Fábrica - Modelos Logs - Log de Transacciones

Este es el log utilizado para registrar los logs en las transacciones de modificación de información de los procesos de las aplicaciones.

Tipos de Transacciones

A continuación se listan los tipos de transacciones que deben ser considerados en el log.

- Inserciones de datos.
- Actualizaciones de datos.
- Eliminación de datos.

¿Donde usar el Servicio?

Este servicio debe consumirse en los procesos de cargas de:

- Aplicaciones Java
- Aplicaciones .Net
- Web Service
- Aplicaciones Móbiles
- Aplicaciones Powerbuilder (Ver Nota siguiente)
- Soluciones que afecten los productos SICOF

Nota: Aplicaciones Powerbuilder

Powerbuilder tiene restricciones para el consumo de servicios Rest por lo tanto en las aplicaciones de esta tecnología se implementará un API para realizar las llamadas.

Diccionario de Datos

OWNER	SICOF	**TABLE**	TRANSACTIONAL_LOG	**COMMENTS**	Contiene el log de transacciones de las aplicaciones de la compañia
#	**NAME**	**NULLEABLE**	**TYPE**	**COMMENTS**	**WS**[1)]
1	ID	N	NUMBER	Código interno del registro (Se controla por secuencia)	Interno, Autoincremental

2	FECHA	Y	DATE	Fecha en la cuál se genera la transacción	No utilizado
3	NOMBRE_PROCESO	Y	VARCHAR2(256)	Nombre del proceso que realiza la transacción	Externo, Requerido
4	TIPO_TRANSACCION	Y	VARCHAR2(256)	Identifica el tipo de operación (Insert, Select, Update, Delete, Execute)	Externo, Requerido
5	LOG_TABLA	Y	VARCHAR2(256)	Nombre de la Tabla	Externo, Requerido
6	LOG_COLUMNA	Y	VARCHAR2(256)	Nombre de la columna	Obsoleto, Externo
7	LOG_VALOR_ANT	Y	VARCHAR2(4000)	Valor anterior de la columna	Obsoleto, Externo, Requerido
8	LOG_VALOR_ACT	Y	VARCHAR2(4000)	Valor actual de la columna	Obsoleto, Externo
9	LOG_PETICION	Y	CLOB	Se utiliza en caso de ser necesario para almacenar el bloque de la petición completa realizada	Externo, Requerido
10	LOGIN_USUARIO	Y	VARCHAR2(256)	Login del usuario que realiza el proceso	Externo, Requerido
11	HOST_CLIENTE	Y	VARCHAR2(50)	Host del cliente (Dirección IP)	Externo, Requerido
12	FECHA_REGISTRO	Y	DATE	Fecha del sistema DB	Interno, Formato dd/mm/yyyy hh:mm:ss, Requerido
13	CODIGO_USUARIO	Y	NUMBER	Código del usuario de la sesión en la cuál se genera el error	Externo, Requerido

14	CODIGO_MEMPRESA	Y	VARCHAR2(64)	Código de la empresa de la sesión en la cuál se genera el error	Externo, Requerido
15	CODIGO_APLICACION	Y	NUMBER	Código de la aplicación (Identificador interno numérico)	Externo, Requerido
16	INFO_APP	Y	VARCHAR2(256)	Información de la aplicación (En las situaciónes donde no se identifique código interno se puede enviar el nombre de la aplicación o información adicional)	Externo
17	SESSION_MAC	Y	VARCHAR2(64)	MAC del equipo del usuario	Externo
18	SESSION_BROWSERVERSION	Y	VARCHAR2(64)	Versión del Navegador	Externo, Requerido
19	SESSION_OSTYPE	Y	VARCHAR2(64)	Sistema Operativo	Externo, Requerido

Columna: WS

Se adiciona esta columna para identificar reglas asociadas a la implementación de los servicios web que permiten gestionar el almacenamiento de los logs. La columna es una referencia y no hace parte del servicio sin embargo las reglas que se definen en ella si aplican para la columna relacionada:

Reglas

- **Interno**: Indica que el campo se gestiona dentro del servicio y por lo tanto no se pedira en los parametros.
- **Autoincremental**: Indica que el campo se comporta como una secuencia.
- **Externo**: Indica que el campo debe estar en los parametros del consumo.
- **Requerido**: Indica que el campo debe ser enviado en el consumo y el servicio debe validarlo para continuar.
- **Obsoleto**: Indica que el campo ya no es utilizado en la nueva implementación.
- **No utilizado**: Indica que el campo no será utilizado en ninguna implementación.

