EBook Gratis

APRENDIZAJE netsuite

Free unaffiliated eBook created from
Stack Overflow contributors.

#netsuite

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con netsuite

Observaciones

NetSuite es una plataforma de gestión de ERP, CRM, comercio electrónico y servicios profesionales basada en la nube. Es usado por más de 30,000 compañías para dirigir todo su negocio.

NetSuite es totalmente personalizable por los administradores y desarrolladores, incluso a través de una API basada en JavaScript llamada SuiteScript. Los desarrolladores pueden escribir scripts que son activados por varios eventos en todo el sistema NetSuite para automatizar los procesos de negocios.

Dónde obtener ayuda

1. Únase a la comunidad Slack de NetSuite Professionals , donde tiene acceso instantáneo a más de 200 profesionales de NetSuite en todo el mundo.
2. Utilice el navegador de registros de NetSuite para el esquema de todos los tipos de registros
3. Guía de referencia de JavaScript de Mozilla Developer Network

Versiones

Versión	Fecha de lanzamiento
2016.2	2016-09-20

Examples

Configuración de Eclipse SuiteCloud IDE

1. Descarga e instala la última versión de Eclipse IDE.
    - Instala Eclipse de una de estas dos maneras:
        1. Instalador Eclipse
        2. Descarga el zip de tu paquete favorito.
    - Si aún no tiene un paquete Eclipse preferido, se recomienda *Eclipse para desarrolladores de JavaScript.*
2. Instalar el complemento IDE de SuiteCloud
    1. Una vez completada la instalación, inicie Eclipse
    2. Vaya a *Ayuda* > *Instalar nuevo software ...*
    3. Haga clic en *Agregar ...* para agregar un nuevo sitio de actualización
        - **Nombre** : SuiteCloud IDE
        - **Ubicación** : http://system.netsuite.com/download/ide/update_e4
            - **Nota:** la ubicación depende de la versión de NetSuite en la que se

encuentre actualmente.
        - **Por ejemplo** : si actualmente está en la versión 2017.1, debe usar esta url en su lugar: http://system.netsuite.com/download/ide/update_17_1
    4. Seleccione el sitio "SuiteCloud IDE" en el menú desplegable *Trabajar con*
    5. Continúe con el asistente de instalación
    6. Reinicie Eclipse cuando se le solicite
3. Configurar el complemento IDE de SuiteCloud
    1. Cuando se reinicia Eclipse, se le solicitará que configure el complemento de SuiteCloud con una contraseña maestra y una cuenta de NetSuite predeterminada
    2. Después de completar este asistente de configuración, vaya a *Preferencias* > *NetSuite*
        - Aquí encontrarás todas las preferencias de SuiteCloud IDE.
    3. [Opcional] Si su uso principal para Eclipse es el desarrollo de NetSuite, vaya a *Preferencias* > *General* > *Perspectivas* y haga que la Perspectiva de "NetSuite" sea su valor predeterminado
4. Crea un nuevo proyecto de NetSuite.
    1. Haga clic con el botón derecho en la ventana de *NS Explorer* y seleccione *Nuevo* > *Proyecto NetSuite*
    2. Siga el asistente para la configuración del proyecto de su elección. Los tipos de proyectos son los siguientes:
        1. *Personalización de la cuenta* : un proyecto que aprovecha el *marco de desarrollo de SuiteCloud* para crear objetos, registros y scripts personalizados para personalizar una cuenta de NetSuite.
        2. *SuiteScript* : Un proyecto utilizado exclusivamente para escribir guiones.
        3. *Aplicación SSP* : una aplicación de páginas de servidor SuiteScript, que se utiliza normalmente junto con SiteBuilder o SuiteCommerce para aplicaciones de comercio electrónico respaldadas por NetSuite.

Hola, World 1.0 Client Script

1. Cree el archivo fuente para su nuevo script de cliente

    1. Crea un nuevo archivo JavaScript usando tu editor favorito o IDE

    2. Agregue el siguiente código fuente a su archivo (fuente original aquí)

```
/**
 * 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. Guarda el archivo como `hello-world.js` donde quieras

2. Utilice el archivo fuente que acabamos de crear para crear un nuevo registro de *Script* en NetSuite
    1. En su cuenta de NetSuite, navegue a *Personalización* > *Secuencias de comandos* >

Secuencias de comandos > *Nuevo*
2. Cuando se le solicite, seleccione `hello-world.js` como el *archivo de script*
3. Haga clic en *Crear registro de script*
4. Cuando se le solicite, seleccione *Script de cliente* como el tipo de script
5. Nombra tu registro de Script *Hello World*
6. Asigne la función denominada `pageInit` en nuestro archivo de origen al evento de script de *inicio de página* ingresando `pageInit` en el campo *Función de inicio de página*
7. Guarda tu nuevo registro de Script

3. Implemente su nuevo script en el registro de empleado
    1. En su registro de Script recién creado, haga clic en *Implementar Script*
    2. En el campo *Aplica a* , seleccione *Empleado*
    3. Asegúrese de que el campo *Estado* esté configurado en *Prueba*
    4. Haga clic en *guardar*
4. ¡Vea su guión en acción!
    1. Abra la consola de desarrollador / JavaScript de su navegador (normalmente F12 en la mayoría de los navegadores)
    2. Cree un nuevo empleado navegando a *Listas* > *Empleados* > *Empleados* > *Nuevo*
    3. Observe su mensaje "Hola, Mundo" en la consola del navegador.

Hola, World 2.0 Client Script

1. Cree el archivo fuente para su nuevo script de cliente

    1. Crea un nuevo archivo JavaScript usando tu editor favorito o IDE

    2. Agregue el siguiente código fuente a su archivo (fuente original aquí)

```
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. Guarda el archivo como `hello-world2.js` donde quieras

2. Utilice el archivo fuente que acabamos de crear para crear un nuevo registro de *Script* en NetSuite
    1. En su cuenta de NetSuite, navegue a *Personalización* > *Secuencias de comandos* > *Secuencias de comandos* > *Nuevo*
    2. Cuando se le solicite, seleccione `hello-world2.js` como el *archivo de script*

    3. Haga clic en *Crear registro de script*
    4. Nombra tu registro de Script *Hello World*
    5. Guarda tu nuevo registro de Script
3. Implemente su nuevo script en el registro de empleado
    1. En su registro de Script recién creado, haga clic en *Implementar Script*
    2. En el campo *Aplica a* , seleccione *Empleado*
    3. Asegúrese de que el campo *Estado* esté configurado en *Prueba*
    4. Haga clic en *guardar*
4. ¡Vea su guión en acción!
    1. Abra la consola de desarrollador / JavaScript de su navegador (normalmente F12 en la mayoría de los navegadores)
    2. Cree un nuevo empleado navegando a *Listas* > *Empleados* > *Empleados* > *Nuevo*
    3. Observe su mensaje "Hola, Mundo" en la consola del navegador.

Lea Empezando con netsuite en línea: https://riptutorial.com/es/netsuite/topic/3828/empezando-con-netsuite

Capítulo 2: Abastecimiento

Parámetros

Parámetro	Detalles
Lista de fuentes	El campo en el registro de destino que enlaza con el registro de origen. Debe elegir una lista de fuentes antes de poder elegir su campo de fuente.
Fuente de	El campo en el registro de origen del cual se extraerán los datos. El campo que elija debe coincidir con el tipo del campo de destino. Por ejemplo, si está obteniendo de un campo *Número de teléfono* , el campo de destino también debe ser un campo *Número de teléfono* .

Observaciones

Impacto del *valor* de la *tienda*

La configuración del *valor de la tienda* en la definición del campo personalizado juega un papel muy importante en el comportamiento de Sourcing:

- Cuando se **comprueba** *Tienda Básica,* los datos se obtiene en el campo *sólo tras la creación inicial* del registro. Después de eso, NetSuite rompe el enlace de origen entre los campos y se convierten en dos campos independientes. Esto le permite efectivamente aprovechar el Sourcing como un mecanismo para establecer el valor inicial o predeterminado de su campo personalizado.
- Cuando el *valor de la tienda* **no** está **marcado** , los datos se obtienen dinámicamente en el campo **cada vez que se carga el registro** . Cualquier cambio que un usuario o script pueda hacer en el campo **nunca se guarda** . Si deja *Store Value* sin marcar, es una buena idea hacer que su campo sea de solo lectura.

Limitaciones de Sourcing

- El aprovisionamiento no se puede aplicar a *los campos nativos de NetSuite* . Si necesita un campo nativo como campo de destino, deberá crear un flujo de trabajo o escribir un script para realizar el suministro de datos.
- El aprovisionamiento no se puede aplicar a los *campos de columna sublista* . Si necesita una columna sublista como campo de destino, deberá crear un flujo de trabajo o escribir un script para realizar el suministro de datos.

Examples

Recopilación de datos en un campo personalizado en Campo cambiado

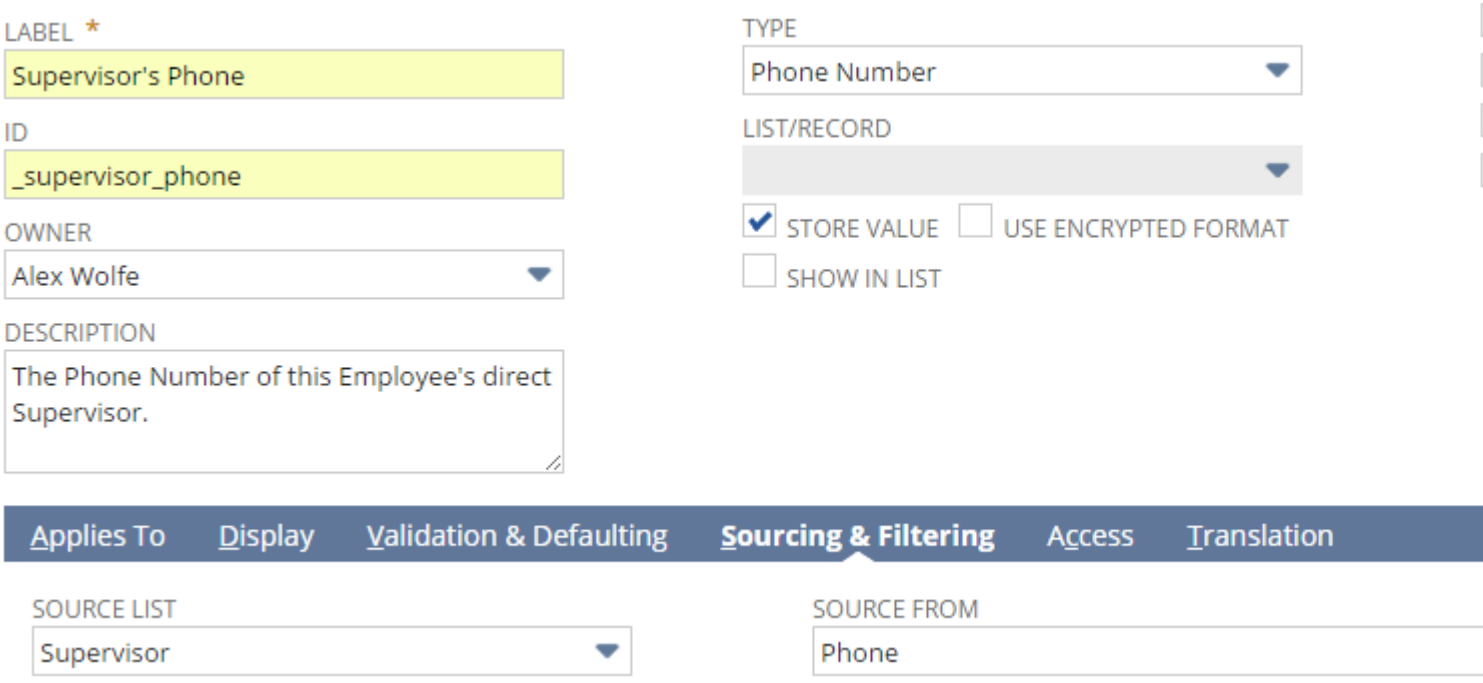
```
// If you find yourself doing something like this...
function fieldChanged(type, name, index) {
    if (name == 'salesrep') {
        var salesRepId = nlapiGetFieldValue('salesrep');
        var salesRepEmail = nlapiLookupField('employee', salesRepId, 'email');
        nlapiSetFieldValue('custbody_salesrep_email',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

Definición de Sourcing

Aunque no es estrictamente un tema de SuiteScript, *Sourcing* es una característica increíblemente poderosa de NetSuite, y es una herramienta importante en el cinturón de herramientas para cualquier desarrollador de SuiteScript. El abastecimiento nos permite *extraer datos en un registro de cualquiera de sus registros relacionados* , sin escribir ningún código ni crear un flujo de trabajo para hacerlo.

Menos código es siempre un código más mantenible.

La fuente se define en la pestaña *Fuente y filtrado* de una definición de campo personalizado.



Lea Abastecimiento en línea: https://riptutorial.com/es/netsuite/topic/7034/abastecimiento

Capítulo 3: Buscar datos de registros relacionados

Introducción

Al procesar un registro dado, a menudo necesitará recuperar datos de uno de sus registros relacionados. Por ejemplo, cuando se trabaja con un Pedido de venta determinado, es posible que deba recuperar datos del Representante de ventas relacionado. En la terminología de SuiteScript, esto se denomina **búsqueda** .

La funcionalidad de búsqueda la proporciona la función global `nlapiLookupField` en SuiteScript 1.0 y el método `lookupFields` del módulo `N/search` en SuiteScript 2.0.

Sintaxis

- nlapiLookupField (recordType, recordId, columnas);

Parámetros

Parámetro	Detalles
recordType	`String` : la ID interna del tipo de registro que se está buscando (p. `salesorder` , `salesorder` , `employee`)
recordId	`String` o `Number` : la ID interna del registro que se está buscando
columnas	`String` o `String[]` : la lista de campos para recuperar del registro. Las ID de campo se pueden consultar en la sección "Buscar columnas" del Explorador de registros . Los campos unidos se pueden recuperar utilizando la sintaxis de puntos (por ejemplo, `salesrep.email`)

Observaciones

Actuación

Una búsqueda es solo una abreviatura para realizar una búsqueda que filtra la identificación interna de un solo registro para el resultado. Bajo el capó, las búsquedas están realizando una búsqueda, por lo que el rendimiento será similar al de una búsqueda que devuelve un registro único.

Esto también significa que una búsqueda se realizará más rápido que cargar el registro para

recuperar la misma información.

Limitaciones

Las búsquedas solo se pueden utilizar para recuperar datos de campo del cuerpo. No puede recuperar datos de las listas secundarias de un registro relacionado mediante una búsqueda. Si necesita datos de la lista secundaria, deberá realizar una búsqueda o cargar el registro relacionado.

Examples

[1.0] Buscar un solo campo

```
/**
 * 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] Buscar campos múltiples

```
/**
 * 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] Búsqueda de campos unidos

```
/**
 * 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] Buscar un solo campo

```
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] Buscar campos múltiples

```
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] Búsqueda de campos unidos

```
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"]);
    })();
});
```

Lea Buscar datos de registros relacionados en línea:
https://riptutorial.com/es/netsuite/topic/9068/buscar-datos-de-registros-relacionados

Capítulo 4: Búsquedas con gran cantidad de resultados.

