fonction définie, chaque objet stocké contribue à l'utilisation de la mémoire de votre script.

Le **script programmé** et le **script Map / Reduce** ont tous deux des limites de mémoire documentées de `50MB` Il existe également une limite documentée de `10MB` pour la taille de toute chaîne transmise ou renvoyée par un RESTlet. Il n'y a pas d'autre documentation sur les limites spécifiques d'un script donné.

Examples

Combien reste-t-il d'appareils?

Dans SuiteScript 1.0, utilisez `nlobjContext.getRemainingUsage()` pour récupérer les unités restantes. Une référence `nlobjContext` est extraite à l'aide de la fonction globale `nlapiGetContext` .

```
// 1.0
var context = nlapiGetContext();
nlapiLogExecution("DEBUG", "Governance Monitoring", "Remaining Usage = " +
context.getRemainingUsage());

nlapiSearchRecord("transaction"); // uses 10 units
```

```
nlapiLogExecution("DEBUG", "Governance Monitoring", "Remaining Usage = " +
context.getRemainingUsage());
```

Dans SuiteScript 2.0, utilisez la méthode `getRemainingUsage` de l'objet `Script` du module `N/runtime` .

```
// 2.0
require(["N/log", "N/runtime", "N/search"], function (log, runtime, s) {
    var script = runtime.getCurrentScript();
    log.debug({
        "title": "Governance Monitoring",
        "details": "Remaining Usage = " + script.getRemainingUsage()
    });

    s.load({"id":"customsearch_mysearch"}); // uses 5 units
    log.debug({
        "title": "Governance Monitoring",
        "details": "Remaining Usage = " + script.getRemainingUsage()
    });
});
```

Lire Gouvernance en ligne: https://riptutorial.com/fr/netsuite/topic/7227/gouvernance

Chapitre 14: Mass Delete

Introduction

Cet exemple montre comment supprimer en masse des enregistrements dans NetSuite en exploitant la fonctionnalité de mise à jour en masse. En règle générale, on nous dit de ne pas supprimer les enregistrements, mais de rendre les enregistrements inactifs, mais si vous devez le faire, alors ce petit script ne fait que cela. Une fois le script déployé en tant que type de script «Mass Update», accédez simplement à Lists> Mass Update> Mass Updates> Custom Updates. Vous devriez voir votre suppression de masse. Ensuite, configurez vos critères de recherche dans votre suppression de masse et effectuez un aperçu pour valider vos données avant de les supprimer.

Examples

Supprimer en fonction des critères de recherche

```
/**
 *  NetSuite will loop through each record in your search
 *  and pass the record type and id for deletion
 *  Try / Catch is useful if you wish to handle potential errors
 */

function MassDelete(record_type, record_id)
{
    try
    {
        nlapiDeleteRecord(record_type, record_id)
    }
    catch (err)
    {
        var errMessage = err;
        if(err instanceof nlobjError)
        {
            errMessage = errMessage + ' ' + err.getDetails() + ' ' + 'Failed to Delete ID : '
+ record_id;
        }
        nlapiLogExecution('ERROR', 'Error', errMessage);
        return err
    }
}
```

Lire Mass Delete en ligne: https://riptutorial.com/fr/netsuite/topic/9062/mass-delete

Chapitre 15: Présentation du type de script

Introduction

Vous créez des personnalisations SuiteScript en utilisant un système piloté par des événements. Vous définissez différents types d'enregistrements de script, chacun ayant son propre ensemble d'événements, et dans votre fichier source, vous définissez des fonctions qui seront appelées pour gérer ces événements lorsqu'ils se produisent.

Les scripts sont l'un des principaux composants avec lesquels vous allez concevoir et développer vos applications. Le but de cet article est simplement de se familiariser avec les types de script et les événements disponibles.

Examples

Le script client

Le script client est l'un des types de script les plus utilisés et les plus complexes à votre disposition. Comme son nom l'indique, le script client s'exécute dans le navigateur, c'est-à-dire côté client. C'est le seul type de script qui s'exécute du côté client. tous les autres s'exécuteront du côté serveur de NetSuite.

La principale utilisation du script client est de répondre aux interactions des utilisateurs avec les formulaires d'enregistrement dans l'interface utilisateur de NetSuite.

Dès que l'utilisateur charge un formulaire d'enregistrement en mode Edition, un événement `pageInit` est déclenché, que nous pouvons utiliser pour exécuter du code lorsque le formulaire est initialisé, avant que l'utilisateur puisse interagir avec lui.

Chaque fois que l'utilisateur modifie un champ du formulaire, une série d'événements se déclenche:

1. Un événement `validateField` déclenche pour nous permettre de valider la valeur que l'utilisateur tente d'entrer dans le champ. Nous pouvons l'utiliser pour accepter ou empêcher le changement.
2. Un événement `fieldChanged` se déclenche puis nous permet de répondre à la nouvelle valeur dans le champ.
3. Enfin, un événement `postSourcing` - `postSourcing` déclenche après que tous les champs dépendants ont également été inclus dans leurs valeurs. Cela nous permet de réagir au changement *et de* nous assurer que nous travaillons avec toutes les données correctes.

Cette série d'événements se déclenche, que l'utilisateur modifie un champ de corps ou un champ de sous-liste.

Lorsque l'utilisateur modifie les lignes de sous-liste, une autre série d'événements sera déclenchée:

1. Un événement `lineInit` est déclenché à chaque fois que l'utilisateur sélectionne initialement une ligne nouvelle ou existante avant de pouvoir apporter des modifications aux champs de la ligne.
2. Chaque fois que l'utilisateur clique sur le bouton *Ajouter* pour ajouter une nouvelle ligne, un événement `validateLine` est déclenché, ce qui nous permet de vérifier que la ligne entière est valide et peut être ajoutée à l'enregistrement.
3. Chaque fois que l'utilisateur clique sur le bouton *Insérer* pour ajouter une nouvelle ligne au-dessus d'une ligne existante, un événement `validateInsert` est déclenché, ce qui fonctionne exactement comme l'événement `validateLine` .
4. De même, chaque fois que l'utilisateur essaie de supprimer une ligne, un `validateDelete` qui permet d'autoriser ou de refuser la suppression de la ligne est déclenché.
5. [SuiteScript 1.0 uniquement] Enfin, après la réussite de l'événement de validation approprié, si la modification de la ligne a également modifié le montant total d'une transaction, un événement de `recalc` est déclenché, ce qui nous permet de répondre à la modification du montant de notre transaction. transaction.
6. [SuiteScript 2.0 uniquement] Enfin, après la `sublistChanged` événement de validation approprié, un événement `sublistChanged` est déclenché pour nous permettre de répondre au changement de ligne terminé.

Enfin, lorsque l'utilisateur clique sur le bouton *Enregistrer* de l'enregistrement, un événement `saveRecord` est déclenché, ce qui nous permet de valider si l'enregistrement est valide et peut être enregistré. Nous pouvons empêcher la sauvegarde de se produire ou lui permettre de poursuivre cet événement.

Le script client a de loin le plus grand nombre d'événements de tout type de script et la relation la plus complexe entre ces événements.

Le script d'événement utilisateur

Le script d'événement utilisateur est étroitement lié au script client. Les événements de ce type de script sont à nouveau déclenchés lors du chargement ou de l'enregistrement d'un enregistrement, mais s'exécutent plutôt du côté du serveur. En tant que tel, il ne peut pas être utilisé pour répondre immédiatement aux changements de champs, mais il ne se limite pas aux seuls utilisateurs qui interagissent avec l'enregistrement sur un formulaire.

Les scripts d'événements utilisateur s'exécuteront peu importe d'où provient la demande de chargement ou d'envoi, qu'il s'agisse d'un utilisateur travaillant dans l'interface utilisateur, d'une intégration tierce ou d'un autre script interne effectuant la requête.

Lorsqu'un processus ou un utilisateur tente de lire un enregistrement hors de la base de données, l'événement `beforeLoad` événement utilisateur est déclenché. Nous pouvons l'utiliser pour pré-traiter des données, définir des valeurs par défaut ou manipuler le formulaire d'interface utilisateur avant que l'utilisateur ne le voit.

Une fois qu'un processus ou un utilisateur tente de soumettre un enregistrement à la base de données, qu'il s'agisse de créer un nouvel enregistrement, de modifier un enregistrement existant ou de supprimer un enregistrement, la séquence suivante se produit:

1. Tout d'abord, avant que la demande ne `beforeSubmit` réellement à la base de données, un événement `beforeSubmit` déclenche. Nous pouvons utiliser cet événement, par exemple, pour nettoyer l'enregistrement avant qu'il entre dans la base de données.
2. La demande est envoyée à la base de données et l'enregistrement est créé / modifié / supprimé en conséquence.
3. Une fois le traitement de la base de données terminé, un événement `afterSubmit` déclenche. Nous pouvons utiliser cet événement, par exemple, pour envoyer des notifications par courrier électronique des modifications ou pour vous synchroniser avec des systèmes tiers intégrés.