Nota

- Todas las operaciones del servicio que gestiona la persistencia de la tabla deben estar documentadas incluyendo la definición de los campos, formatos, longitudes de columnas e indicar si es requerido o no.

Columna: LOG_PETICION

Esta columna sirve para almacenar la información de la peticion[2)] que se realiza en el registro que va a la base de datos. Se define la siguiente estructura base ejemplo:

```
{
  "columns":[
    {
      "column_name":"Requerido: Nombre de la columna",
      "column_old_value": "Valor anterior de la columna",
      "column_new_value": "Valor actual de la columna"
    }
  ]
}
```

Donde:

- **columns**: Arrya de Columnas en la transacción
- **column_name**: Propiedad contenida en cada indice del array json que representa el nombre de la columna en la transacción.
- **column_old_value**: Propiedad contenida en cada indice del array json que representa el valor anterior de la columna en la transacción (Requerida para operaciones Update, Delete).
- **column_new_value**: Propiedad contenida en cada indice del array json que representa el valor actual de la columna en la transacción (Requerida para operaciones Insert, Update).

Ejemplo:

```
{
  "columns":[
    {
      "column_name":"codigo_tercero",
      "column_old_value": "1234",
      "column_new_value": "3456"
    },
    {
      "column_name":"valor_debito",
      "column_old_value": "0",
      "column_new_value": "100000"
    }
  ]
}
```

Notas

- Los valores de las columnas deben ser registrados como String

Modo de uso: Powerbuilder - Documentación

Para visualizar la documentación debe descargar el siguiente repositorio Documentación, abrir la pagina Index.html en su navegador web la cual es similar a la siguiente imagen:

En ella encontrará la documentación de las librerias que hacen parte del framework **Objetos SICOF** el cuál se irá actualizando frecuentmente a medida que se documenten las clases.

La Libreria que contiene la funcionalidad de los logs es la librería **sf00util.pbl**

Los Objetos relacionados en el API son:

- **n_cst_app**: Clase contenedora de objetos logs
- **n_cst_log_transactional**: Clase para la gestión de log de transacciones

Ejemplos de Uso

Para facilitar la implementación y uso del API de gestión de log de transacciones se crea un objeto interno privado en la clase global **guo_app** el cual puede ser accedido por el método **of_log_transactional()** que devuelve la instancia del objeto. Sin embargo para implementaciones específicas se puede optar por crear y administrar la clase **n_cst_log_transactional** según considere el desarrollador.

A continuación se listan ejemplos de uso el cuál presenta las forma de utilizar el API, para más información debe consultar la documentación en el repositorio.