Introducción

Suitescript 2.0 proporciona 4 métodos para manejar los resultados de búsqueda.

Tienen diferentes sintaxis, limitaciones y gobernanza, y son apropiados para diferentes situaciones. Nos centraremos aquí en cómo acceder a **TODOS** los resultados de búsqueda, utilizando cada uno de estos métodos.

Examples

Usando el método Search.ResultSet.each

Este es el método más corto, fácil y usado. Desafortunadamente, tiene una limitación importante: no se puede utilizar en búsquedas con más de 4000 resultados (filas).

```
// 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;

});
```

Usando el método ResultSet.getRange

Para utilizar getRange para manejar la gran cantidad de resultados, tendremos que considerar lo siguiente:

1. getRange tiene 2 parámetros: **inicio** y **final** . Siempre positivo, siempre (inicio <fin)
2. **Inicio** es el índice inclusivo del primer resultado a devolver.
3. **End** es el índice exclusivo del último resultado a devolver.
4. Si hay menos resultados disponibles que los solicitados, entonces la matriz contendrá menos de las entradas de inicio final. Por ejemplo, si solo hay 25 resultados de búsqueda, entonces getRange (20, 30) devolverá una matriz de 5 objetos search.Result.
5. Aunque la oración de ayuda anterior no lo dice directamente, tanto el **inicio** como el **final** podrían estar fuera del rango de resultados disponibles. En el mismo ejemplo, si solo hay 25 resultados de búsqueda, getRange (100, 200) devolverá una matriz vacía []
6. Máximo 1000 filas a la vez. (final - inicio) <= 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
        });
    }
}
```

Permite calcular la gobernanza. Tenemos 1 + recuento / 1000 llamadas getRange que toman 10 unidades cada una, así que:

G = (1 + cuenta / 1000) * 10

Ejemplo: 9500 filas tomarán 100 unidades.

Usando el método Search.PagedData.fetch

PagedData es un objeto, devuelto por el método Search.runPaged (opciones). Funciona exactamente como lo hacen las búsquedas de interfaz de usuario. El objeto PagedData contiene 2 propiedades importantes, que puede ver en el lado derecho del encabezado de resultados en la página de resultados de búsqueda en la interfaz de usuario de Netsuite:

- **contar** (el número total de los resultados)
- **pageRanges** (lista de páginas, disponible en UI como selector de cuadro combinado)

El parámetro options.pageSize se limita de nuevo a 1000 filas de resultados.
El método **PagedData.fetch** se usa para obtener la parte del resultado que desea (indexado por el parámetro pageIndex). Con un poco más de código, recibirá la misma función de devolución de llamada conveniente que Search.ResultSet.each, sin tener el límite de 4000 filas.

```
// 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');

    });

}
```

Permite calcular la gobernanza. Tenemos 5 unidades para las llamadas runPaged () y 1 + count / 1000 pagedData.fetch que toman 5 unidades cada una, por lo que:

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

Ejemplo: 9500 filas tomarán 55 unidades. Aproximadamente la mitad de las unidades de gobierno

de getRange.

Usando un mapa dedicado / Reducir script

Para resultados de búsqueda realmente grandes, puede usar un mapa dedicado / Reducir script. Es mucho más inconveniente, pero a veces inevitable. Y a veces podría ser muy útil.
El truco aquí es que, en la etapa Obtener datos de entrada, puede proporcionar al motor de NS no los datos reales (es decir, el resultado del script), sino solo la definición de la búsqueda. NS ejecutará la búsqueda por ti sin contar las unidades de gobierno. Luego, cada fila de un solo resultado se pasará a la etapa Mapa.
Por supuesto, hay una limitación: el tamaño total persistido de los datos para un script de mapa / reducción no puede exceder los 50 MB. En un resultado de búsqueda, cada clave y el tamaño serializado de cada valor se cuentan para el tamaño total. "Serializado" significa que la fila del resultado de la búsqueda se convierte en cadena con JSON.stringify. Por lo tanto, el tamaño del valor es proporcional al número de columnas de resultados de búsqueda en un conjunto de resultados. Si llega a tener problemas con el error STORAGE_SIZE_EXCEEDED, considere la posibilidad de reducir las columnas, combinarlas en fórmulas, agrupar el resultado o incluso dividir la búsqueda en varias búsquedas secundarias, que podrían ejecutarse en las etapas Mapa o Reducir.

```
/**
 * @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
    };
});
```

Por supuesto, el ejemplo aquí es simplificado, sin manejo de errores y se da solo para compararlo con otros. Hay más ejemplos disponibles en Map / Reduce Script Type en el Centro de ayuda de NS

Lea Búsquedas con gran cantidad de resultados. en línea: https://riptutorial.com/es/netsuite/topic/10687/busquedas-con-gran-cantidad-de-resultados-

Capítulo 5: Búsquedas de scripts con expresiones de filtro

Introducción

Cuando crea búsquedas con Suitescript, puede proporcionar como "filtros" una matriz de objetos de filtro o expresión de filtro. La segunda opción es más legible y le brinda una opción muy flexible para proporcionar expresiones anidadas (hasta 3 niveles) utilizando no solo los operadores "AND" predeterminados, sino también los operadores "OR" y "NOT".

Examples

Término de filtro

Para entender las expresiones de filtro, deberíamos comenzar con Filter Term. Esta es una simple **serie de cadenas** , que contiene al menos 3 elementos:

1. **Filtro** (Campo / Campo de unión / Fórmula / Resumen)
2. **Operador** (search.Operator)
3. **Valores** (valor de cadena (o matriz de valores de cadena), para usar como parámetro de filtro)

```
// 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']
```

En algunos campos de selección, puede utilizar valores especiales.

```
 // 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@']
```

Expresión de filtro

La expresión de filtro simple es también una **matriz** . Contiene uno o más términos de filtro, combinados con los operadores: 'AND', 'OR', 'NOT'. (Los operadores no distinguen entre mayúsculas y minúsculas):

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

Las expresiones de filtro más complejas, podrían contener términos de filtro **Y** expresiones de filtro anidadas, combinadas con operadores. No se permiten más de 3 niveles de expresiones anidadas:

```
[
  ['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']
 ]
```

Y finalmente, pongamos todo esto en conjunto en una muestra 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']
              ]
});
```

Expresiones de filtro vs Objetos de filtro

Las expresiones de filtro **no pueden incluir** objetos de filtro. Esto es muy importante. Si decide formar sus filtros con Expresión de filtro, utiliza una matriz de matrices de cadenas. La siguiente sintaxis es **incorrecta** :

```
// 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
});
```

En su lugar, utilice el **correcto** :

```
// 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 ]
```

```
});
```

o si desea mantener el enfoque de Objetos de filtro, pase una serie de objetos de filtro y olvídese de los operadores 'AND', 'OR', 'NOT'. Siempre sera **y**

```
// 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
});
```

Consejos útiles

1. Aquí puede encontrar la lista de valores de filtro de búsqueda disponibles para los datos de fecha:
   https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3010842.html
   Estos se pueden usar en expresiones como:

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

2. Aquí están los operadores de búsqueda:
   https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3005172.html
   Por supuesto que puede utilizar **serach.Operator** enum:
   https://system.netsuite.com/app/help/helpcenter.nl?fid=section_4345782273.html

3. Aquí están los tipos de resumen de búsqueda:
   https://system.netsuite.com/app/help/helpcenter.nl?fid=section_N3010474.html

4. Puede usar el operador ANYOF solo en campos de tipo de selección (Lista / Registro). Si desea usarlo en campos de texto libre (como nombres, correos electrónicos, etc.), la única forma es crear una Expresión de filtro anidada con operadores 'OR':

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

o puedes escribir un pequeño script, haciendo esto en lugar de ti:

```
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. ¿Todavía no estás seguro? Buscando una trampa? :)
   Cree una búsqueda guardada en la interfaz de usuario de Netsuite, tome el ID de búsqueda (digamos: customsearch1234) y log.debug la expresión de filtro:

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

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

Lea Búsquedas de scripts con expresiones de filtro en línea:
https://riptutorial.com/es/netsuite/topic/10732/busquedas-de-scripts-con-expresiones-de-filtro

Capítulo 6: Cargando un registro

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

Este ejemplo asume que el módulo de registro está establecido en la variable RECORDMODULE, como se muestra a continuación.