Vous pouvez également regarder cette série de vidéos qui aident à visualiser les événements de ce type de script.

Les scripts programmés et de mappage / réduction

Nous pouvons utiliser deux types de scripts pour exécuter le traitement en arrière-plan sur un intervalle spécifique et régulier. Ce sont les scripts *Scheduled* et *Map / Reduce* . Notez que le type de script *Map / Reduce* est uniquement disponible dans SuiteScript 2.0. Le script *programmé* est disponible pour les versions 1.0 et 2.0.

Le script planifié ne comporte qu'un seul événement d' `execute` qui est déclenché selon le planning que vous définissez. Par exemple, vous pouvez exécuter un script de nuit qui applique des paiements aux factures ou un script horaire qui synchronise les données avec un système externe. Lorsque l'intervalle de temps est écoulé, NetSuite déclenche cet événement d' `execute` sur votre script planifié.

Le script Map / Reduce fonctionne de la même manière, mais une fois déclenché, il divise le traitement en quatre phases distinctes:

1. La phase `getInputData` est l'endroit où vous collectez toutes les données d'entrée dont vous aurez besoin pour terminer le processus technique. Vous pouvez utiliser cette phase pour effectuer des recherches, lire des enregistrements et regrouper vos données dans une structure de données déchiffrable.
2. NetSuite transmet automatiquement les résultats de votre phase `getInputData` à la deuxième phase, appelée `map` . Cette phase est responsable du regroupement logique de vos données d'entrée pour le traitement. Par exemple, si vous appliquez des paiements à des factures, vous pouvez d'abord regrouper les factures par le client.
3. Les résultats de la phase de `map` sont ensuite transmis à la phase de `reduce` , qui correspond au traitement proprement dit. C'est ici que, conformément à notre exemple, vous appliquez réellement les paiements aux factures.
4. Enfin, une phase `summary` contenant des données sur les résultats de tous vos traitements au cours des trois phases précédentes est appelée. Vous pouvez l'utiliser pour générer des rapports ou envoyer des e-mails dont le traitement est terminé.

Le principal avantage du script Map / Reduce est que NetSuite met automatiquement le traitement en parallèle pour vous sur plusieurs files d'attente, le cas échéant.

Ces deux types de script ont une limite de gouvernance extrêmement étendue. Vous pouvez donc

également les utiliser pour le traitement en bloc ou les processus d'arrière-plan généralement longs.

L'intervalle le plus court de ces types de script peut être configuré pour s'exécuter toutes les 15 minutes.

Ces deux types de script peuvent également être appelés à la demande par les utilisateurs ou par d'autres scripts, si nécessaire.

Les scripts Suitelet et Portlet

Nous souhaitons souvent créer des pages d'interface personnalisées dans NetSuite; entrez la Suitelet. Le script Suitelet est conçu pour générer des pages d'interface utilisateur personnalisées internes. Les pages peuvent être du format HTML libre ou utiliser les API de Builder UI de NetSuite pour créer des formulaires conformes à l'apparence de NetSuite.

Lorsqu'il est déployé, Suitelet reçoit sa propre URL unique. Suitelet possède alors un événement de `render` unique appelé chaque fois que cette URL est frappée avec une requête HTTP `GET` ou `POST` . En règle générale, la réponse à l' `GET` demande serait de rendre la forme elle - même, et la forme serait `POST` à lui - même pour le traitement des données de formulaire.

Nous pouvons également tirer parti de Suitelets pour créer des progressions d'interface de type assistant à l'aide des composants de l'interface utilisateur "Assistant" de NetSuite.

Les portlets sont extrêmement similaires à Suitelets, sauf qu'ils sont spécifiquement utilisés pour créer des widgets de tableau de bord personnalisés plutôt que des pages personnalisées complètes. En dehors de cela, les deux types de script fonctionnent de manière très similaire.

Le RESTlet

Les RESTlets nous permettent de créer des points de terminaison personnalisés basés sur REST dans NetSuite; ainsi, les RESTlets constituent la colonne vertébrale de presque toutes les intégrations dans NetSuite.

Les RESTlets fournissent des gestionnaires d'événements individuels pour quatre des méthodes de requête HTTP les plus utilisées:

- `GET`
- `POST`
- `PUT`
- `DELETE`

Lorsqu'un RESTlet reçoit une demande, il achemine la demande vers la fonction de gestionnaire d'événements appropriée en fonction de la méthode de requête HTTP utilisée.

L'authentification à un RESTlet peut être effectuée via une session utilisateur, des en-têtes HTTP ou des jetons OAuth.

Le script de mise à jour de masse

En utilisant le script de mise à jour de masse, nous pouvons créer des mises à jour de masse personnalisées pour les utilisateurs. Cela fonctionne comme une mise à jour de masse normale, où l'utilisateur sélectionne le type de mise à jour de masse, crée une recherche qui renvoie les enregistrements à mettre à jour, puis chaque résultat de recherche est transmis individuellement au script de mise à jour de masse personnalisé.

Le script fournit un `each` gestionnaire d'événements qui reçoit l'ID interne et le type d' enregistrement de l'enregistrement qui doit être mis à jour.

Les scripts de mise à jour en masse doivent être déclenchés manuellement par les utilisateurs via l'interface de mise à jour standard.

Les scripts de mise à jour en masse ont une limite de gouvernance extrêmement élevée et sont destinés au traitement en masse personnalisé, couramment utilisé.

Le script d'action de workflow

Les flux de travail peuvent être quelque peu limités dans leurs fonctionnalités. Par exemple, les flux de travail ne peuvent pas interagir avec les éléments de campagne. Le type de script Action de flux de travail est destiné à être appelé par un flux de travail pour ajouter une fonctionnalité de script afin de réaliser ce que le flux de travail lui-même ne peut pas.

Les actions de workflow ont un seul `onAction` événement `onAction` qui sera `onAction` par le workflow.

Le script d'installation de l'ensemble

Enfin, nous avons le type de script Installation de l'ensemble, qui fournit plusieurs événements qui nous permettent d'interagir avec l'installation, la mise à jour et la désinstallation d'un ensemble particulier. C'est un type de script rarement rencontré, mais il est important d'en être conscient.

L'installation de l'ensemble inclut les gestionnaires d'événements suivants, qui doivent être assez explicites:

- `beforeInstall`
- `afterInstall`
- `beforeUpdate`
- `afterUpdate`
- `beforeUninstall`

Lire Présentation du type de script en ligne:
https://riptutorial.com/fr/netsuite/topic/7829/presentation-du-type-de-script

Chapitre 16: Recherche de données à partir d'enregistrements associés

Introduction

Lors du traitement d'un enregistrement donné, vous devrez récupérer des données d'un de ses enregistrements associés. Par exemple, lorsque vous utilisez une commande client donnée, vous devrez peut-être extraire des données du représentant commercial associé. Dans la terminologie de SuiteScript, cela s'appelle une **recherche** .

La fonction de recherche est fournie par la fonction globale `nlapiLookupField` dans SuiteScript 1.0 et la méthode `lookupFields` du module `N/search` dans SuiteScript 2.0

Syntaxe

- nlapiLookupField (recordType, recordId, columns);

Paramètres

Paramètre	Détails
recordType	`String` - ID interne du type d'enregistrement recherché (par exemple, `salesorder` , `employee`)
recordId	`String` ou `Number` - L'ID interne de l'enregistrement recherché
colonnes	`String` ou `String[]` - Liste des champs à extraire de l'enregistrement. Les ID de champ peuvent être référencés à partir de la section "Colonnes de recherche" du navigateur d'enregistrements . Les champs `salesrep.email` peuvent être récupérés à l'aide de la syntaxe à points (par exemple, `salesrep.email`)

Remarques

Performance

Une recherche est juste un raccourci pour effectuer une recherche qui filtre l'ID interne d'un seul enregistrement pour le résultat. Sous le capot, les recherches effectuent effectivement une recherche, de sorte que la performance sera similaire à celle d'une recherche qui renvoie un seul enregistrement.

Cela signifie également qu'une recherche fonctionnera plus rapidement que le chargement de

l'enregistrement pour récupérer les mêmes informations.