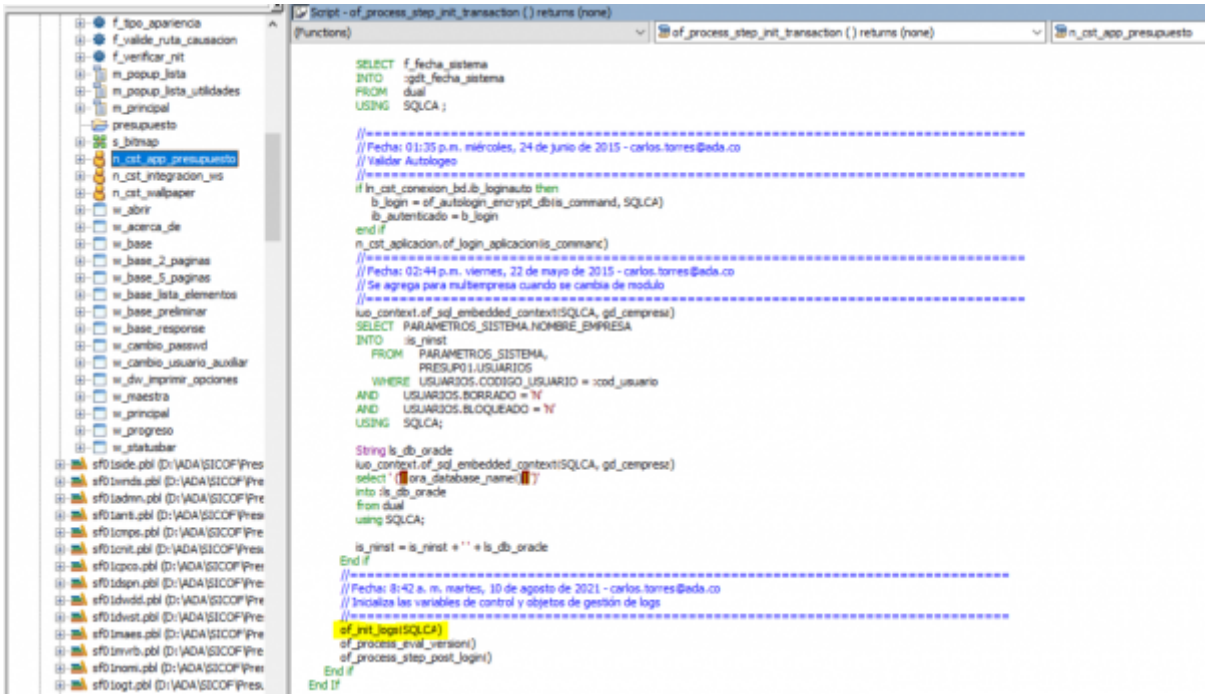
```
/*Ejemplos de uso utilizando la instancia genérica de la clase guo_app*/
guo_app.of_log_transactional( ).of_add_log(f_hoy(), "Causación por
```

```
Plantilla", "Update", "PRESUP01.MAESTRO_ASIENTO_CONTABLE", "codigo_tercero",
"0", "6949", true, SQLCA)
guo_app.of_log_transactional( ).of_add_log(f_hoy(), "Causación por
Plantilla", "Update", "PRESUP01.MAESTRO_ASIENTO_CONTABLE", ljson, true,
SQLCA)

/*Ejemplo de uso definiendo la clase*/
n_cst_log_transactional luo_log_transactional
luo_log_transactional = Create n_cst_log_transactional
luo_log_transactional.of_add_log(f_hoy(), "Causación por Plantilla",
"Update", "PRESUP01.MAESTRO_ASIENTO_CONTABLE", ljson, true, SQLCA)
destroy luo_log_transactional
```

Consideraciones

- El API puede ser activada o desactivada por medio de la constante: **LOG_TRANSACCIONAL** (Solo en aplicaciones SICOF ERP (Appeon/Powerbuilder)) siempre y cuando se utilice la implementación de la clase **guo_app**.
- El desarrollador es el encargado de gestionar la transacción que realiza la persistencia.
- Para el procesamiento de logs de bloques se implementa Clase sailjson para procesamiento de cadenas de texto en ese formato.
- Cada módulo (Contabilidad, Prespuesto, Tesorería, Compras, Talento y Nómina) debe implementar el método de inicialización **guo_app.of_init_logs(SQLCA)** en el método **of_process_step_init_transaction** de la clase **guo_app** especializada por cada módulo. A continuación se muestra una imagen de referencia de la implementación del módulo de presupuesto. Utilice esta guía para implementaciones en otros módulos teniendo presente que la clase **n_cst_app** se especializa con el nombre de la aplicación que la contiene. Ejemplo: en presupuesto la clase especializada es **n_cst_app_presupuesto**, por lo general la clase esta en la libreria principal que contiene el objeto **Application**.