```
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,
    });
});
```

Lea Cargando un registro en línea: https://riptutorial.com/es/netsuite/topic/4685/cargando-un-registro

Capítulo 7: Crear un registro

Examples

Crear nueva tarea

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

Creación de registro en modo dinámico

var record = nlapiCreateRecord ('customrecord_ennveeitissuetracker', {recordmode: 'dynamic'}); nlapiLogExecution ('DEBUG', 'record', record); record.setFieldValue ('custrecord_name1', nombre); record.setFieldValue ('custrecord_empid', id); record.setFieldValue ('custrecord_contactno', contactno); record.setFieldValue ('custrecord_email', correo electrónico); record.setFieldValue ('custrecord_location', loc); record.setFieldValue ('custrecord_incidentdate', incidentdate); record.setFieldValue ('custrecord_issuedescription', desc); // record.setFieldValue ('custrecord_reportedby', report); record.setFieldValue ('custrecord_issuetype', issuetype); record.setFieldValue ('custrecord_priority', prioridad); // record.setFieldValue ('custrecord_replacementprovided', repl); record.setFieldValue ('custrecord_issuestatus', issuestatus); // record.setFieldValue ('custrecord_resolvedby', resolverby); record.setFieldValue ('custrecord_remarks', comentarios); record.setFieldValue ('custrecord_resolvedby', resuelto por); record.setFieldValue ('custrecord_updatedstatus', updatedstatus); var id = nlapiSubmitRecord (record, true); var recordId = nlapiGetRecordId (); record = nlapiLoadRecord ('customrecord_ennveeitissuetracker', id);

Lea Crear un registro en línea: https://riptutorial.com/es/netsuite/topic/5127/crear-un-registro

Capítulo 8: Descripción general del tipo de script

Introducción

Puede crear personalizaciones de SuiteScript utilizando un sistema controlado por eventos. Usted define varios tipos de registros de Script, cada uno de los cuales tiene su propio conjunto único de eventos, y en su archivo fuente, define las funciones a las que se llamará para manejar esos eventos a medida que ocurren.

Los scripts son uno de los componentes principales con los que diseñará y construirá sus aplicaciones. El objetivo de este artículo es simplemente familiarizarse con los tipos de secuencias de comandos y los eventos disponibles.

Examples

El script del cliente

El script del cliente es uno de los tipos de script más comúnmente utilizados y complejos disponibles para usted. Como su nombre lo indica, el script del cliente se ejecuta en el navegador, es decir, en el lado del cliente. Es el único tipo de script que se ejecuta en el lado del cliente; todos los demás se ejecutarán en el lado del servidor de NetSuite.

El uso principal de Client Script es para responder a las interacciones de los usuarios con los formularios de registro dentro de la interfaz de usuario de NetSuite.

Tan pronto como el usuario carga un formulario de registro en el modo de edición, se `pageInit` un evento `pageInit` que podemos usar para ejecutar el código cuando se inicializa el formulario, antes de que el usuario pueda interactuar con él.

Cada vez que el usuario cambia un campo en el formulario, se activarán una serie de eventos:

1. Se `validateField` evento `validateField` que nos permite validar el valor que el usuario está tratando de ingresar en el campo. Podemos usar esto para aceptar o evitar que se produzca el cambio.
2. Luego se dispara un evento `fieldChanged` que nos permite responder al nuevo valor en el campo.
3. Por último, se `postSourcing` un evento `postSourcing` después de que todos y cada uno de los campos dependientes también hayan obtenido sus valores. Esto nos permite responder al cambio *y* asegurarnos de que estamos trabajando con todos los datos correctos.

Esta serie de eventos se activa sin importar si el usuario está cambiando un campo de cuerpo o un campo de sublista.

A medida que el usuario realice cambios en las líneas de sub-listas, se activará otra serie de

eventos:

1. Se `lineInit` evento `lineInit` cada vez que el usuario selecciona inicialmente una línea nueva o existente, antes de que puedan realizar cambios en los campos de la línea.
2. Cuando el usuario hace clic en el botón *Agregar* para agregar una nueva línea, se `validateLine` un evento `validateLine` que nos permite verificar que toda la línea es válida y se puede agregar al registro.
3. Cuando el usuario hace clic en el botón *Insertar* para agregar una nueva línea por encima de una existente, se `validateInsert` un evento `validateInsert` , que funciona exactamente igual que el evento `validateLine` .
4. De forma similar, siempre que el usuario intente eliminar una línea, se `validateDelete` un `validateDelete` que permite permitir o denegar la eliminación de la línea.
5. [Solo SuiteScript 1.0] Finalmente, después de que el evento de validación apropiado tenga éxito, si el cambio en la línea también `recalc` el monto total de una transacción, se `recalc` un evento de `recalc` que nos permite responder al cambio en el monto de nuestra transacción. transacción.
6. [Solo para SuiteScript 2.0] Finalmente, después de que el evento de validación apropiado tenga éxito, se `sublistChanged` un evento `sublistChanged` para permitirnos responder al cambio de línea completado.

Finalmente, cuando el usuario hace clic en el botón *Guardar* en el registro, se `saveRecord` un evento `saveRecord` que nos permite validar si el registro es válido y se puede guardar. Podemos evitar que ocurra el guardado o permitir que continúe con este evento.

La secuencia de comandos del cliente tiene, con mucho, la mayoría de los eventos de cualquier tipo de secuencia de comandos, y la relación más compleja entre esos eventos.

El script de eventos del usuario

Muy relacionado con el script del cliente está el script de eventos del usuario. Los eventos de este tipo de script se activan nuevamente cuando se carga o guarda un registro, pero en su lugar se ejecuta en el lado del servidor. Como tal, no se puede utilizar para responder inmediatamente a los cambios de campo, pero tampoco se limita a los usuarios que interactúan con el registro en un formulario.

Los scripts de eventos de usuario se ejecutarán sin importar de dónde venga la carga o la solicitud de envío, ya sea un usuario que trabaja en la interfaz de usuario, una integración de terceros u otro script interno que realiza la solicitud.

Cada vez que un proceso o usuario intenta leer un registro fuera de la base de datos, se desencadena el evento `beforeLoad` del Evento de `beforeLoad` . Podemos usar esto para preprocesar datos, establecer valores predeterminados o manipular el formulario de UI antes de que el usuario lo vea.

Una vez que un proceso o usuario intenta enviar un registro a la base de datos, ya sea la creación de un nuevo registro, la edición de un registro existente o la eliminación de un registro, se produce la siguiente secuencia:

1. Primero, antes de que la solicitud `beforeSubmit` a la base de datos, se `beforeSubmit` un evento `beforeSubmit` . Podemos usar este evento, por ejemplo, para limpiar el registro antes de que llegue a la base de datos.
2. La solicitud se envía a la base de datos y el registro se crea / modifica / elimina en consecuencia.
3. Una vez que se completa el procesamiento de la base de datos, se `afterSubmit` un evento `afterSubmit` . Podemos usar este evento, por ejemplo, para enviar notificaciones de cambios por correo electrónico o para sincronizar con sistemas de terceros integrados.

También puede ver esta serie de videos que ayudan a visualizar los eventos de este tipo de script.

Los scripts programados y mapear / reducir

Hay dos tipos de scripts que podemos aprovechar para ejecutar el procesamiento en segundo plano en un intervalo regular específico; estos son los scripts *Programados* y *Mapa / Reducir* . Tenga en cuenta que el tipo de script *Map / Reduce* solo está disponible en SuiteScript 2.0. El script *programado* está disponible tanto para 1.0 como para 2.0.

La secuencia de comandos programada solo tiene un único evento de `execute` que se activa en cualquier programación que defina. Por ejemplo, es posible que desee ejecutar un script nocturno que aplique los pagos a las facturas, o un script por hora que sincronice los datos con un sistema externo. Cuando llega el intervalo de tiempo, NetSuite dispara este evento de `execute` en su script programado.

El script Map / Reduce funciona de manera similar, pero una vez que se activa, divide el procesamiento en cuatro fases distintas:

1. La fase `getInputData` es donde recopila todos los datos de entrada que necesitará para completar el proceso de negocios. Puede usar esta fase para realizar búsquedas, leer registros y empaquetar sus datos en una estructura de datos descifrable.
2. NetSuite pasa automáticamente los resultados de su fase `getInputData` a la segunda fase, llamada `map` . Esta fase es responsable de agrupar los datos de entrada de forma lógica para su procesamiento. Por ejemplo, si está aplicando pagos a las facturas, es posible que desee agrupar primero las facturas por Cliente.
3. Los resultados de la fase del `map` se pasan luego a la fase de `reduce` , que es donde se lleva a cabo el procesamiento real. Aquí es donde usted, siguiendo nuestro ejemplo, aplicaría los pagos a las facturas.
4. Por último, se invoca una fase de `summary` que contiene datos sobre los resultados de todo su procesamiento en las tres fases anteriores. Puede usar esto para generar informes o enviar correos electrónicos con el procesamiento completo.

La principal ventaja de la secuencia de comandos Map / Reduce es que NetSuite paralelizará automáticamente el procesamiento para usted en múltiples colas, si está disponible.

Ambos tipos de script tienen un límite de gobierno extremadamente grande, por lo que también puede usarlos para procesos en masa o procesos en general de larga ejecución.

El intervalo más corto que se puede configurar para ejecutar cada uno de estos tipos de script es cada 15 minutos.

Ambos tipos de secuencias de comandos también pueden ser invocados a pedido por los usuarios o por otras secuencias de comandos, si es necesario.

Los scripts Suitelet y Portlet

A menudo, desearemos crear páginas de IU personalizadas en NetSuite; entrar en la Suitelet. El script de Suitelet está diseñado para crear páginas de UI internas y personalizadas. Las páginas pueden ser HTML de forma libre, o pueden utilizar las API de UI Builder de NetSuite para construir formularios que sigan la apariencia de NetSuite.

Cuando se implementa, un Suitelet recibe su propia URL única. El Suitelet luego tiene un solo evento de `render` que se invoca cada vez que esa URL recibe un `GET` HTTP `GET` o `POST` . Normalmente, la respuesta a la solicitud `GET` sería representar el formulario en sí, y luego el formulario se `POST` a `POST` para procesar los datos del formulario.

También podemos aprovechar los Suitelets para crear progresiones de UI de estilo asistente usando los componentes de UI "Asistente" de NetSuite.

Los portlets son extremadamente similares a los Suitelets, excepto que se usan específicamente para crear widgets de tablero personalizados en lugar de páginas personalizadas completas. Aparte de eso, los dos tipos de scripts funcionan muy parecidos.

El RESTlet

Los RESTlets nos permiten crear puntos finales personalizados basados en REST en NetSuite; por lo tanto, los RESTlets forman la columna vertebral de casi cualquier integración en NetSuite.

Los RESTlets proporcionan controladores de eventos individuales para cuatro de los métodos de solicitud HTTP más utilizados:

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

Cuando un RESTlet recibe una solicitud, la enrutará a la función apropiada del controlador de eventos en función del método de solicitud HTTP utilizado.

La autenticación a un RESTlet se puede realizar a través de una sesión de usuario, encabezados HTTP o tokens OAuth.

El guión de actualización masiva

Usando el script de actualización masiva, podemos crear actualizaciones masivas personalizadas para que los usuarios realicen. Esto funciona igual que una actualización masiva normal, donde el usuario selecciona el tipo de actualización masiva, crea una búsqueda que devuelve los registros

a actualizar, y luego cada resultado de búsqueda se pasa individualmente al script personalizado de actualización masiva.

La secuencia de comandos proporciona un solo controlador de `each` evento que recibe la identificación interna y el tipo de registro del registro que se actualizará.

Los usuarios deben activar manualmente los scripts de actualización masiva a través de la interfaz estándar de actualización masiva.

Los scripts de actualización masiva tienen un límite de gobierno masivamente alto y están diseñados para el procesamiento masivo personalizado de uso común.

El script de acción de flujo de trabajo

Los flujos de trabajo pueden ser algo limitados en su funcionalidad; por ejemplo, los flujos de trabajo no pueden interactuar con artículos de línea. El tipo de secuencia de comandos de acción de flujo de trabajo está destinado a ser invocado por un flujo de trabajo para agregar funcionalidad de secuencias de comandos para lograr lo que el flujo de trabajo en sí no puede.

Las acciones de flujo de trabajo tienen un único controlador de eventos `onAction` que será invocado por el flujo de trabajo.

El script de instalación de paquete

Por último, tenemos el tipo de script de instalación de paquete, que proporciona varios eventos que nos permiten interactuar con la instalación, actualización y desinstalación de un paquete en particular. Este es un tipo de script raramente encontrado, pero es importante tenerlo en cuenta.

La instalación del paquete incluye los siguientes controladores de eventos, que deberían explicarse por sí mismos:

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

Lea Descripción general del tipo de script en línea:
https://riptutorial.com/es/netsuite/topic/7829/descripcion-general-del-tipo-de-script

Capítulo 9: Edición en línea con SuiteScript

Introducción

La edición en línea permite a los usuarios modificar y actualizar muy rápidamente los datos de un registro en particular sin tener que cargar todo el registro en una página, editar el formulario y luego guardar el registro.

Los desarrolladores de NetSuite tienen una funcionalidad correspondiente llamada `submitFields` . La funcionalidad `submitFields` es proporcionada por la función global `nlapiSubmitField` en SuiteScript 1.0 y el método `N/record#submitFields` en SuiteScript 2.0.

Sintaxis

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

Parámetros

Parámetro	Detalles
recordType	`String` : la ID interna del tipo de registro que se está actualizando
recordId	`String` o `Number` : el ID interno del registro que se está actualizando
FieldIds	`String` o `String[]` : los ID internos de los campos que se están actualizando.
fieldValues	`any` o `any[]` - Los valores correspondientes que se establecerán en los campos dados
doSourcing	`Boolean` : si los valores dependientes se deben obtener al enviar el registro. El valor predeterminado es `false`

Observaciones

La funcionalidad `submitFields` es una característica complementaria de la funcionalidad `lookupFields` .

Rendimiento y limitaciones

`submitFields` desempeña significativamente más rápido y usa menos `submitFields` que los mismos cambios al cargar y enviar el registro completo.

Se pueden actualizar varios campos a la vez por el mismo costo que actualizar un solo campo. La actualización de más campos con `submitFields` *no implica* un mayor costo de gobierno.

Sin embargo, debe tener en cuenta que solo ciertos campos en cada tipo de registro se pueden editar en línea, y el ahorro de rendimiento *solo se* aplica a estos campos editables en línea. Si usa la función `submitFields` en cualquier campo no editable en línea, el campo *se* actualizará correctamente, pero entre bambalinas, NetSuite realmente cargará y enviará el registro, por lo que tomará más tiempo y usará más gobernanza. Puede determinar si un campo se puede editar en línea consultando la columna "nlapiSubmitField" en el Explorador de registros .

`submitFields` funcionalidad de `submitFields` también se limita a los campos del *cuerpo* de un registro. Si necesita modificar los datos de la lista secundaria, deberá cargar el registro para realizar los cambios y luego enviar el registro.

Referencias:

- Ayuda de NetSuite: "Edición en línea y descripción general de SuiteScript"
- NetSuite Help: "Edición en línea usando nlapiSubmitField"
- Ayuda de NetSuite: "Consecuencias de usar nlapiSubmitField en campos editables no en línea"
- NetSuite Help: "API de campo"
- NetSuite Help: "record.submitFields (opciones)"

Examples

[1.0] Enviar un campo único

```
/**
 * 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] Enviar campos múltiples

```
/**
 * 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] Enviar un campo único

```
/**
 * 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] Enviar campos múltiples

```
/**
 * 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
        }
    });
});
```

Lea Edición en línea con SuiteScript en línea: https://riptutorial.com/es/netsuite/topic/9082/edicion-en-linea-con-suitescript

Capítulo 10: Ejecutando una búsqueda

Examples

Búsqueda ad hoc SS 2.0

```
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 de la búsqueda guardada

```
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
        }
    }
});
```

Lea Ejecutando una búsqueda en línea: https://riptutorial.com/es/netsuite/topic/6081/ejecutando-una-busqueda

Capítulo 11: Ejecutando una búsqueda

Examples

SS 2.0 de la búsqueda guardada

```
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
        }
    }
});
```

Búsqueda ad hoc SS 2.0

```
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
        }
    }
});
```

Realizando una búsqueda resumida

```
// 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."
    });
});
```

Lea Ejecutando una búsqueda en línea: https://riptutorial.com/es/netsuite/topic/6359/ejecutando-una-busqueda

Capítulo 12: Eliminar en masa

Introducción

Este ejemplo muestra cómo eliminar registros en masa en NetSuite al aprovechar la función Actualización masiva. Por lo general, se nos dice que no eliminemos registros, sino que los hagamos inactivos, pero si es necesario, entonces este pequeño script hace exactamente eso. Una vez que el script se implementa como un tipo de script 'Actualización masiva', simplemente vaya a Listas> Actualización masiva> Actualizaciones masivas> Actualizaciones personalizadas. Debería ver su eliminación masiva. A continuación, configure los criterios de búsqueda en su eliminación masiva y realice una vista previa para validar sus datos antes de eliminarlos.

Examples

Eliminar basado en criterios de búsqueda