Limites

Les recherches ne peuvent être utilisées que pour récupérer des données de corps. Vous ne pouvez pas extraire des données des sous-listes d'un enregistrement associé à l'aide d'une recherche. Si vous avez besoin de données de sous-liste, vous devrez soit effectuer une recherche, soit charger l'enregistrement correspondant.

Examples

[1.0] Champ unique de recherche

```
/**
 * An example of nlapiLookupField to retrieve a single field from a related record
 */

// Get the Sales Rep record ID
var repId = nlapiGetFieldValue("salesrep");

// Get the name of the Sales Rep
var repName = nlapiGetFieldText("salesrep");

// Retrieve the email address from the associated Sales Rep
var repEmail = nlapiLookupField("employee", repId, "email");

console.log(repEmail);
console.log(repName + "'s email address is " + repEmail);
```

[1.0] Recherche de champs multiples

```
/**
 * An example of nlapiLookupField to retrieve multiple fields from a related record
 */

// Get the Sales Rep record ID
var repId = nlapiGetFieldValue("salesrep");

// Retrieve multiple fields from the associated Sales Rep
var repData = nlapiLookupField("employee", repId, ["email", "firstname"]);

console.log(repData);
console.log(repData.firstname + "'s email address is " + repData.email);
```

[1.0] Recherche de champs joints

```
/**
 * An example of nlapiLookupField to retrieve joined fields from a related record
 */

var repId = nlapiGetFieldValue("salesrep");
```

```
// Retrieve multiple fields from the associated Sales Rep
var repData = nlapiLookupField("employee", repId, ["email", "firstname", "department.name"]);

console.log(repData);
console.log(repData.firstname + "'s email address is " + repData.email);
console.log(repData.firstname + "'s department is " + repData["department.name"]);
```

[2.0] Champ unique de recherche

```
require(["N/search", "N/currentRecord"], function (s, cr) {

    /**
     * An example of N/search#lookupFields to retrieve a single field from a related record
     */
    (function () {

        var record = cr.get();

        // Get the Sales Rep record ID
        var repId = record.getValue({
            "fieldId": "salesrep"
        });

        // Get the name of the Sales Rep
        var repName = record.getText({
            "fieldId": "salesrep"
        });

        // Retrieve the email address from the associated Sales Rep
        var repData = s.lookupFields({
            "type": "employee",
            "id": repId,
            "columns": ["email"]
        });

        console.log(repData);
        console.log(repName + "'s email address is " + repData.email);
    })();
});
```

[2.0] Recherche de champs multiples

```
require(["N/search", "N/currentRecord"], function (s, cr) {

    /**
     * An example of N/search#lookupFields to retrieve multiple fields from a related record
     */
    (function () {

        var record = cr.get();

        // Get the Sales Rep record ID
        var repId = record.getValue({
            "fieldId": "salesrep"
        });
```

```
        // Retrieve the email address from the associated Sales Rep
        var repData = s.lookupFields({
            "type": "employee",
            "id": repId,
            "columns": ["email", "firstname"]
        });

        console.log(repData);
        console.log(repData.firstname + "'s email address is " + repData.email);
    })();
});
```

[2.0] Recherche de champs joints

```
require(["N/search", "N/currentRecord"], function (s, cr) {

    /**
     * An example of N/search#lookupFields to retrieve joined fields from a related record
     */
    (function () {

        var record = cr.get();

        // Get the Sales Rep record ID
        var repId = record.getValue({
            "fieldId": "salesrep"
        });

        // Retrieve the email address from the associated Sales Rep
        var repData = s.lookupFields({
            "type": "employee",
            "id": repId,
            "columns": ["email", "firstname", "department.name"]
        });

        console.log(repData);
        console.log(repData.firstname + "'s email address is " + repData.email);
        console.log(repData.firstname + "'s department is " + repData["department.name"]);
    })();
});
```

Lire Recherche de données à partir d'enregistrements associés en ligne: https://riptutorial.com/fr/netsuite/topic/9068/recherche-de-donnees-a-partir-d-enregistrements-associes

Chapitre 17: Recherches avec un grand nombre de résultats

Introduction

Suitescript 2.0 propose 4 méthodes pour gérer les résultats de la recherche.

Ils ont une syntaxe, des limites et une gouvernance différentes et conviennent à différentes situations. Nous allons nous concentrer ici sur la façon d'accéder à **TOUS** les résultats de recherche, en utilisant chacune de ces méthodes.

Examples

Utilisation de la méthode Search.ResultSet.each

C'est la méthode la plus courte, la plus facile et la plus utilisée. Malheureusement, il a une limitation majeure - ne peut pas être utilisé sur des recherches avec plus de 4000 résultats (lignes).

```
// Assume that 'N/search' module is included as 'search'

var s = search.create({
    type : search.Type.TRANSACTION,
    columns : ['entity','amount'],
    filters : [  ['mainline', 'is', 'T'],
          'and', ['type', 'is', 'CustInvc'],
          'and', ['status', 'is', 'open']
        ]
});

var resultSet = s.run();

// you can use "each" method on searches with up to 4000 results
resultSet.each( function(result) {

    // you have the result row. use it like this....
    var transId = result.id;
    var entityId = result.getValue('entity');
    var entityName = result.getText('entity');
    var amount = result.getValue('amount');

    // don't forget to return true, in order to continue the loop
    return true;

});
```

Utilisation de la méthode ResultSet.getRange

Afin d'utiliser getRange pour gérer le grand nombre de résultats, nous devrons prendre en compte

les éléments suivants:

1. getRange a 2 paramètres: **start** et **end** . Toujours positif, toujours (début <fin)
2. **start** est l'index inclusif du premier résultat à retourner
3. **end** est l'index exclusif du dernier résultat à retourner
4. Si le nombre de résultats disponibles est inférieur à celui demandé, le tableau contiendra moins que les entrées de début et de fin. Par exemple, s'il n'y a que 25 résultats de recherche, getRange (20, 30) renverra un tableau de 5 objets search.Result.
5. Bien que la phrase d'aide ci-dessus ne le dise pas directement, le **début** et la **fin** peuvent être en dehors de la plage des résultats disponibles. Dans le même exemple - s'il n'y a que 25 résultats de recherche, getRange (100, 200) renverra un tableau vide []
6. Maximum 1000 lignes à la fois. (fin - début) <= 1000

```
// Assume that 'N/search' module is included as 'search'

// this search will return a lot of results (not having any filters)
var s = search.create({
    type: search.Type.TRANSACTION,
    columns : ['entity','amount'],
    filters: []
});

var resultSet = s.run();

// now take the first portion of data.
var currentRange = resultSet.getRange({
        start : 0,
        end : 1000
});

var i = 0;  // iterator for all search results
var j = 0;  // iterator for current result range 0..999

while ( j < currentRange.length ) {

    // take the result row
    var result = currentRange[j];
    // and use it like this....
    var transId = result.id;
    var entityId = result.getValue('entity');
    var entityName = result.getText('entity');
    var amount = result.getValue('amount');

    // finally:
    i++; j++;
    if( j==1000 ) {   // check if it reaches 1000
        j=0;          // reset j an reload the next portion
        currentRange = resultSet.getRange({
            start : i,
            end : i+1000
        });
    }
}
```

Permet de calculer la gouvernance. Nous avons 1 + compte / 1000 appels getRange prenant chacun 10 unités, donc:

G = (1 + compte / 1000) * 10

Exemple: 9500 lignes prendront 100 unités

Utilisation de la méthode Search.PagedData.fetch

PagedData est un objet renvoyé par la méthode Search.runPaged (options). Cela fonctionne exactement comme le font les recherches d'interface utilisateur. L'objet PagedData contient 2 propriétés importantes, que vous pouvez voir à droite de l'en-tête des résultats dans la page de résultats de recherche de l'interface utilisateur de Netsuite:

- **count** (le nombre total des résultats)
- **pageRanges** (liste de pages, disponible dans l'interface utilisateur en tant que sélecteur de zone de liste déroulante)

Le paramètre options.pageSize est limité à 1000 lignes de résultat.
La méthode **PagedData.fetch** est utilisée pour récupérer la partie de résultat souhaitée (indexée par le paramètre pageIndex). Avec un peu plus de code, vous recevez la même fonction de rappel pratique que Search.ResultSet.each, sans avoir la limitation de 4000 lignes.

```
// Assume that 'N/search' module is included as 'search'

// this search will return a lot of results (not having any filters)
var s = search.create({
    type: search.Type.TRANSACTION,
    columns : ['entity','amount'],
    filters : []
});

var pagedData = s.runPaged({pageSize : 1000});

// iterate the pages
for( var i=0; i < pagedData.pageRanges.length; i++ ) {

    // fetch the current page data
    var currentPage = pagedData.fetch(i);

    // and forEach() thru all results
    currentPage.data.forEach( function(result) {

        // you have the result row. use it like this....
        var transId = result.id;
        var entityId = result.getValue('entity');
        var entityName = result.getText('entity');
        var amount = result.getValue('amount');

    });

}
```

Permet de calculer la gouvernance. Nous avons 5 unités pour les appels runPaged () et 1 + count / 1000 pagedData.fetch en prenant chacun 5 unités, donc:

G = 5 + ceil (nombre / 1000) * 5

Exemple: 9500 lignes prendront 55 unités. Environ la moitié des unités de gouvernance de getRange.