API Json

Se adicionan métodos de procesamiento de bloques en formato json los cuales pueden ser utilizados para registrar trazas de error a continuación se muestran ejemplos de uso del API.

```
sailjson ljson, ljson1
String ls_content

ljson = create sailjson
ljson.setattribute( 'version', '1001')
//add json object
ljson1 = ljson.addobject( 'header')
ljson1.setattribute( 'count', 3)
ljson1.setattribute( 'comment', 'items count')

//add json object array, first item
ljson1 = ljson.addarrayitem( 'data')
ljson1.setattribute( 'colid', 1)
ljson1.setattribute( 'colname', 'aaaaaa')
ljson1.setattribute( 'coladdr', '')
//add second item of the array
ljson1 = ljson.addarrayitem( 'data')
ljson1.setattribute( 'colid', 2)
ljson1.setattribute( 'colname', 'bbbbbbbb')
setnull(ls)
ljson1.setattribute( 'coladdr', ls)
//add third item of the array
ljson1 = ljson.addarrayitem( 'data')
ljson1.setattribute( 'colid', 3)
ljson1.setattribute( 'colname', 'cccccc')

ljson.setattribute( 'creattime', string(now(), 'yyyymmdd.hhmmss'))

ls_content =  ljson.getformatjson('' )
destroy ljson
```

Limitaciones

Se identifica que actualmente la versión de Appeon (2013/2016) presenta problemas con el procesamiento del API Json para estructuras complejas como Arrays por esta razon cada procesamiento del API debe ser realizado a un solo nivel de complejidad por cada registro que realice operaciones DML. A continuacion se muestra un ejemplo de uso funcional en Appeon/Powerbuilder que puede ser tomado como referencia.

```
/*PASO 1: Definir un objeto de la clase sailjson en la zona de instancia del componente donde se va utilizar el API como una ventana, objeto no visual, etc*/
sailjson iuo_json
```

```
/*PASO 2: Incializar el objeto iuo_json antes de utilizarlo*/
iuo_json= create sailjson
```

```
/*
PASO 3: No es obligatorio, pero se recomienda crear un método local de
asignación para simplificar el proceso como ejemplo se define metodo
of_add_log_transactional con los siguientes argumentos
string as_dataobject
string as_column
string as_old_value
string as_new_value
*/

//Validaciones previas de control
if IsNull(as_dataobject) then return
if IsNull(as_column) then return
if IsNull(as_old_value) then as_old_value = ""
if IsNull(as_new_value) then as_new_value = ""

//Crear propiedad de la columna con el valor anterior
iuo_json.setattribute(as_dataobject + "." + as_column + "_OldValue",
as_old_value)

//Crear propiedad de la columna con el nuevo valor
iuo_json.setattribute(as_dataobject + "." + as_column + "_NewValue",
as_new_value)
```

```
//En el metodo de guardado del proceso consumir el API de Log de
transacciones con el objeto json
guo_app.of_log_transactional( ).of_add_log(Date(ldt_fecha),
"w_conexion_cliente_fe::guardar", "update", 'TBL_FE_CONEXION_CLIENTE',
iuo_json, true, ts_transaccion)

//Por último se debe reiniciar el componente para otro proceso
iuo_json = Create sailjson
```

```
//En el evento itemchanged del datawindow consumir el método local
of_add_log_transactional
String ls_column, ls_old_data
ls_column = dwo.name
Choose Case ls_column
  Case 'codigo_cliente'
    ls_old_data = getitemstring(row, ls_column)
    of_add_log_transactional(dataobject, ls_column, ls_old_data, data)
End Choose
```

Modo de uso: Java

Para las aplicaciones desarrolladas en las tecnologías (Web):

- Java
- .Net
- PHP

el log de sesión será implementado por medio de un Servicio Web el cual deberá considerar las reglas de Columna: WS

←Volver atras

1)
Define las reglas que debe aplicar el Web Service
2)
Valores anteriores y Actuales según corresponda