```
/**
 *  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
    }
}
```

Lea Eliminar en masa en línea: https://riptutorial.com/es/netsuite/topic/9062/eliminar-en-masa

Capítulo 13: Entender las búsquedas de transacciones

Introducción

Una comprensión profunda de cómo funciona la búsqueda de transacciones es un conocimiento crucial para todos los desarrolladores de NetSuite, pero el comportamiento predeterminado de estas búsquedas, y el control de ese comportamiento, puede ser bastante confuso al principio.

Observaciones

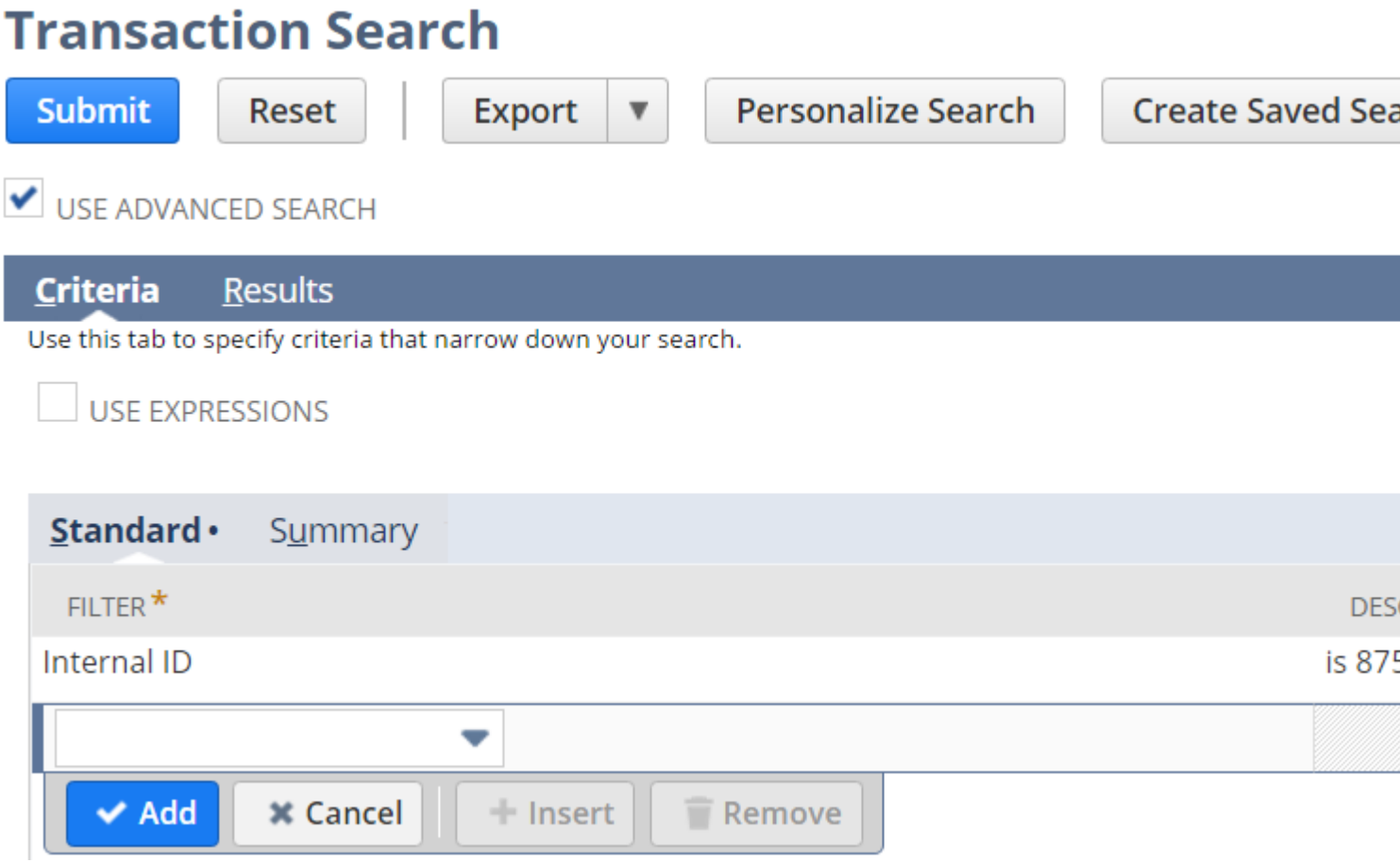
Referencias:

- Página de ayuda de NetSuite: "Uso de la línea principal en los criterios de búsqueda de transacciones"

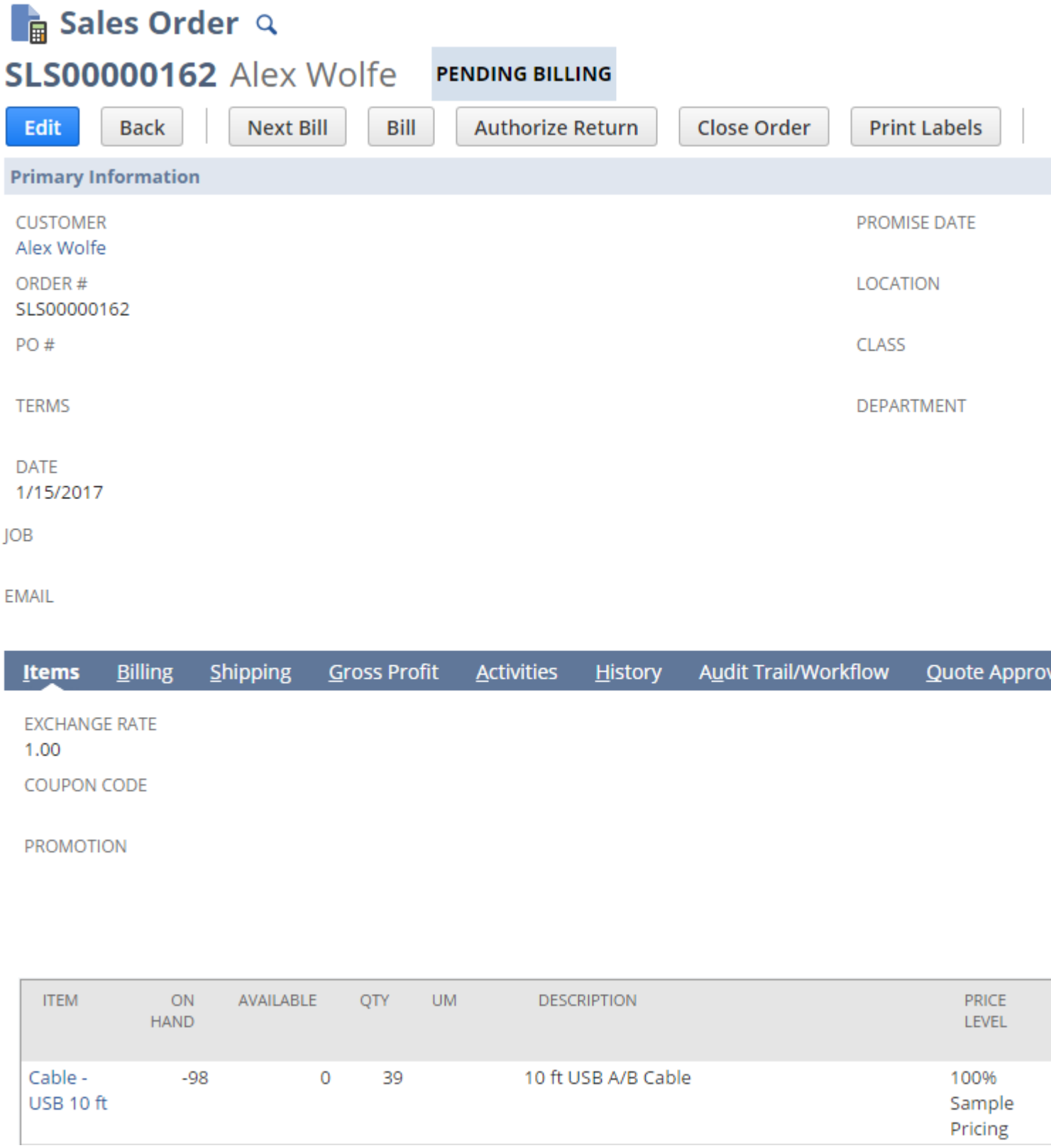
Examples

Filtrado solo en ID interna

Exploremos un ejemplo de búsqueda de transacciones donde definimos un filtro para el ID interno de una sola transacción:

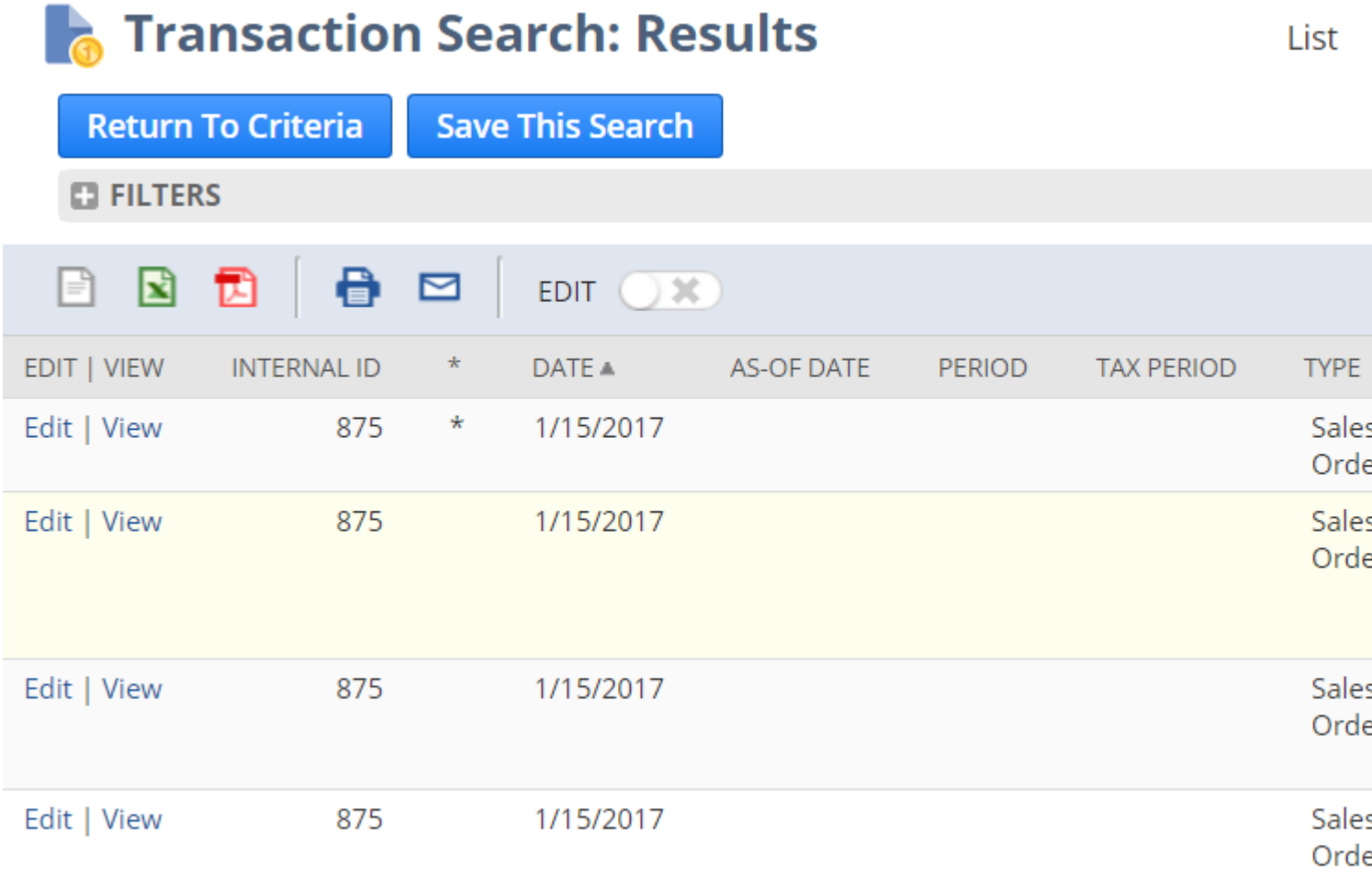


Hemos especificado un filtro para que solo nos muestre los resultados de la transacción con el ID interno de 875; Aquí está esa transacción:



Podemos ver que es una orden de venta con una sola línea de pedido.

Debido a que las ID internas son únicas en todas las transacciones, podemos esperar solo un resultado de búsqueda para esta búsqueda. Aquí está el resultado de la búsqueda:



En lugar del resultado único que esperamos, obtenemos *cuatro* resultados. Además, cada resultado tiene exactamente la misma identificación interna. ¿Cómo es eso posible?

Para comprender lo que ocurre aquí, debemos recordar que los datos almacenados en los registros de NetSuite se dividen en dos categorías:

1. Datos del cuerpo: Datos almacenados en campos independientes del registro (por ejemplo, Fecha, Representante de ventas, Número de documento, Código de cupón)
2. Datos de lista: datos almacenados en listas dentro de cada registro, generalmente se muestran en subfichas en la interfaz de usuario (por ejemplo, artículos en una orden de venta)

Las transacciones contienen múltiples listas secundarias de datos, incluido su:

- artículos de línea
- Información de envío
- Información sobre los impuestos
- COGS (Costo de Bienes Vendidos) detalles

En estos resultados de búsqueda, NetSuite en realidad nos muestra un resultado para el cuerpo de la transacción, y luego otros resultados para los datos de las diversas sublistas dentro de la misma transacción.

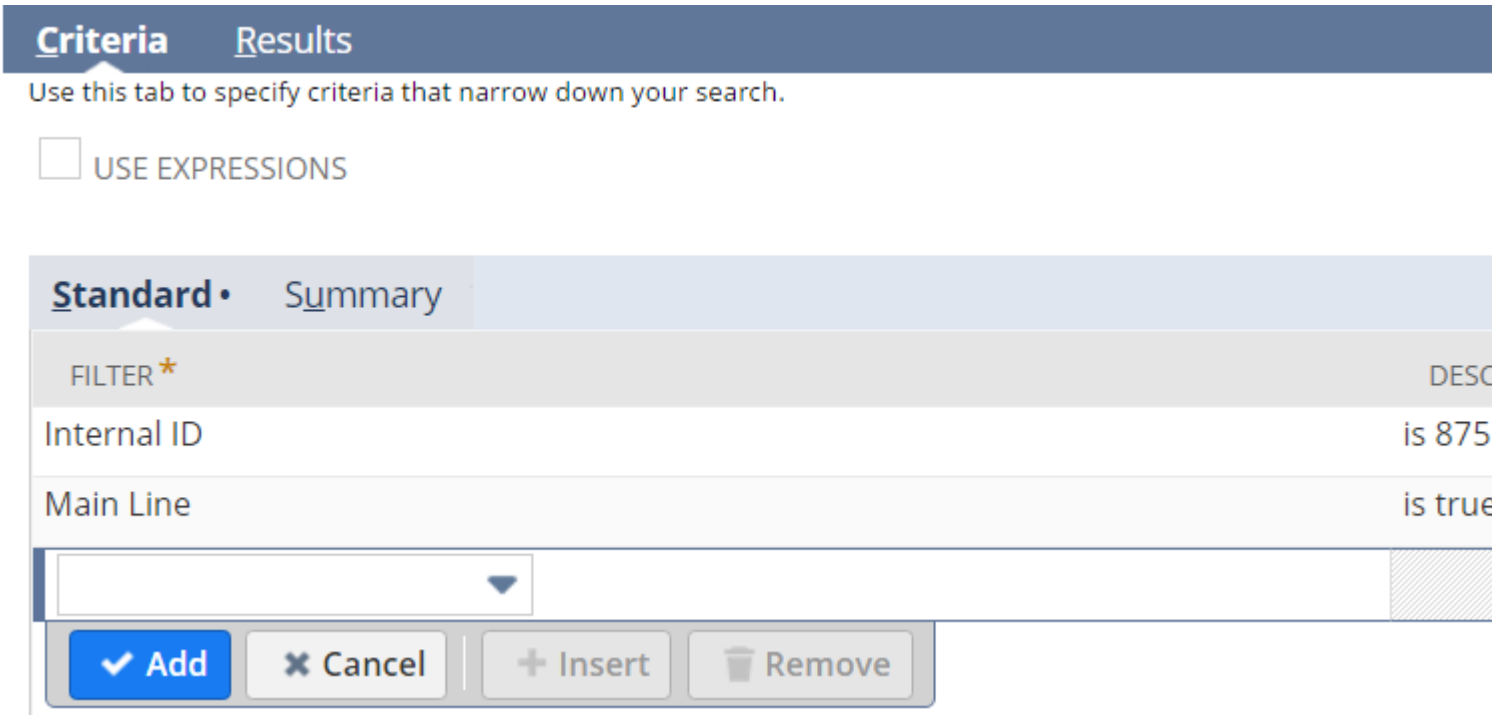
Observe la columna en nuestros resultados de búsqueda simplemente nombrada con un asterisco (*). Observe también que uno de los resultados tiene un asterisco rellenado en esta columna, mientras que el resto está vacío. Esta columna indica qué resultado de búsqueda representa el cuerpo de la transacción, que también se denomina Línea principal de la transacción.

Hay ocasiones en las que deseará que las búsquedas de transacciones solo muestren los datos de la Línea principal, y en ocasiones en que solo querrá los detalles de nivel de línea. Los ejemplos restantes muestran cómo controlar lo que aparece en nuestros resultados.

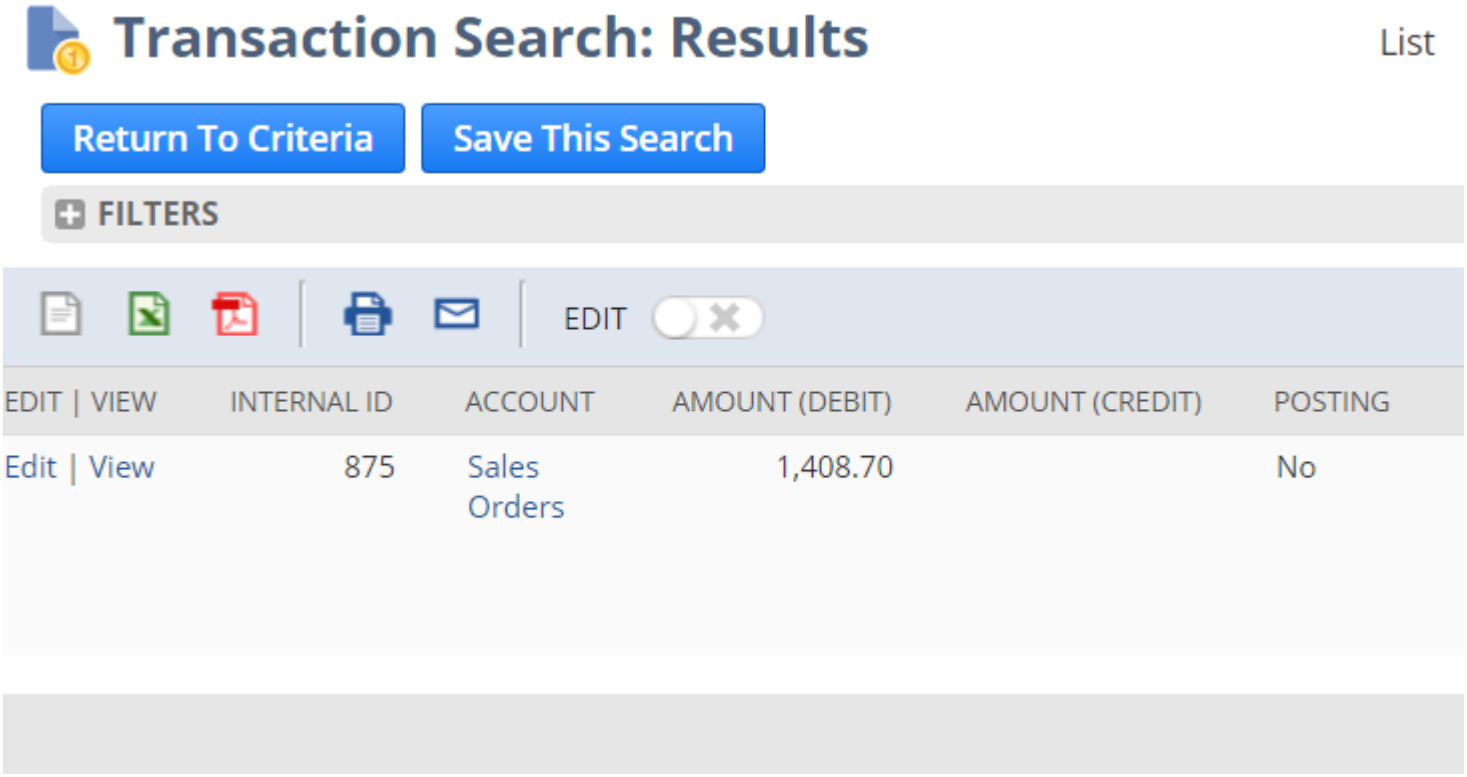
Filtrado con línea principal

Cuando solo queremos un resultado por transacción, eso significa que solo queremos el Cuerpo o Línea principal de cada transacción. Para lograr esto, hay un filtro llamado "Línea principal".

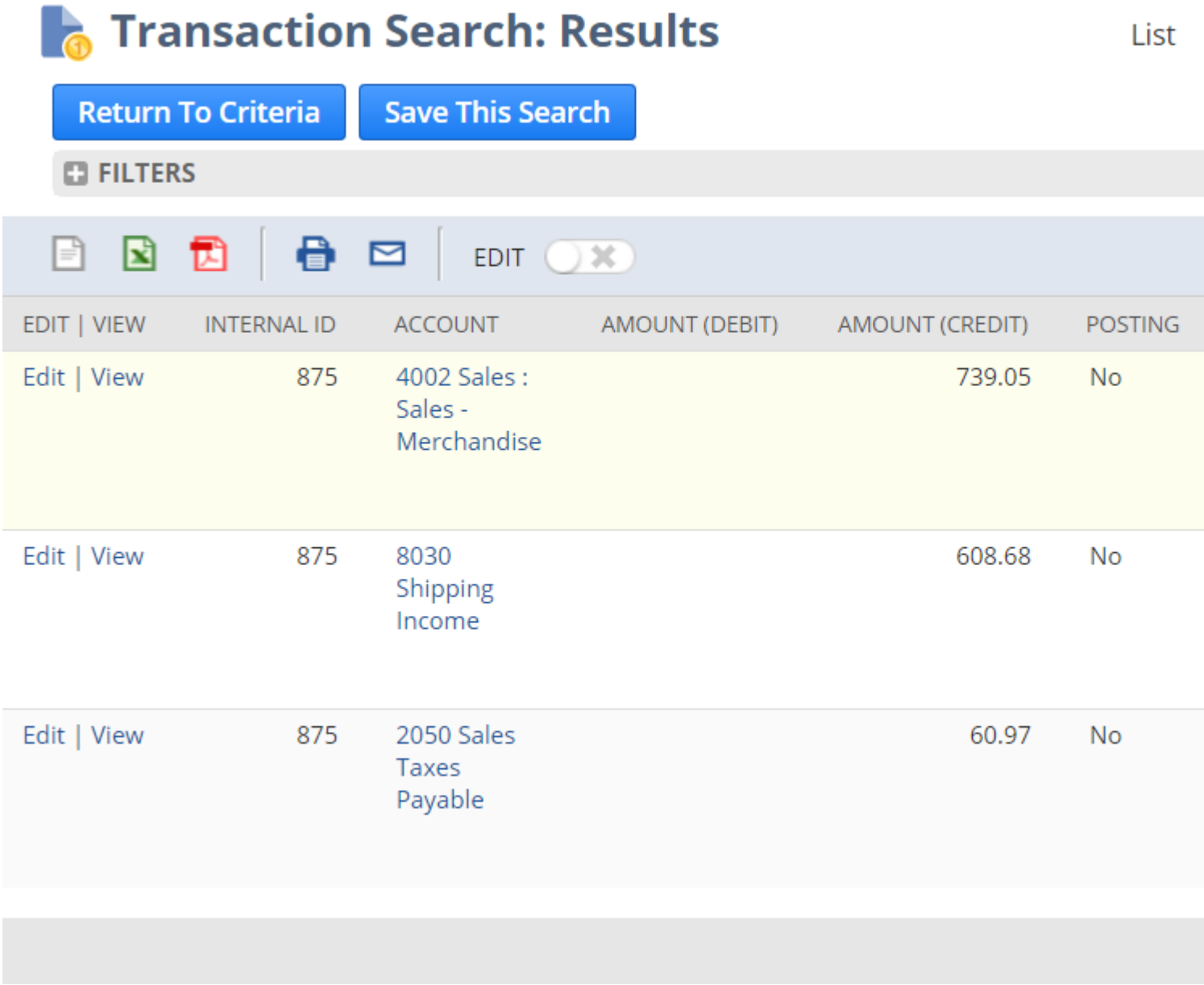
Al establecer el filtro de la *línea principal* en *Sí* en nuestros criterios de búsqueda, básicamente estamos diciendo "Mostrar solo datos a nivel de cuerpo para las transacciones en mis resultados":



La modificación de nuestros criterios de búsqueda anteriores de esta manera ahora nos da el único resultado que originalmente esperábamos:



Si cambiamos nuestro filtro de la *línea principal* a *No* , estamos diciendo "Mostrar solo los datos de los sublistas en mis resultados":



Para recapitular el comportamiento de la *línea principal* :

- Con *la Línea principal* configurada en *Sí* , recibimos *un* resultado *solo* para *el cuerpo* de la transacción.
- Con *la Línea principal* configurada en *No* , recibimos *tres* resultados *solo* para *los datos* de la *sublista* de la transacción.
- Sin ningún filtro de *línea principal* , recibimos *cuatro* resultados, esencialmente la combinación de todos los datos del cuerpo y la lista secundaria de la transacción.

Tenga en cuenta que el filtro Línea principal no es compatible con las búsquedas de entradas de diario.