Utilisation du script Map / Reduce dédié

Pour des résultats de recherche très importants, vous pouvez utiliser un script Map / Reduce dédié. C'est beaucoup plus gênant, mais parfois inévitable. Et parfois pourrait être très pratique. L'astuce est que, dans l'étape Obtenir les données d'entrée, vous pouvez fournir au moteur NS non pas les données réelles (c.-à-d. Le résultat du script), mais simplement la définition de la recherche. NS effectuera la recherche pour vous sans compter les unités de gouvernance. Ensuite, chaque ligne de résultat sera transmise à l'étape Map.
Bien sûr, il y a une limitation: la taille totale persistante des données pour un script map / Reduce ne doit pas dépasser 50 Mo. Dans un résultat de recherche, chaque clé et la taille sérialisée de chaque valeur sont comptabilisées dans la taille totale. "Sérialisé" signifie que la ligne de résultat de la recherche est convertie en chaîne avec JSON.stringify. Ainsi, la taille de la valeur est proportionnelle au nombre de colonnes de résultats de recherche dans un jeu de résultats. Si vous rencontrez des problèmes avec l'erreur STORAGE_SIZE_EXCEEDED, envisagez de réduire les colonnes, de combiner les formules, de regrouper les résultats ou même de diviser la recherche en plusieurs sous-recherches, qui pourraient être exécutées dans les étapes Map ou Reduce.

```
/**
 * @NApiVersion 2.0
 * @NScriptType MapReduceScript
 */
define(['N/search'], function(search) {

function getInputData()
{
    return search.create({
        type: search.Type.TRANSACTION,
        columns : ['entity','amount'],
        filters : []
    });
}

function map(context)
{
    var searchResult = JSON.parse(context.value);
    // you have the result row. use it like this....
    var transId = searchResult.id;
    var entityId = searchResult.values.entity.value;
    var entityName = searchResult.values.entity.text;
    var amount = searchResult.values.amount.value;

    // if you want to pass some part of the search result to the next stage
    // write it to context:
    context.write(entityId, transId);
}

function reduce(context)
{
   // your code here ...
```

```
    }

    function summarize(summary)
    {
        // your code here ...
    }

    return {
        getInputData: getInputData,
        map: map,
        reduce: reduce,
        summarize: summarize
    };
});
```

Bien entendu, l'exemple ici est simplifié, sans traitement des erreurs et est donné juste pour être comparé aux autres. D'autres exemples sont disponibles dans les exemples de type de script Map / Reduce Script dans le centre d'aide de NS

Lire Recherches avec un grand nombre de résultats en ligne:
https://riptutorial.com/fr/netsuite/topic/10687/recherches-avec-un-grand-nombre-de-resultats

Chapitre 18: Recherches par script avec des expressions de filtre

Introduction

Lorsque vous créez des recherches avec Suitescript, vous pouvez fournir comme "filtres" un tableau d'objets filtre ou une expression de filtre. La seconde option est plus lisible et vous offre une option très flexible pour fournir des expressions imbriquées (jusqu'à 3 niveaux) en utilisant non seulement les opérateurs par défaut "AND", mais aussi "OR" et "NOT".

Examples

Terme du filtre

Pour comprendre les expressions du filtre, nous devons commencer par Filter Term. C'est un **tableau** simple **de chaînes** contenant au moins 3 éléments:

1. **Filtrer** (Champ / Champ de jointure / Formule / Résumé)
2. **Opérateur** (search.Operator)
3. **Valeurs** (valeur de chaîne (ou tableau de valeurs de chaîne), à utiliser comme paramètre de filtre)

```
// Simple example:
['amount', 'equalto', '0.00']

// When the field is checkbox, use 'T' or 'F'
['mainline', 'is', 'T']

// You can use join fields
['customer.companyname', 'contains', 'ltd']

// summary filter term
['sum(amount)', 'notlessthan', '170.50']

// summary of joined fields
['sum(transaction.amount)', 'greatherthan', '1000.00']

// formula:
["formulatext: NVL({fullname},'John')", "contains", "ohn"]

// and even summary formula refering joined fields:
['sum(formulanumeric: {transaction.netamount} + {transaction.taxtotal})',
'greaterthanorequalto','100.00']

// for selection fields, you may use 'anyof'
// and put values in array
['type','anyof',['CustInvc','VendBill','VendCred']]

// when using unary operator, like isempty/isnotempty
// don't forget that the filter term array contains at least 3 elements
```

```
 // and put an empty string as third:
 ['email', 'isnotempty', '']

 // you may have more than 3 elements in Filter Term array,
 // when the operator requires more than one value:
 ['grossamount','between','100.00','200.00']
```

Dans certains champs de sélecteur, vous pouvez utiliser des valeurs spéciales.

```
 // When filtering the user related fields, you can use:
 // Me (Current user): @CURRENT@
 // My Team (somebody from the team I am leading): @HIERARCHY@
 ['nextapprover','is','@CURRENT@']

 // The same is valid for Subsidiary, Department, Location, etc.
 // @CURRENT@ means MINE (Subsidiary, Department...)
 // @HIERARCHY@ means MINE or DESCENDANTS
 ["subsidiary","is","@HIERARCHY@"]

 // on selection fields you may use @ANY@ and @NONE@
 ['nextapprover','is','@NONE@']
```

Expression de filtre

L'expression de filtre simple est également un **tableau** . Il contient un ou plusieurs termes de filtre, associés aux opérateurs - "ET", "OU", "NON". (Les opérateurs sont insensibles à la casse):

```
[
  ['mainline', 'is', 'T'],
  'and', ['type','anyof',['CustInvc','CustCred']],
  'and', 'not', ['amount', 'equalto', '0.00'],
  'or', ['customer.companyname', 'contains', 'ltd']
]
```

Des expressions de filtre plus complexes peuvent contenir des termes de filtre **ET** des expressions de filtre imbriquées, associées à des opérateurs. Pas plus de 3 niveaux d'expressions imbriquées sont autorisés:

```
[
  ['mainline', 'is', 'T'],
  'and', ['type','anyof',['CustInvc','CustCred']],
  'and', [ ['customer.companyname', 'contains', 'ltd'],
           'or', ['customer.companyname', 'contains', 'inc']
         ],
  'and', [ ['subsidiary', 'is', 'HQ'],
           'or', ['subsidiary', 'anyof', '@HIERARCHY@']
         ],
  'and', ['trandate', 'notbefore', 'yesterday']
 ]
```

Et enfin, mettons tout cela dans un exemple SS2.0:

```
var s = search.create({
    type    : 'transaction',
```

```
    columns : [
               'trandate',
               'tranid',
               'currency',
               'customer.companyname',
               'customer.country',
               'amount'
              ],
    filters : [
                ['mainline', 'is', 'T'],
                'and', ['type','anyof',['VendBill','VendCred']],
                'and', [ ['customer.companyname', 'contains', 'ltd'],
                         'or', ['customer.companyname', 'contains', 'inc']
                       ],
                'and', [ ['subsidiary', 'is', 'HQ'],
                         'or', ['subsidiary', 'anyof', '@HIERARCHY@']
                       ],
                'and', ['trandate', 'notbefore', 'yesterday']
              ]
});
```

Filtrer les expressions contre les objets de filtre

Les expressions de filtre **ne peuvent pas inclure d'** objets de filtrage. C'est très important. Si vous décidez de former vos filtres avec Filter Expression, vous utilisez un tableau de tableaux de chaînes. La syntaxe suivante est **incorrecte** :

```
// WRONG!!!
var f1 = search.createFilter({
            name: 'mainline',
            operator: search.Operator.IS,
            values: 'T'
});

var f2 = search.createFilter({
            name: 'type',
            operator: search.Operator.ANYOF,
            values: ['VendBill','VendCred']
});

// here you will receive an error message
var s = search.create({
    type    : 'transaction',
    filters : [ f1, 'and', f2 ] // f1,f2 are Filter Objects, instead of string arrays
});
```

Au lieu de cela, utilisez le **bon** :

```
// CORRECT!!!
var f1 = ['mainline', search.Operator.IS, 'T'];

var f2 = ['type', search.Operator.ANYOF, ['VendBill','VendCred'] ];

var s = search.create({
    type    : 'transaction',
    filters : [ f1, 'and', f2 ]
});
```

ou, si vous souhaitez conserver l'approche Filtrer les objets, passez un tableau d'objets filtrants et oubliez les opérateurs 'ET', 'OU', 'NON'. Ce sera toujours **ET**

```
// correct, but not useful
var f1 = search.createFilter({
            name: 'mainline',
            operator: search.Operator.IS,
            values: 'T'
});

var f2 = search.createFilter({
            name: 'type',
            operator: search.Operator.ANYOF,
            values: ['VendBill','VendCred']
});


var s = search.create({
    type    : 'transaction',
    filters : [ f1, f2 ] // here you have array of Filter Objects,
                         // filtering only when all of them are TRUE
});
```

Conseils utiles

1. Vous trouverez ici la liste des filtres de recherche disponibles pour les fichiers de date: https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3010842.html
   Vous pouvez les utiliser dans des expressions telles que:

```
['trandate', 'notbefore', 'daysAgo17']
```

2. Voici les opérateurs de recherche: https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3005172.html
   Bien sûr, vous pouvez utiliser **sérum.Operator** enum: https://system.netsuite.com/app/help/helpcenter.nl?fid=section_4345782273.html

3. Voici les types de résumé de recherche: https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3010474.html

4. Vous pouvez utiliser l'opérateur ANYOF uniquement sur certains champs de type (Liste / Enregistrement). Si vous voulez l'utiliser contre des champs de texte libre (comme les noms, les courriels, etc.), la seule solution consiste à créer une expression de filtre imbriquée avec des opérateurs 'OU':

```
[ ['email', 'startswith', 'user1@abcd.com'],
  'or', ['email', 'startswith', 'user2@abcd.com'],
  'or', ['email', 'startswith', 'user3@abcd.com'],
  'or', ['email', 'startswith', 'user4@abcd.com']
]
```

ou vous pouvez écrire un petit script, en faisant cela à votre place:

```
function stringFieldAnyOf(fieldId, listOfValues) {
    var result = [];
    if (listOfValues.length > 0) {
        for (var i = 0; i < listOfValues.length; i++) {
            result.push([fieldId, 'startswith', listOfValues[i]]);
            result.push('or');
        }
        result.pop(); // remove the last 'or'
    }
    return result;
}

// usage: (two more filters added just to illustrate how to combine with other filters)
var custSearch = search.create({
  type: record.Type.CUSTOMER,
  columns: searchColumn,
  filters: [
            ['companyname', 'startswith', 'A'],
            'and', stringFieldAnyOf('email', ['user1@abcd.com', 'user2@abcd.com']),
            'and', ['companyname', 'contains', 'b']
           ]

});
```

5. Toujours pas confiant? Vous cherchez une triche? :)
   Créez une recherche enregistrée dans l'interface utilisateur de Netsuite, prenez l'ID de recherche (par exemple, customsearch1234) et log.debug l'expression du filtre:

```
    var s = search.load('customsearch1234');

    log.debug('filterExpression', JSON.stringify(s.filterExpression));
```

Lire Recherches par script avec des expressions de filtre en ligne:
https://riptutorial.com/fr/netsuite/topic/10732/recherches-par-script-avec-des-expressions-de-filtre

Chapitre 19: RestLet - Récupérer des données (Basic)

Introduction

Cet exemple montre la structure de base d'un script RESTlet destiné à extraire des données d'un système externe. Les RESTlets sont des points de terminaison créés pour permettre la communication avec les systèmes externes.

Examples

Récupérer le nom du client

```
/**
 * requestdata - the data packet expected to be passed in by external system
 * JSON - data format exchange
 * stringify() convert javascript object into a string with JSON.stringify()
 * nlobjError - add in catch block to log exceptions
 */

function GetCustomerData(requestdata)
{
    var jsonString = JSON.stringify(requestdata);
    nlapiLogExecution('DEBUG', 'JSON', jsonString);

    try
    {
        var customer = requestdata.customer;
        nlapiLogExecution('DEBUG', 'customer', customer);
    }
    catch (err)
    {
        var errMessage = err;
        if(err instanceof nlobjError)
        {
            errMessage = errMessage + ' ' + err.getDetails() + ' ' + errMessage;
        }
        nlapiLogExecution('DEBUG', 'Error', errMessage);
    }
}
```

Lire RestLet - Récupérer des données (Basic) en ligne:
https://riptutorial.com/fr/netsuite/topic/9006/restlet---recuperer-des-donnees--basic-

Chapitre 20: RESTlet - Traite les documents externes

Introduction

Lors de la récupération d'un document à partir d'un système externe, nous devons nous assurer que la bonne extension de document est apposée sur le document. L'exemple de code montre comment stocker correctement un document dans le File Cabinet de NetSuite et le joindre à l'enregistrement correspondant.

Examples

RESTlet - stocker et joindre un fichier

```
 /**
 * data - passed in object
 * switch - get file extension if there is one
 * nlapiCreateFile - create file in File Cabinet
 * nlapiAttachRecord - attach file to record
 */

function StoreAttachFile(data)
{
    var record_type = data.recordType
    var record_id = data.recordId;

    if(record_id && record_type == 'vendorbill')
    {
        try
        {
            var file_type = data.fileType;
            var file_extension;

            switch (file_type)
            {
            case "pdf":
                file_extension = "pdf";
                break;
            case "docx":
                file_extension = "doc";
                break;
            case "txt":
                file_extension = "txt";
                break;
            case "JPGIMAGE":
                file_extension = "jpg";
                break;
            case "png":
                file_extension = "png";
                break;
            default:
                // unknown type
```

```
                // there should probably be some error-handling
            }

            var file_name = data.fileName + "." + file_extension;
            var file = data.fileContent;

            var doc = nlapiCreateFile(file_name, file_type, file);
            doc.setFolder(115);//Get Folder ID from: Documents > File > File Cabinet

            var file_id = nlapiSubmitFile(doc);

            nlapiAttachRecord("file", file_id, record_type, record_id);
            nlapiLogExecution('DEBUG', 'after submit', file_id);
        }

        catch (err)
        {
            var errMessage = err;
            if(err instanceof nlobjError)
            {
                errMessage = errMessage + ' ' + err.getDetails();
            }
            nlapiLogExecution('DEBUG', 'Error', errMessage)
        }
    }
    return true;
}
```

Lire RESTlet - Traite les documents externes en ligne:
https://riptutorial.com/fr/netsuite/topic/9021/restlet---traite-les-documents-externes

Chapitre 21: Sourcing

Paramètres

Paramètre	Détails
Liste des sources	Le champ de l'enregistrement de destination qui est lié à l'enregistrement source. Vous devez choisir une liste de sources avant de pouvoir choisir votre champ source.
Source à partir de	Le champ de l'enregistrement source à partir duquel les données seront effectivement extraites. Le champ que vous choisissez doit correspondre au type du champ de destination. Par exemple, si vous effectuez une recherche dans un champ *Numéro de téléphone* , le champ de destination doit également être un champ *Numéro de téléphone* .

Remarques

Impact de la *valeur* du *magasin*

Le paramètre *Valeur de* stockage de la définition de champ personnalisé joue un rôle très important dans le comportement de l'approvisionnement:

- Lorsque la *valeur de la* **banque** est **cochée** , les données proviennent du champ *uniquement lors de la création initiale* de l'enregistrement. Après cela, NetSuite rompt le lien d'approvisionnement entre les champs et ils deviennent deux champs indépendants. Cela vous permet d'exploiter efficacement le sourcing comme mécanisme de définition de la valeur initiale ou par défaut de votre champ personnalisé.
- Lorsque l' *option Store Value* est **décochée** , les données sont générées dynamiquement dans le champ à **chaque chargement de l'enregistrement** . Toutes les modifications qu'un utilisateur ou un script peut apporter au champ **ne** sont **jamais enregistrées** . Si vous ne *cochez* pas la *valeur Store* , il est conseillé de rendre votre champ en lecture seule.

Limites de l'approvisionnement

- Le sourcing ne peut pas être appliqué aux *champs NetSuite natifs* . Si vous avez besoin d'un champ natif en tant que champ de destination, vous devrez créer un flux de travail ou écrire un script pour effectuer le sourcing de données.
- Le sourcing ne peut pas être appliqué aux *champs de colonne de sous-liste* . Si vous avez besoin d'une colonne de sous-liste comme champ de destination, vous devrez créer un flux de travail ou rédiger un script pour effectuer la sélection des données.