Filtrado de sublimistas específicos

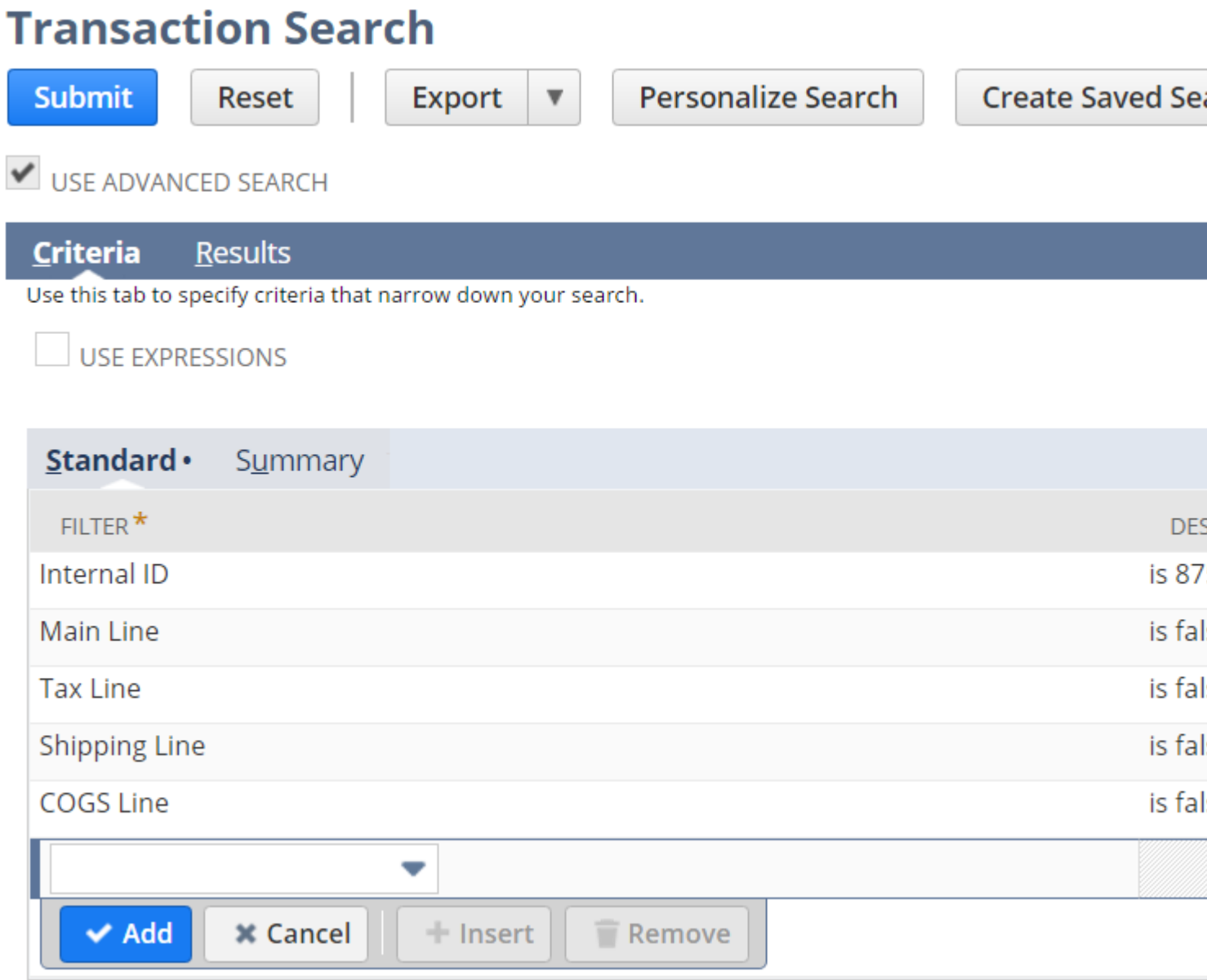
Recuerde que cada transacción contiene múltiples sublistas de datos. Ahora que podemos mostrar solo los datos de la lista secundaria utilizando *Main Line* , podemos refinar aún más nuestros resultados de búsqueda a los datos específicos de la lista secundaria.

La mayoría de las listas secundarias incluidas en los resultados de Transacción tienen un filtro de búsqueda correspondiente para alternar si se incluyen en sus resultados:

- Utilice el filtro de la *línea de envío* para controlar los datos de la lista secundaria de envío
- Utilice el filtro *Línea de impuestos* para controlar los datos de la lista de impuestos.
- Utilice el filtro de *línea COGS* para controlar los datos de la lista secundaria de COGS

Cada uno de estos filtros se comporta como *Línea principal* o cualquier otro filtro de casilla de verificación: *Sí* para incluir estos datos, *No* para excluirlos de sus resultados.

Observe que no hay un filtro para la *Línea de artículo* para controlar los datos de la lista secundaria del Artículo. Esencialmente, para poder decir "Mostrar solo los datos de la lista secundaria de elementos", necesitamos especificar todos estos filtros mencionados anteriormente como *No* en nuestros criterios:



Con este criterio, su búsqueda devolverá un resultado por línea de artículo en cada transacción coincidente.

En mi opinión, este filtro faltante es una brecha importante en la funcionalidad de búsqueda que debe solucionarse; Sería mucho más fácil y más consistente simplemente tener una *línea de elementos con el* filtro *Sí* . Hasta entonces, así es como debe especificar que solo desea datos de Artículo en los resultados de su transacción.

Lea Entender las búsquedas de transacciones en línea:
https://riptutorial.com/es/netsuite/topic/9012/entender-las-busquedas-de-transacciones

Capítulo 14: Evento de usuario: antes del evento de carga

Parámetros

Parámetro	Detalles
SuiteScript 2.0	-
`scriptContext`	{ `Object` }
`scriptContext.newRecord`	{ `N/record.Record` } Una referencia al registro que se está cargando desde la base de datos
`scriptContext.type`	{ `UserEventType` } El tipo de acción que desencadenó este evento de usuario
`scriptContext.form`	{ `N/ui/serverWidget.Form` } Una referencia al formulario de UI que se procesará
SuiteScript 1.0	-
`type`	{ `Object` } El tipo de acción que desencadenó este evento de usuario
`form`	{ `nlobjForm` } Una referencia al formulario de UI que se procesará
`request`	{ `nlobjRequest` } la solicitud HTTP GET; solo disponible cuando es activado por solicitudes del navegador

Observaciones

`beforeLoad`

El evento `Before Load` se desencadena por cualquier operación de lectura en un registro. Cada vez que un usuario, una secuencia de comandos, una importación de CSV o una solicitud de servicio web intentan leer un registro de la base de datos, se `Before Load` evento `Before Load` .

Registrar acciones que desencadenan un evento `beforeLoad` :

- Crear
- Editar
- Ver / Cargar
- Dupdo
- Impresión

- Email
- Vista rápida

Casos de uso típicos para `beforeLoad` de la `beforeLoad`

- Modificar el formulario de interfaz de usuario antes de que el usuario lo vea
- Establecer valores de campo predeterminados
- Preprocesamiento de datos

Eventos de usuario no encadenan

El código escrito en Eventos del usuario no activará ningún evento del Usuario en *otros* registros. Por ejemplo, cargar el registro de cliente asociado desde la `beforeLoad` de un registro de orden de venta *no activará* la `beforeLoad` del registro de `beforeLoad` . Incluso si está cargando otro registro de transacción, sus eventos de usuario no se activarán.

NetSuite hace esto para evitar que los eventos de usuario se activen entre sí en un bucle infinito. *Si* necesita eventos de usuario para disparar en una secuencia encadenada, tendrán que ser inyectado en los eventos entre otros tipos de script (por ejemplo, RESTlets Suitelets, scripts programados).

Event Handler devuelve `void`

El tipo de retorno del controlador de eventos `beforeLoad` es `void` . Cualquier dato devuelto por nuestro controlador de eventos no tiene ningún efecto en el sistema. No necesitamos devolver nada de nuestra función de manejador, ya que no podemos hacer nada con su valor devuelto.

Examples

Mínimo: registrar un mensaje en antes de cargar