Examples

Tirer des données dans un champ personnalisé sur le champ modifié

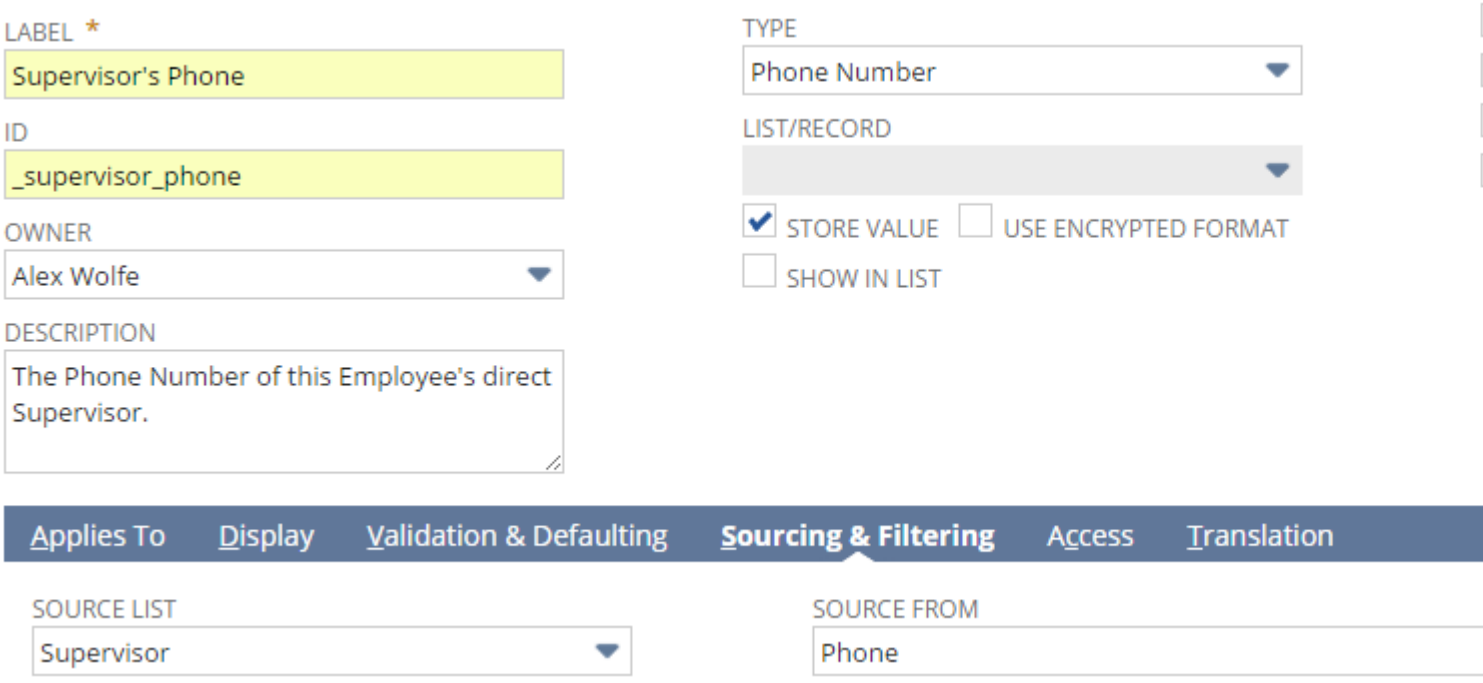
```
// If you find yourself doing something like this...
function fieldChanged(type, name, index) {
    if (name == 'salesrep') {
        var salesRepId = nlapiGetFieldValue('salesrep');
        var salesRepEmail = nlapiLookupField('employee', salesRepId, 'email');
        nlapiSetFieldValue('custbody_salesrep_email', salesRepEmail);
    }
}
// Stop! and consider using Sourcing for your custom field instead of code
```

Définition du sourcing

Bien que n'étant pas strictement une rubrique SuiteScript, le *sourcing* est une fonctionnalité incroyablement puissante de NetSuite et constitue un outil important dans la barre d'outils pour tout développeur SuiteScript. Le sourcing nous permet d' *extraire des données dans un enregistrement depuis n'importe lequel de ses enregistrements* , sans écrire de code ni créer un workflow pour le faire.

Moins de code est toujours un code plus maintenable.

Le sourcing est défini dans l'onglet *Sourcing & Filtering* d'une définition de champ personnalisé.



Lire Sourcing en ligne: https://riptutorial.com/fr/netsuite/topic/7034/sourcing

Chapitre 22: SS2.0 Suitelet Bonjour tout le monde

Examples

Basic Hello World Suitelet - Réponse en texte brut

```
/**
*@NApiVersion 2.x
*@NScriptType Suitelet
*/

define([],function() { // NetSuite's AMD pattern
    function onRequest_entry(context) { // Suitelet entry function receives a context obj
        context.response.write('Hello World'); // Write a response using the context obj
    }
    return {
        onRequest: onRequest_entry // Function assigned to entry point
    };
});
```

Lire SS2.0 Suitelet Bonjour tout le monde en ligne:
https://riptutorial.com/fr/netsuite/topic/6723/ss2-0-suitelet-bonjour-tout-le-monde

Chapitre 23: SuiteScript - Traiter des données à partir d'Excel

Introduction

Parfois, les résultats de recherche renvoyés dans une mise à jour groupée ne sont pas les mêmes que ceux d'une recherche standard. Cela est dû à certaines limitations dans une recherche de mise à jour groupée. Un exemple de ceci est les entrées Rev Rec Journal. Par conséquent, la solution de contournement consistait à obtenir les données de la recherche enregistrée standard et à utiliser un script pour lire les données Excel et la mise à jour, plutôt que d'utiliser la fonctionnalité de mise à jour en masse.

Examples

Mettre à jour les dates d'enregistrement et la règle

```
/**
 * Save the results from the saved search as .csv and store in file cabinet
 * Get the file's internal id when loading the file
 * Use \n to process each row
 * Get the internal id and whatever columns that need updating
 * Create a filtered search and pass the internal id
 * If the passed in internal id finds a record match, then update the rev rec dates and rule
 */

function ProcessSearchData()
{
    var loaded_file = nlapiLoadFile(4954);//loads from file cabinet
    var loaded_string = loaded_file.getValue();
    var lines = loaded_string.split('\n');//split on newlines
    nlapiLogExecution('DEBUG', 'lines', lines);
    var values;
    for (var i = 1; i < lines.length; i++)
    {
        nlapiLogExecution('DEBUG', 'count', i);
        values = lines[i].split(',');//split by comma
        var internal_id = values[0];//first column value
        nlapiLogExecution('DEBUG', 'internal_id', internal_id);
        var start_date = values[1];
        var end_date = values[2];

        if(internal_id)
        {
            UpdateDates(internal_id, start_date, end_date)
            nlapiLogExecution('DEBUG', '"""REV REC PLANs UPDATED"""');
        }
    }
    return true;
}
```

```
function UpdateDates(internal_id, start_date, end_date)
{
    var filters = new Array();
    filters[0] = new nlobjSearchFilter('internalid', null, 'is', internal_id);

    var columns = [];
    columns[0] = new nlobjSearchColumn('internalid');
    columns[1] = new nlobjSearchColumn('revrecstartdate');
    columns[2] = new nlobjSearchColumn('revrecenddate');

    var rev_rec_plan = nlapiSearchRecord('revenueplan', null, filters, columns);
    if(rev_rec_plan)
    {
        for (var i = 0; rev_rec_plan != null && i < rev_rec_plan.length; i++)
        {
            var record = nlapiLoadRecord('revenueplan', rev_rec_plan[0].getValue(columns[0]));
            var id = record.getId();
            record.setFieldValue('revrecstartdate', start_date);
            record.setFieldValue('revrecenddate', end_date);
            record.setFieldValue('revenuerecognitionrule', 2)//Exact days based on Arrangement
dates
            nlapiSubmitRecord(record);
        }
    }
    return internal_id;
}
```

Lire SuiteScript - Traiter des données à partir d'Excel en ligne:
https://riptutorial.com/fr/netsuite/topic/9034/suitescript---traiter-des-donnees-a-partir-d-excel

Chapitre 24: Travailler avec des sous-listes

Introduction

Les enregistrements NetSuite sont divisés en champs de corps et en sous-listes. Il existe quatre types de sous-listes: Static, Editor, Inline Editor et List.

Nous sommes en mesure d'ajouter, d'insérer, de modifier et de supprimer des éléments de ligne à l'aide d'API Sublist.

Pour obtenir une référence précise sur les sous-listes qui prennent en charge SuiteScript, consultez la page d'aide de NetSuite intitulée "Sublistes scriptables".

Remarques

Indices Sublistes

Chaque élément de ligne d'une sous-liste a un index que nous pouvons utiliser pour le référencer.

Dans SuiteScript 1.0, ces index sont basés sur `1` , de sorte que le premier élément de ligne a l'index `1` , le second l'index `2` , etc.

Dans SuiteScript 2.0, ces index sont basés sur `0` , de sorte que le premier élément de ligne a l'index `0` , le second l'index `1` , etc. Cela correspond bien sûr à l'indexation des tableaux dans la plupart des langues, y compris JavaScript.

Mode standard vs dynamique

L'API que nous utilisons pour interagir avec une sous-liste dépend de si nous travaillons avec l'enregistrement en mode Standard ou Dynamique.

Les API en mode standard nous permettent simplement de fournir l'index de la ligne avec laquelle nous voulons travailler en tant que paramètre de la fonction appropriée.

Les API en mode dynamique suivent un modèle:

1. Sélectionnez la ligne avec laquelle nous voulons travailler
2. Modifier la ligne sélectionnée comme vous le souhaitez
3. Valider les modifications sur la ligne

En mode dynamique, si nous ne commettons pas les modifications sur *chaque* ligne que nous modifions, ces modifications ne seront pas reflétées lors de l'enregistrement de l'enregistrement.

Limites

Pour travailler avec des données de sous-liste via SuiteScript, nous devons avoir une référence en mémoire à l'enregistrement. Cela signifie que l'enregistrement doit être extrait du contexte du script ou que nous devons charger l'enregistrement depuis la base de données.

Nous *ne pouvons pas* travailler avec les sous-listes via les fonctionnalités de recherche ou de soumission .

Les sous-listes *statiques* ne supportent pas du tout SuiteScript.

Les références:

- NetSuite Help: "Qu'est-ce qu'un Sublist?"
- NetSuite Help: "Types Sublist"
- NetSuite Help: "Sublistes scriptables"
- NetSuite Help: "Utilisation des éléments de campagne de sous-liste"
- NetSuite Help: "API de sous-liste"
- NetSuite Help: "Utilisation des enregistrements en mode dynamique"

Examples

[1.0] Combien de lignes sur une sous-liste?

```
// How many Items does a Sales Order have...

// ... if we're in the context of a Sales Order record
var itemCount = nlapiGetLineItemCount("item");

// ... or if we've loaded the Sales Order
var order = nlapiLoadRecord("salesorder", 123);
var itemCount = order.getLineItemCount("item");
```

[1.0] Sublistes en mode standard

```
// Working with Sublists in Standard mode ...

// ... if the record is in context:

// Add item 456 with quantity 10 at the end of the item sublist
var nextIndex = nlapiGetLineItemCount("item") + 1;
nlapiSetLineItemValue("item", "item", nextIndex, 456);
nlapiSetLineItemValue("item", "quantity", nextIndex, 10);

// Inserting item 234 with quantity 3 at the beginning of a sublist
nlapiInsertLineItem("item", 1);
nlapiSetLineItemValue("item", "item", 1, 234);
nlapiSetLineItemValue("item", "quantity", 1, 3);

// Insert item 777 with quantity 2 before the end of the last item
var itemCount = nlapiGetLineItemCount("item");
nlapiInsertLineItem("item", itemCount);
nlapiSetLineItemValue("item", "item", itemCount, 777);
```

```
nlapiSetLineItemValue("item", "quantity", itemCount, 2);

// Remove the first line item
nlapiRemoveLineItem("item", 1);

// Remove the last line item
nlapiRemoveLineItem("item", nlapiGetLineItemCount("item"));

// ... or if we have a reference to the record (rec):

// Add item 456 with quantity 10 at the end of the item sublist
var nextIndex = rec.getLineItemCount("item") + 1;
rec.setLineItemValue("item", "item", nextIndex, 456);
rec.setLineItemValue("item", "quantity", nextIndex, 10);

// Insert item 777 with quantity 3 at the beginning of the sublist
rec.insertLineItem("item", 1);
rec.setLineItemValue("item", "item", 1, 777);
rec.setLineItemValue("item", "quantity", 1, 3);

// Remove the first line
rec.removeLineItem("item", 1);

// Remove the last line
rec.removeLineItem("item", rec.getLineItemCount("item"));
```

[1.0] Sublistes en mode dynamique

```
// Adding a line item to the end of a sublist in Dynamic Mode...

// ... if the record is in context:
nlapiSelectNewLineItem("item");
nlapiSetCurrentLineItemValue("item", "item", 456);
nlapiSetCurrentLineItemValue("item", "quantity", 10);
nlapiCommitLineItem("item");

// ... or if we have a reference to the record (rec):
rec.selectNewLineItem("item");
rec.setCurrentLineItemValue("item", "item", 456);
rec.setCurrentLineItemValue("item", "quantity", 10);
rec.commitLineItem("item");
```

[1.0] Trouver un élément de ligne

```
// Which line has item 456 on it...

// ... if we're in the context of a record
var index = nlapiFindLineItemValue("item", "item", 456);
if (index > -1) {
    // we found it...
} else {
    // item 456 is not in the list
}

// ... or if we have a reference to the record (rec)
var index = rec.findLineItemValue("item", "item", 456);
if (index > -1) {
```

```
    // we found it on line "index"...
} else {
    // item 456 is not in the list
}
```

[2.0] Combien de lignes sur une sous-liste?

```
// How many lines in a sublist in SuiteScript 2.0...

require(["N/record"], function (r) {
    var rec = r.load({
        "type": r.Type.SALES_ORDER,
        "id": 123
    });

    // How many lines are on the Items sublist?
    var itemCount = rec.getLineCount({"sublistId": "item"});
});
```

[2.0] Sublistes en mode standard

```
// Working with a sublist in Standard Mode in SuiteScript 2.0...

require(["N/record"], function (r) {
    var rec = r.create({
        "type": r.Type.SALES_ORDER,
        "isDynamic": false
    });

    // Set relevant body fields ...

    // Add line item 456 with quantity 10 at the beginning of the Items sublist
    rec.setSublistValue({"sublistId": "item", "fieldId": "item", "value": 456, "line": 0});
    rec.setSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 10, "line": 0});

    // Insert line item 238 with quantity 5 at the beginning of the Items sublist
    rec.insertLine({"sublistId": "item", "line": 0});
    rec.setSublistValue({"sublistId": "item", "fieldId": "item", "value": 238, "line": 0});
    rec.setSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 5, "line": 0});

    // Insert line item 777 with quantity 3 before the last line of the Items sublist
    var lastIndex = rec.getLineCount({"sublistId": "item"}) - 1; // 2.0 sublists have 0-based
index
    rec.insertLine({"sublistId": "item", "line": lastIndex}); // The last line will now
actually be at lastIndex + 1
    rec.setSublistValue({"sublistId": "item", "fieldId": "item", "value": 777, "line":
lastIndex});
    rec.setSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 3, "line":
lastIndex});

    // Remove the first line
    rec.removeLine({"sublistId": "item", "line": 0});

    // Remove the last line
    rec.removeLine({"sublistId": "item", "line": rec.getLineCount({"sublistId": "item"}) -
1});
```

```
    rec.save();
});
```

[2.0] Sublistes en mode dynamique

```
// Working with Sublists in Dynamic Mode in SuiteScript 2.0...

require(["N/record"], function (r) {
    var rec = r.create({
        "type": r.Type.SALES_ORDER,
        "isDynamic": true
    });

    // Set relevant body fields ...

    // Add line item 456 with quantity 10 at the end of the Items sublist
    var itemCount = rec.selectNewLine({"sublistId": "item"});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "item", "value": 456});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 10});
    rec.commitLine({"sublistId": "item"});

    // Insert line item 378 with quantity 2 at the beginning of the Items sublist
    rec.insertLine({"sublistId": "item", "line": 0});
    rec.selectLine({"sublistId": "item", "line": 0});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "item", "value": 378});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 2});
    rec.commitLine({"sublistId": "item"});

    // Insert line item 777 with quantity 3 before the last line of the Items sublist
    var lastIndex = rec.getLineCount({"sublistId": "item"}) - 1; // 2.0 sublists have 0-based
index
    rec.insertLine({"sublistId": "item", "line": lastIndex}); // The last line will now
actually be at lastIndex + 1
    rec.selectLine({"sublistId": "item", "line": lastIndex});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "item", "value": 777});
    rec.setCurrentSublistValue({"sublistId": "item", "fieldId": "quantity", "value": 3});
    rec.commitLine({"sublistId": "item"});

    // Remove the first line
    rec.removeLine({"sublistId": "item", "line": 0});

    // Remove the last line
    rec.removeLine({"sublistId": "item", "line": rec.getLineCount({"sublistId": "item"}) -
1});

    rec.save();
});
```