```
// 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
    };
});
```

Modificar el formulario de la interfaz de usuario

```
// 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;
});
```

Restrinja la ejecución en función de la acción que desencadenó el evento de usuario

```
// 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;
});
```

Restrinja la ejecución según el contexto que activó el evento de usuario

En SuiteScript 1.0, recuperamos el contexto de ejecución actual utilizando `nlapiGetContext().getExecutionContext()` , luego comparamos el resultado con las cadenas en bruto apropiadas.

```
// 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;
})();
```

En SuiteScript 2.0, obtenemos el contexto de ejecución actual importando el `N/runtime` del módulo y la inspección de su `executionContext` propiedad. Luego podemos comparar su valor con los valores de enumeración en tiempo de `runtime.ContextType` lugar de cadenas sin `runtime.ContextType` .

```
// 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;
});
```

Lea Evento de usuario: antes del evento de carga en línea:
https://riptutorial.com/es/netsuite/topic/7119/evento-de-usuario--antes-del-evento-de-carga

Capítulo 15: Evento de usuario: antes y después de enviar eventos

Sintaxis

- beforeSubmit (type) // Before Submit, 1.0
- beforeSubmit (scriptContext) // Before Submit, 2.0
- afterSubmit (type) // After Submit, 1.0
- afterSubmit (scriptContext) // After Submit, 2.0

Parámetros

Parámetro	Detalles
SuiteScript 2.0	-
`scriptContext`	`{Object}`
`scriptContext.newRecord`	`{N/record.Record}` Una referencia al registro que se está leyendo de la base de datos. Podemos utilizarlo para modificar los valores de campo en el registro.
`scriptContext.oldRecord`	`{N/record.Record}` Una referencia de solo lectura al estado anterior del registro. Podemos usarlo para comparar con los nuevos valores.
`scriptContext.type`	`{UserEventType}` Una enumeración del tipo de acción de escritura que se está realizando
SuiteScript 1.0	-
`type`	`{String}` El tipo de acción de escritura que se está realizando.

Observaciones

`beforeSubmit` y después `afterSubmit`

Estos dos eventos son activados por cualquier operación de escritura de base de datos en un registro. Cada vez que un usuario, una secuencia de comandos, una importación CSV o una solicitud de servicio web intentan escribir un registro en la base de datos, se activan los eventos Enviar.

Registrar acciones que desencadenan *ambos* eventos Enviar:

- Crear
- Editar
- Borrar
- XEdit (edición en línea)
- Aprobar
- Rechazar
- Cancelar
- Paquete
- Enviar

Registre las acciones que se activan antes de `beforeSubmit` solo:

- Marca completa
- Reasignar (casos de soporte)
- Editar pronóstico

Registrar acciones que desencadenan después de `afterSubmit` solo:

- Dropship
- Orden especial
- Encargar artículos
- Pagar facturas

Casos de uso típicos de `beforeSubmit`

- Validar registro antes de que se confirma a la base de datos
- Controles de permisos y restricciones.
- Cambios de última hora antes de confirmar la base de datos
- Obtenga actualizaciones de sistemas externos

Casos de uso típicos para `afterSubmit`

- Notificación por correo electrónico de los cambios de registro
- Redirección del navegador
- Crear / actualizar registros dependientes
- Impulsar cambios a sistemas externos.

Eventos de usuario no encadenan

El código escrito en Eventos del usuario no activará ningún evento del Usuario en *otros* registros. Por ejemplo, la modificación del registro de Cliente asociado de la `beforeSubmit` de un registro de Orden de venta *no activará* los eventos de envío del registro de Cliente.

NetSuite hace esto para evitar que los eventos de usuario se activen entre sí en un bucle infinito.

Si necesita eventos de usuario para disparar en una secuencia encadenada, tendrán que ser inyectado en los eventos entre otros tipos de script (por ejemplo, RESTlets Suitelets, scripts programados).

Los manejadores de eventos devuelven el `void`

El tipo de retorno de los controladores de eventos Enviar es `void` . Cualquier dato devuelto por nuestro controlador de eventos no tiene ningún efecto en el sistema. No necesitamos devolver nada de nuestra función de manejador, ya que no podemos hacer nada con su valor devuelto.

!! PRECAUCIÓN !!

Tenga mucho cuidado al comparar valores entre registros antiguos y nuevos. Los campos vacíos desde el *antiguo* registro se devuelven como `null` , mientras que los campos vacíos desde el *nuevo* registro se devuelven como una cadena vacía. Esto significa que no puede simplemente comparar lo antiguo con lo nuevo, o obtendrá falsos positivos. Cualquier lógica que escriba debe manejar el caso donde uno es `null` y el otro es una cadena vacía de manera apropiada.

Examples

Mínimo: registrar un mensaje

```
// 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;
});
```

Antes del envío: Valide el registro antes de que se confirme en la base de datos

Para este ejemplo, queremos asegurarnos de que cualquier Empleado que esté marcado como un *Recurso de Proyecto* también tenga un *Costo Laboral* adecuado definido.

```
// 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;
});
```

Tenga en cuenta que pasamos referencias al *nuevo* registro en nuestra validación porque no nos importa cuáles fueron los valores que solían ser; solo nos preocupan los valores que están por escribirse en la base de datos. En 2.0, lo hacemos a través de la referencia `scriptContext.newRecord` , y en 1.0 llamamos a la función global `nlapiGetNewRecord` .

Cuando los datos que se envían no son válidos, creamos y lanzamos un error. En un evento `beforeSubmit` , para evitar que los cambios se escriban en la base de datos, su función debe `throw` una excepción. A menudo, los desarrolladores intentan `return false` de su función, esperando que sea suficiente, pero eso no es suficiente. Los objetos de error se crean en 2.0 usando el módulo `N/error` y en 1.0 usando la función global `nlapiCreateError` ; luego generamos una excepción utilizando nuestro objeto de error creado con la palabra clave `throw` .

Después del envío: Determine si se cambió un campo

Una vez que el registro se almacena en la base de datos, queremos inspeccionar lo que se cambió en el registro. Haremos esta inspección comparando los valores entre las instancias de registro antiguas y nuevas.

```
// 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;
});
```

Tenga mucho cuidado al comparar valores entre registros antiguos y nuevos. Los campos vacíos desde el *antiguo* registro se devuelven como `null` , mientras que los campos vacíos desde el *nuevo* registro se devuelven como una cadena vacía. Esto significa que no puede simplemente comparar lo antiguo con lo nuevo, o obtendrá falsos positivos. Cualquier lógica que escriba debe manejar el caso donde uno es `null` y el otro es una cadena vacía de manera apropiada.

Lea Evento de usuario: antes y después de enviar eventos en línea: https://riptutorial.com/es/netsuite/topic/7200/evento-de-usuario--antes-y-despues-de-enviar-eventos

Capítulo 16: Explotando columnas de fórmulas en búsquedas guardadas

Introducción

Las columnas de fórmula en las búsquedas guardadas pueden explotar muchas características de Oracle SQL y HTML. Los ejemplos muestran cómo se pueden usar estas características, así como los errores que se deben evitar.

Examples

Sentencia Oracle SQL CASE en una fórmula Netsuite

Usando una declaración CASE, muestre condicionalmente una expresión en la columna basada en los valores encontrados en otra columna, también conocida como "mi reino para un OR". En el ejemplo, el resultado se obtiene cuando el estado de la transacción es `Pending Fulfillment` o `Partially Fulfilled`:

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

Analizar un nombre de registro jerárquico usando una expresión regular

Usando una expresión regular, analice un nombre de registro que pueda ser jerárquico. La expresión busca los dos puntos finales en el nombre. Devuelve lo que sigue a los dos puntos, o el nombre completo si no hay ninguno:

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

Construye una cadena compleja concatenando múltiples campos

El ejemplo crea una cadena a partir del nombre del registro principal, el nombre de este registro y la nota de este registro.

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

Personalice el CSS (hoja de estilo) para una columna insertando un elemento DIV

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

Proteger fórmulas de cadena de corrupción y ataques de inyección

En un campo de fórmula de cadena, considere que algunos valores pueden contener subcadenas

que se parecen al navegador como HTML. A menos que esto sea intencional, es importante proteger los valores de la corrupción. Esto es útil para evitar ataques de inyección: evita que alguien ingrese HTML en un campo de comentarios en un pedido web que luego se interpreta en el escritorio del representante de servicio al cliente.

`htf.escape_sc(` *expresión* `)`

Proteger los valores de campo contra la corrupción al pasar a través de una URL

`utl_url.escape(` *expresión* `)`

Probar el valor de `mainline` en una sentencia CASE de SQL

En una fórmula de búsqueda guardada, los posibles valores de la `mainline` están diseñados para ser útiles en un contexto HTML. Cuando `mainline` es verdadero, el valor de `{mainline}` es la cadena de 1 carácter `*` (asterisco). Cuando `mainline` es false, el valor de `{mainline}` es la cadena de 6 caracteres ` ` (espacio de no ruptura, HTML codificado como una referencia de entidad de carácter). Estos valores de cadena se pueden comparar con literales de cadena en un contexto SQL.