[2.0] Trouver un élément de campagne

```
// Finding a specific line item in SuiteScript 2.0...

require(["N/record"], function (r) {
    var rec = r.load({
        "type": r.Type.SALES_ORDER,
        "id": 123
    });
```

```
    // Find the line that contains item 777
    var index = rec.findSublistLineWithValue({"sublistId": "item", "fieldId": "item", "value":
777});

    // find returns -1 if the item isn't found
    if (index > -1) {
        // we found it on line "index"
    } else {
        // item 777 is not in the list
    }
});
```

Lire Travailler avec des sous-listes en ligne: https://riptutorial.com/fr/netsuite/topic/9098/travailler-avec-des-sous-listes

Chapitre 25: Utilisation du navigateur NetSuite Records

Examples

Utilisation du navigateur NetSuite Records

Le *navigateur d'enregistrements* définit le schéma pour tous les types d'enregistrement scriptables. c'est un outil de référence extrêmement critique pour chaque développeur SuiteScript. Lorsque vous avez besoin de savoir comment référencer un champ particulier sur un type d'enregistrement spécifique dans votre script, le *navigateur d'enregistrements* est votre guide.

Lien direct

Autre schéma

Vous pouvez également remarquer des onglets en haut du *navigateur d'enregistrements* pour le *navigateur* de *schémas* et le *navigateur de connexions* . Celles-ci sont très similaires au *navigateur d'enregistrements* , mais pour différentes API NetSuite.

Le *navigateur de schéma* fournit le schéma de l'API de services Web SOAP, tandis que le *navigateur de connexion* fournit le schéma du connecteur ODBC.

Navigation dans le navigateur de notices

Vous parcourez d'abord le *navigateur* d'enregistrements par type d'enregistrement, à savoir "commande client", "facture", "employé". Il n'y a pas de fonction de recherche dans le *navigateur d'enregistrements* , de sorte que toute la navigation est effectuée manuellement. Les types d'enregistrement sont classés par ordre alphabétique. Vous devez donc d'abord cliquer sur la première lettre du type d'enregistrement qui vous intéresse, puis sélectionner le type d'enregistrement lui-même à gauche.

Par exemple, si vous voulez voir le schéma du type d'enregistrement *Subisidary* , vous devez d'abord cliquer sur *S* en haut, puis sur *Subsidiary* sur la gauche.

Lecture du schéma

Chaque schéma vous fournit une quantité considérable d'informations sur chaque type d'enregistrement. Il est important de savoir comment décomposer toutes ces informations.

En haut du schéma se trouve le nom du type d'enregistrement suivi de l'ID interne du type d'enregistrement; cet ID interne est la référence de programmation pour le type d'enregistrement. Le schéma est ensuite divisé en plusieurs sections:

- *Champs* : la section *Champs* répertorie les détails de tous les champs de *corps* de l'enregistrement. Les champs décrits ici peuvent être utilisés lorsque vous travaillez avec l'enregistrement actuellement en contexte ou avec une référence directe à un objet d'enregistrement.
- *Sublistes* : la section *Sublistes* affiche toutes les sous-listes de l'enregistrement et toutes les colonnes scriptables de chaque sous-liste. Les champs de cette section s'appliquent à nouveau lorsque vous travaillez avec l'enregistrement actuellement en contexte ou avec une référence directe à un objet d'enregistrement.
- *Onglets* : la section *Onglets* décrit tous les sous-onglets natifs du type d'enregistrement.
- *Jointures de recherche* : la section *Rechercher des jointures* décrit tous les enregistrements associés à travers lesquels vous pouvez créer des jointures dans vos recherches de ce type d'enregistrement.
- *Filtres de recherche* : la section *Filtres de recherche* décrit tous les champs disponibles en tant que filtre de recherche pour ce type d'enregistrement. L'ID interne lorsque vous utilisez un champ spécifique en tant que filtre de recherche *ne correspond pas toujours à* son ID interne en tant que zone de corps.
- *Colonnes de recherche* : la section *Colonnes de recherche* décrit tous les champs disponibles en tant que colonne de recherche pour ce type d'enregistrement. L'ID interne lorsque vous utilisez un champ spécifique en tant que colonne de recherche *ne correspond pas toujours à* son ID interne en tant que zone de corps.
- *Types de transformation* : la section *Types de transformation* décrit tous les types d'enregistrement que celui-ci peut être transformé en utilisant l'API de transformation d'enregistrement.

Trouver un champ

Comme indiqué précédemment, aucune fonctionnalité de recherche n’est intégrée au *navigateur d’enregistrements* . Une fois que vous avez accédé au type d'enregistrement approprié, si vous ne connaissez pas déjà l'ID interne d'un champ particulier, la façon la plus simple de le trouver consiste à utiliser la fonction de recherche de votre navigateur (généralement `CTRL+F`). dans l'interface utilisateur.

Champs obligatoires

La colonne *Obligatoire* du schéma indique si ce champ est requis pour enregistrer l'enregistrement. Si cette colonne indique `true` , vous devrez alors fournir une valeur pour ce champ lorsque vous enregistrez un enregistrement de ce type.

nlapiSubmitField et édition en ligne

La colonne `nlapiSubmitField` est un élément essentiel à comprendre. Cette colonne indique si le champ est disponible pour l'édition en ligne. Si `nlapiSubmitField` est `true` , le champ peut être modifié en ligne. Cela a un impact considérable sur la gestion de ce champ lors de la tentative d'utilisation des fonctions `nlapiSubmitField` ou `record.submitFields` dans vos scripts.

Lorsque cette colonne est `true` , vous pouvez utiliser en toute sécurité les API de champs de soumission pour mettre à jour ce champ en ligne. Quand il est `false` , *vous pouvez toujours utiliser*

ces fonctions pour mettre à jour le champ , mais ce qui se passe réellement en coulisse change de manière significative.

Lorsque `nlapiSubmitField` est `nlapiSubmitField` `false` pour un champ particulier et que vous utilisez l'une des API de champs de soumission, le moteur de script en arrière-plan effectue un chargement complet de l'enregistrement, met à jour le champ et renvoie la modification à la base de données. Le résultat final est le même, mais comme l'intégralité de l'enregistrement est chargée et enregistrée, votre script utilisera beaucoup plus de gouvernance que prévu et son exécution prendra plus de temps.

Pour plus d'informations à ce sujet, consultez la page d'aide intitulée "Conséquences de l'utilisation de nlapiSubmitField sur des champs modifiables non intégrés".

Lire Utilisation du navigateur NetSuite Records en ligne:
https://riptutorial.com/fr/netsuite/topic/7756/utilisation-du-navigateur-netsuite-records

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec netsuite	4444, Community, erictgrubaugh, Kirk Lampert
2	Chargement d'un enregistrement	Adolfo Garza, Eidolon108, LittleZul, michoel, VicDid, YNK
3	Comprendre les recherches de transactions	erictgrubaugh
4	Créer un enregistrement	Deepa N, michoel, YNK, Zain Shaikh
5	Demande de customField, customFieldList & customSearchJoin avec PHP API Advanced Search	Hayden Thring
6	Edition en ligne avec SuiteScript	erictgrubaugh
7	Enregistrements de déploiement de script et de script	erictgrubaugh
8	Evénement utilisateur: Evénement avant chargement	erictgrubaugh
9	Evénement utilisateur: événements avant et après la soumission	erictgrubaugh
10	Exécuter une recherche	Adolfo Garza
11	Exploitation des colonnes de	MetaEd

	formules dans les recherches enregistrées	
12	Gouvernance	erictgrubaugh
13	Mass Delete	MG2016
14	Présentation du type de script	erictgrubaugh
15	Recherche de données à partir d'enregistrements associés	erictgrubaugh
16	Recherches avec un grand nombre de résultats	Slavi Slavov
17	Recherches par script avec des expressions de filtre	Slavi Slavov
18	RestLet - Récupérer des données (Basic)	MG2016
19	RESTlet - Traite les documents externes	MG2016
20	Sourcing	erictgrubaugh
21	SS2.0 Suitelet Bonjour tout le monde	LittleZul
22	SuiteScript - Traiter des données à partir d'Excel	MG2016
23	Travailler avec des sous-listes	erictgrubaugh
24	Utilisation du navigateur NetSuite Records	erictgrubaugh