```
CASE
WHEN {mainline} = '*' THEN expresión-when-true
WHEN {mainline} = ' ' THEN expresión-cuando-falso
END
```

Ejemplo complejo, del mundo real

El siguiente ejemplo combina varias de las técnicas cubiertas aquí. Coloca un hipervínculo en una columna con formato personalizado que, al hacer clic, abre el registro de pedido de ventas asociado a una fila. El hipervínculo está diseñado para abrir el registro en una nueva ventana o pestaña al hacer clic, y para mostrar una información sobre herramientas cuando se desplaza. El campo `internalid` usado en la URL está protegido de la codificación de la URL. El nombre del cliente, cuando esté disponible, se muestra en la misma columna, protegido de la codificación 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>'
```

Contar registros con y sin un valor proporcionado en un campo (contar valores perdidos y no perdidos)

Usando la función `NVL2()` Oracle SQL, puede crear una columna de visualización que contenga un valor si un campo contiene datos y otro valor si un campo no contiene datos. Por ejemplo, en una búsqueda de Entidad, convierta la presencia de una dirección de correo electrónico principal en una columna de visualización de texto:

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

Esto le permite contar los registros subtotalizados por la presencia o ausencia de una dirección de correo electrónico:

```
Field: Internal ID
Summary Type: Count

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

Lea Explotando columnas de fórmulas en búsquedas guardadas en línea: https://riptutorial.com/es/netsuite/topic/8298/explotando-columnas-de-formulas-en-busquedas-guardadas

Capítulo 17: Gobernancia

Observaciones

Gobernancia

"Gobernanza" es el nombre que recibe el sistema de NetSuite para detectar y detener scripts de larga ejecución, fugitivos o que requieren un uso intensivo de recursos.

Cada tipo de secuencia de comandos tiene límites de gobernabilidad que no pueden exceder, y existen cuatro tipos de límites de gobernabilidad para cada tipo de secuencia de comandos.

- Límite de uso de API
- Límite de cuenta de instrucción
- Límite de tiempo de espera
- Límite de uso de memoria

Si una secuencia de comandos supera su límite de gobernabilidad en **cualquiera** de estas cuatro áreas, NetSuite **lanzará una excepción que no se puede *atrapar*** y terminará la secuencia de comandos inmediatamente.

Límite de uso de API

NetSuite limita el uso de API de sus scripts con un sistema basado en "unidades de uso". Algunas llamadas a la API de NetSuite, particularmente las que realizan una acción de lectura o escritura en la base de datos, cuestan un número específico de unidades cada vez que se invocan. Cada tipo de script tiene un número máximo de unidades que se pueden usar durante cada ejecución del script.

Si un script excede su límite de uso de API, NetSuite termina el script lanzando un error `SSS_USAGE_LIMIT_EXCEEDED`.

A continuación se muestran algunos ejemplos de costos unitarios para operaciones comunes. Para obtener una lista exhaustiva de los costos de gobierno, consulte el artículo titulado "Gobierno de API" en la Ayuda de NetSuite.

Operación	Costo unitario
Cargando una búsqueda guardada	5
Recuperando Resultados de Búsqueda	10
Programar una tarea	10
Solicitando una URL	10

Operación	Costo unitario
Enviando un correo electrónico	10
Creando un registro personalizado	2
Creación de un registro de empleado	5
Creación de un registro de pedido de venta	10
Guardando un registro personalizado	4
Guardar un registro de contacto	10
Guardar un registro de orden de compra	20

Las diferentes operaciones utilizan diferentes cantidades de unidades, y ciertas operaciones cuestan una cantidad diferente según el tipo de registro que se utiliza. Cuanto mayor sea el número de unidades que cuesta una función, por lo general, más tardará en ejecutarse.

Las transacciones son las más grandes de los tipos de registros, por lo que trabajar con ellas cuesta la mayor cantidad de unidades. Por el contrario, los registros personalizados son muy ligeros y, por lo tanto, no cuestan muchas unidades. Los registros estándar de NetSuite que *no* son Transacciones, como Clientes, Empleados o Contactos, se ubican entre los dos en términos de costo.

Estos son los límites de uso por tipo de script:

Tipo de script	Límite de uso
Cliente	1,000
Evento de usuario	1,000
Suitelet	1,000
Portlet	1,000
Acción de flujo de trabajo	1,000
RESTlet	5,000
Programado	10,000
Mapa reducido	10,000
Instalación del paquete	10,000
Actualización masiva	10,000 por registro

Límites de tiempo de espera e instrucción

NetSuite también usa el sistema de gobierno para detectar y detener scripts fuera de control mediante el uso de un mecanismo de tiempo de espera y un contador de instrucciones.

Si un script tarda demasiado en ejecutarse, NetSuite lo detendrá lanzando un error `SSS_TIME_LIMIT_EXCEEDED` .

Además, los scripts fuera de control se pueden detectar y detener en función de su "Recuento de instrucciones". Si se exceden los límites de recuento de instrucciones definidos, NetSuite detendrá la secuencia de comandos al lanzar un error `SSS_INSTRUCTION_COUNT_EXCEEDED` .

Desafortunadamente, **no hay** documentación de ayuda que defina:

- el tiempo de espera para cada tipo de script
- los límites de conteo de instrucciones para cada tipo de script
- lo que constituye una sola "instrucción"

Simplemente es importante saber que si encuentra el error `SSS_TIME_LIMIT_EXCEEDED` o el error `SSS_INSTRUCTION_COUNT_EXCEEDED` en uno de sus scripts, el proceso se está demorando demasiado. Enfoque su investigación en sus estructuras de bucle para determinar dónde se pueden realizar las optimizaciones.

Límite de uso de memoria

Si su script excede el límite de uso de memoria, NetSuite terminará su script lanzando un error `SSS_MEMORY_USAGE_EXCEEDED` .

Cada variable declarada, cada función definida, cada Objeto almacenado contribuye al uso de memoria de su script.

Tanto la **secuencia de comandos programada como la secuencia de comandos de mapa / reducción** han documentado límites de memoria de `50MB` . También hay un límite documentado de `10MB` para el tamaño de cualquier Cadena pasada o devuelta desde un RESTlet. No hay otra documentación sobre los límites específicos para un script dado.

Examples

¿Cuántas unidades me quedan?

En SuiteScript 1.0, use `nlobjContext.getRemainingUsage()` para recuperar las unidades restantes. Una referencia `nlobjContext` se recupera utilizando la función global `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());
```

En SuiteScript 2.0, use el método `getRemainingUsage` del objeto `Script` del módulo `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()
    });
});
```

Lea Gobernancia en línea: https://riptutorial.com/es/netsuite/topic/7227/gobernancia

Capítulo 18: Registros de implementación de scripts y scripts

Introducción

Para que NetSuite sepa cómo utilizar nuestro código fuente, necesitamos poder decirle a qué funciones llamar, cuándo llamarlos y a quién llamarlos. Logramos todo esto con los registros de *Script* y *Script Deployment* .

Examples

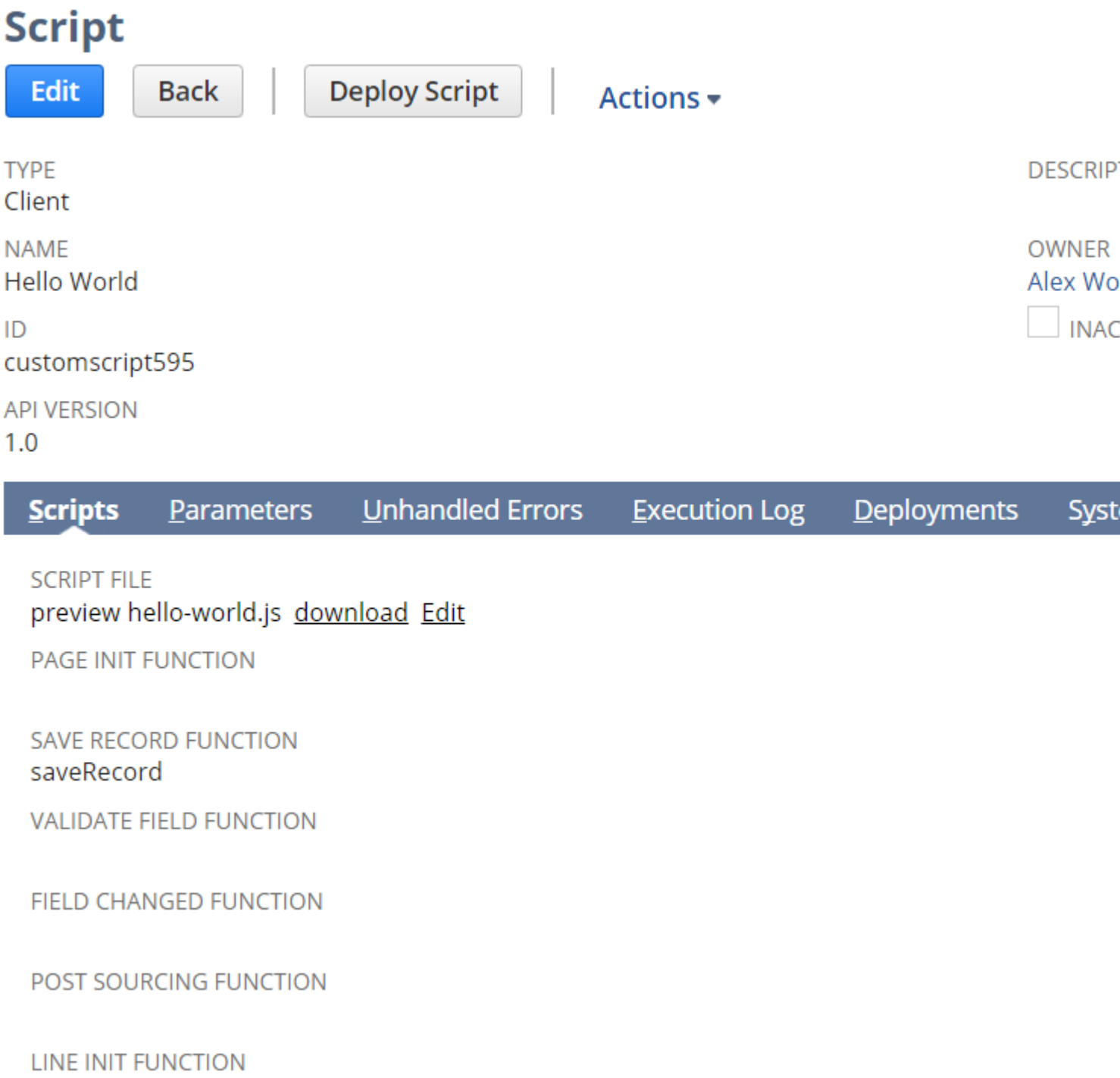
Registros de Script

NetSuite usa el registro de *Script* para asignar las funciones en su archivo fuente a eventos específicos que ocurren en el sistema. Por ejemplo, si necesita alguna lógica de negocios para ejecutar cuando se guarda un formulario en la interfaz de usuario, el registro de Script le dirá a NetSuite a qué función llamar cuando ocurra el evento `Save Record` .

Puede pensar que el registro de *Script* define *cuándo* debe ejecutarse nuestro código fuente; esencialmente define algo parecido a:

> "Cuando se guarda un registro, llame a la función saveRecord en hello-world.js".

Aquí hay un ejemplo de cómo se vería ese registro de Script:



Registros de implementación de script

Una vez que tenemos un registro de *Script* creado, entonces necesitamos implementar ese script en el sistema. Mientras que el registro de *Script* le dice a NetSuite a qué funciones llamar desde nuestro archivo fuente, el registro de *Script Deployment le* permite a NetSuite saber qué registros y usuarios debe ejecutar nuestro Script.

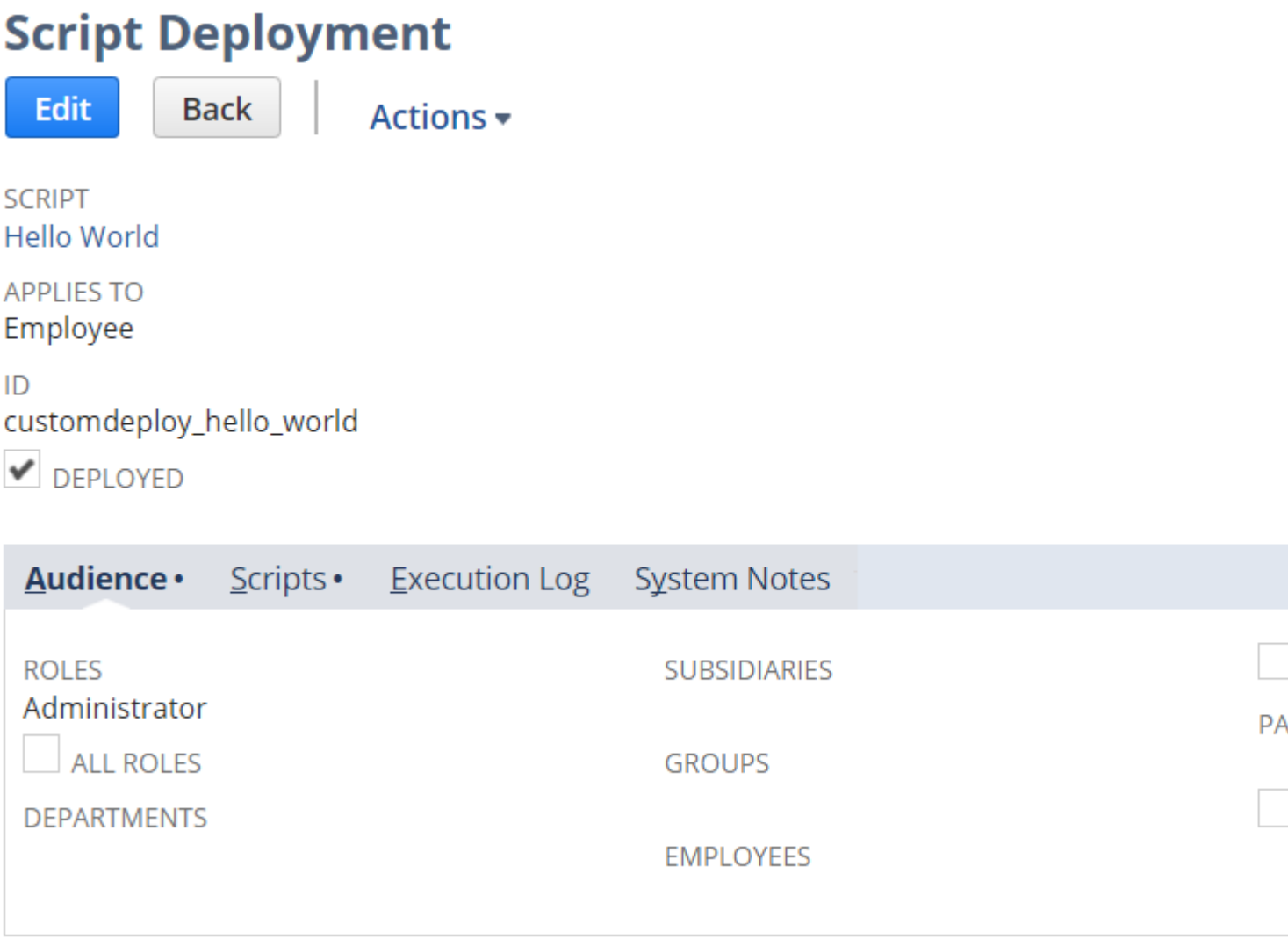
Mientras que el registro de *Script* define *cuándo* debe ejecutarse nuestro código fuente, el *Despliegue de Script* define *dónde* y *quién* puede ejecutar nuestro script. Si tenemos un registro de *Script* que dice:

"Cuando se guarda un registro, llame a la función saveRecord en hello-world.js".

entonces nuestra *Implementación de Script* para ese registro podría modificar eso ligeramente a:

"Cuando se guarda un registro de empleado, llame a la función saveRecord en hello-world.js, pero solo para los usuarios del grupo de administradores".

Una vez más, aquí hay un ejemplo de cómo se vería esa *implementación de scripts* :



Una *secuencia de comandos* puede tener varias *implementaciones de secuencias de comandos* asociadas. Esto nos permite implementar la misma lógica de negocios en múltiples tipos de registros diferentes con diferentes audiencias.

Lea Registros de implementación de scripts y scripts en línea: https://riptutorial.com/es/netsuite/topic/8835/registros-de-implementacion-de-scripts-y-scripts

Capítulo 19: RESTlet - Procesar documentos externos

Introducción

Cuando se recupera un documento de un sistema externo, es necesario que nos aseguremos de que la extensión correcta del documento esté pegada al documento. El código de muestra muestra cómo almacenar un documento correctamente en el Gabinete de archivos de NetSuite, así como adjuntarlo a su registro correspondiente.

Examples

RESTlet - almacenar y adjuntar archivo

```
 /**
 * 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;
}
```

Lea RESTlet - Procesar documentos externos en línea:
https://riptutorial.com/es/netsuite/topic/9021/restlet---procesar-documentos-externos

Capítulo 20: RestLet - Recuperar datos (Basic)

Introducción

Este ejemplo muestra la estructura básica de un script RESTlet que está destinado a ser usado para recuperar datos de un sistema externo. Los RESTlets son puntos finales que se crean para permitir la comunicación con sistemas externos.

Examples

Recuperar nombre del cliente

```
/**
 * 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);
    }
}
```

Lea RestLet - Recuperar datos (Basic) en línea:
https://riptutorial.com/es/netsuite/topic/9006/restlet---recuperar-datos--basic-

Capítulo 21: Solicitando customField, customFieldList y customSearchJoin con PHP API Advanced Search

Introducción

Éstas son algunas de las cosas más difíciles (y de las que menos se habla) que se pueden hacer con la búsqueda avanzada de la API de PHP (donde se especifican los campos).

Estoy en el proceso de migrar a rest_suite github library que usa RESTLET, y sortear el límite de concurrencia de usuarios de la API de PHP de 1.

Pero antes de borrar mi antiguo código, lo estoy publicando aquí. Las especificaciones de ejemplo para estos campos se pueden encontrar aquí: http://www.netsuite.com/help/helpcenter/en_US/srbrowser/Browser2016_1/schema/search/transactionsea

Examples

Uso de customField y 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 Usage

```
$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);
```

Lea Solicitando customField, customFieldList y customSearchJoin con PHP API Advanced Search en línea: https://riptutorial.com/es/netsuite/topic/9799/solicitando-customfield--customfieldlist-y-customsearchjoin-con-php-api-advanced-search

Capítulo 22: SS2.0 Suitelet Hello World

Examples

Basic Hello World Suitelet - Respuesta de texto simple

```
/**
*@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
    };
});
```

Lea SS2.0 Suitelet Hello World en línea: https://riptutorial.com/es/netsuite/topic/6723/ss2-0-suitelet-hello-world

Capítulo 23: SuiteScript - Procesa datos desde Excel

Introducción

A veces, los resultados de búsqueda devueltos en una Actualización masiva no son los mismos que los resultados de una búsqueda estándar, esto se debe a algunas limitaciones en una Búsqueda de actualización masiva. Un ejemplo de esto es Rev Rev Journal Journal. Por lo tanto, la solución para esto fue obtener los datos de la búsqueda estándar guardada y usar un script para leer los datos de Excel y actualizarlos, en lugar de usar la función de actualización masiva.

Examples

Actualizar Rev Rec fechas y regla

```
/**
 * 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;
}
```

Lea SuiteScript - Procesa datos desde Excel en línea:
https://riptutorial.com/es/netsuite/topic/9034/suitescript---procesa-datos-desde-excel

Capítulo 24: Trabajando con Sublistas

Introducción

Los registros de NetSuite se dividen en campos de cuerpo y sub-listas. Hay cuatro tipos de listas secundarias: Estática, Editor, Editor en línea y Lista.

Podemos agregar, insertar, editar y eliminar artículos de línea usando las API de Sublist.

Para obtener una referencia sobre qué sublistas son compatibles con SuiteScript, consulte la página de ayuda de NetSuite titulada "Scriptable Sublists".

Observaciones

Índices Sublistas

Cada elemento de línea en una lista secundaria tiene un índice que podemos usar para referenciarlo.

En SuiteScript 1.0, estos índices están basados en `1` , por lo que la primera línea de pedido tiene el índice `1` , la segunda tiene el índice `2` y así sucesivamente.

En SuiteScript 2.0, estos índices están basados en `0` , por lo que la primera línea de pedido tiene el índice `0` , la segunda tiene el índice `1` y así sucesivamente. Por supuesto, esto se ajusta más a la indexación de Array en la mayoría de los idiomas, incluido JavaScript.

Modo estándar vs dinámico

La API que utilizamos para interactuar con una lista secundaria depende de si estamos trabajando con el registro en modo Estándar o Dinámico.

Las API de modo estándar simplemente nos permiten proporcionar el índice de la línea con la que queremos trabajar como parámetro para la función apropiada.

Las API de modo dinámico siguen un patrón:

1. Selecciona la línea con la que queremos trabajar.
2. Modificar la línea seleccionada como se desee.
3. Confirmar los cambios en la línea.

En el modo dinámico, si no confirmamos los cambios en *cada* línea que modificamos, dichos cambios no se reflejarán cuando se guarde el registro.

Limitaciones

Para poder trabajar con datos de sublistas a través de SuiteScript, debemos tener una referencia en la memoria al registro. Esto significa que el registro debe recuperarse del contexto del script o debemos cargar el registro desde la base de datos.

No *podemos* trabajar con sublistas a través de la funcionalidad de búsqueda o de submitFields .

Las sublistas *estáticas* no son compatibles con SuiteScript.

Referencias:

- NetSuite Help: "¿Qué es un Sublista?"
- NetSuite Help: "Tipos de lista"
- NetSuite Help: "Sublistas de secuencias de comandos"
- NetSuite Help: "Trabajar con elementos de línea Sublist"
- NetSuite Help: "Sublist APIs"
- NetSuite Help: "Trabajar con registros en modo dinámico"

Examples

[1.0] ¿Cuántas líneas en una sublista?

```
// 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] Sublistas en Modo Estándar

```
// 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] Sublistas en modo dinámico

```
// 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] Encontrar un elemento de línea

```
// 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] ¿Cuántas líneas en una sublista?

```
// 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] Sublistas en Modo Estándar

```
// 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] Sublistas en modo dinámico

```
// 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] Encuentra un artículo de línea

```
// 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
    }
});
```

Lea Trabajando con Sublistas en línea: https://riptutorial.com/es/netsuite/topic/9098/trabajando-con-sublistas

Capítulo 25: Usando el navegador de registros NetSuite

Examples

Usando el navegador de registros NetSuite

El *Navegador de registros* define el esquema para todos los tipos de registro de secuencias de comandos; Es una herramienta de referencia extremadamente crítica para todos los desarrolladores de SuiteScript. Cuando necesite saber cómo hacer referencia a un campo particular en un tipo de registro específico en su script, el *Navegador de registros* es su guía.

Enlace directo

Otro esquema

También puede observar pestañas en la parte superior del *Navegador de registros* para el *Navegador* de *esquemas* y el *Navegador de conexión* . Estos son muy similares al *navegador de registros* , pero para diferentes API de NetSuite.

El *navegador de esquema* proporciona el esquema para la API de servicios web basados en SOAP, mientras que el *navegador de conexión* proporciona el esquema para el conector ODBC.

Navegando por el navegador de registros

Primero navega por el *Explorador* de registros por Tipo de registro, es decir, "Pedido de ventas", "Factura", "Empleado". No hay capacidad de búsqueda en el *Navegador de registros* , por lo que toda la navegación se realiza manualmente. Los tipos de registro están organizados alfabéticamente, por lo que primero debe hacer clic en la primera letra del tipo de registro que le interesa y, a continuación, seleccionar el Tipo de registro a la izquierda.

Por ejemplo, si desea ver el esquema para el tipo de registro *Subisidario* , primero debe hacer clic en *S* en la parte superior y luego en *Subsidiaria* a la izquierda.

Leyendo el esquema

Cada esquema le proporciona una cantidad abrumadora de información sobre cada tipo de registro. Es importante saber cómo desglosar toda esta información.

En la parte superior del esquema se encuentra el nombre del tipo de registro seguido de la ID interna del tipo de registro; esta identificación interna es la referencia programática para el tipo de registro. El esquema se divide en varias secciones:

- *Campos* : la sección *Campos* enumera los detalles de todos los campos del *cuerpo* del

registro. Los campos descritos aquí se pueden usar cuando se trabaja con el registro actualmente en contexto, o con una referencia directa a un objeto de registro.
- *Sublistas* : la sección *Sublistas* muestra todas las sublistas en el registro y cada columna de secuencias de comandos dentro de cada sublista. Los campos en esta sección nuevamente se aplican cuando está trabajando con el registro actualmente en contexto o con una referencia directa a un objeto de registro.
- *Fichas:* La sección *aquí* se describen todas las subpestañas nativas en el tipo de registro.
- *Buscar uniones* : la sección *Buscar uniones* describe todos los registros relacionados a través de los cuales puede crear uniones en sus búsquedas de este tipo de registro.
- *Filtros de búsqueda* : la sección *Filtros de búsqueda* describe todos los campos que están disponibles como un filtro de búsqueda para este tipo de registro. La ID interna cuando se utiliza un campo específico como un filtro de búsqueda *no siempre coincide con* su ID interna como un campo de cuerpo.
- *Buscar columnas* : la sección *Buscar columnas* describe todos los campos que están disponibles como una columna de búsqueda para este tipo de registro. La ID interna cuando se usa un campo específico como columna de búsqueda *no siempre coincide con* su ID interna como un campo de cuerpo.
- *Tipos de transformación* : la sección *Tipos de transformación* describe todos los tipos de registro en los que se puede transformar con el uso de la API de transformación de registro.

Encontrar un campo

Como se indicó anteriormente, no hay capacidad de búsqueda integrada en el *Navegador de registros* . Una vez que haya navegado al tipo de registro adecuado, si aún no conoce la ID interna de un campo en particular, la forma más fácil de encontrarlo es usar la función Buscar de su navegador (generalmente `CTRL+F`) para ubicar el campo por su nombre en la interfaz de usuario.

Campos requeridos

La columna *Requerido* del esquema indica si este campo es necesario para guardar el registro. Si esta columna dice `true` , entonces deberá proporcionar un valor para este campo al guardar cualquier registro de este tipo.

nlapiSubmitField y edición en línea

La columna `nlapiSubmitField` es una pieza crítica a comprender. Esta columna indica si el campo está disponible para la edición en línea. Si `nlapiSubmitField` es `true` , entonces el campo se puede editar en línea. Esto tiene un gran impacto en la forma en que se maneja este campo cuando se intenta utilizar las funciones `nlapiSubmitField` o `record.submitFields` en sus scripts.

Cuando esta columna es `true` , puede usar de forma segura las API de Campos de envío para actualizar este campo en línea. Cuando es `false` , *todavía puede usar estas funciones para actualizar el campo* , pero lo que realmente sucede detrás de escena cambia significativamente.

Cuando `nlapiSubmitField` es `false` para un campo en particular y utiliza una de las API de campos de envío en él, el motor de secuencias de comandos detrás de escena realmente hará una carga

completa del registro, actualizará el campo y enviará el cambio a la base de datos. El resultado final es el mismo, pero debido a que todo el registro se carga y se guarda, su script realmente usará mucho más gobierno de lo que podría esperar y demorará más en ejecutarse.

Puede leer sobre esto con más detalle en la página de Ayuda titulada "Consecuencias del uso de nlapiSubmitField en campos editables no en línea".

Lea Usando el navegador de registros NetSuite en línea:
https://riptutorial.com/es/netsuite/topic/7756/usando-el-navegador-de-registros-netsuite

Creditos

S. No	Capítulos	Contributors
1	Empezando con netsuite	4444, Community, erictgrubaugh, Kirk Lampert
2	Abastecimiento	erictgrubaugh
3	Buscar datos de registros relacionados	erictgrubaugh
4	Búsquedas con gran cantidad de resultados.	Slavi Slavov
5	Búsquedas de scripts con expresiones de filtro	Slavi Slavov
6	Cargando un registro	Adolfo Garza, Eidolon108, LittleZul, michoel, VicDid, YNK
7	Crear un registro	Deepa N, michoel, YNK, Zain Shaikh
8	Descripción general del tipo de script	erictgrubaugh
9	Edición en línea con SuiteScript	erictgrubaugh
10	Ejecutando una búsqueda	Adolfo Garza
11	Eliminar en masa	MG2016
12	Entender las búsquedas de transacciones	erictgrubaugh
13	Evento de usuario: antes del evento de carga	erictgrubaugh
14	Evento de usuario: antes y después de enviar eventos	erictgrubaugh

15	Explotando columnas de fórmulas en búsquedas guardadas	MetaEd
16	Gobernancia	erictgrubaugh
17	Registros de implementación de scripts y scripts	erictgrubaugh
18	RESTlet - Procesar documentos externos	MG2016
19	RestLet - Recuperar datos (Basic)	MG2016
20	Solicitando customField, customFieldList y customSearchJoin con PHP API Advanced Search	Hayden Thring
21	SS2.0 Suitelet Hello World	LittleZul
22	SuiteScript - Procesa datos desde Excel	MG2016
23	Trabajando con Sublistas	erictgrubaugh
24	Usando el navegador de registros NetSuite	erictgrubaugh