Table Events

Servoy provides an event model at the data layer, giving developers the opportunity to implement validation and execute business logic just before and after data changes are committed to the database. There are twelve *Table Events,* each of which may be bound to an entity or a global method.

The first three events occur just prior to the change being committed to the database. Moreover, the event handler has the opportunity to veto the event, preventing the change from being committed. This is and ideal location to implement fail-safe data rules.

- **onRecordInsert** - occurs prior to a new record being inserted into the table
- **onRecordUpdate** - occurs prior to an existing record being updated in the database
- **onRecordDelete** - occurs prior to an existing record being deleted from the database

An *onRecordXXX* event is bound to an entity or a global method, which is invoked when the event occurs. The record that is modified is passed in as an argument and the method can veto the change by returning false or throwing an exception.

Parameters

JSRecord - the record object that is to be inserted, updated or deleted

Returns

Boolean - Return true from this method to allow the change to commit to the database. Returning false will result in not allowing to process the change.

> Throwing an exception in the event method will result in a Servoy Exception with a SAVE FAILED error code, which may be handled in the solution's onError event handler.

Example

This is an example of an *onRecordDelete* handler for an *invoices* table. The data rule is that posted invoices will never be deleted by the application.

```
/**
 * Record pre-delete trigger.
 * Validate the record to be deleted.
 * When false is returned the record will not be deleted in the database.
 * When an exception is thrown the record will also not be deleted in the
database but it will be added to databaseManager.getFailedRecords(),
 * the thrown exception can be retrieved via record.exception.getValue().
 *
 * @param {JSRecord} record record that will be deleted
 * @returns {Boolean} true to allow a delete
 * @properties={typeid:24,uuid:"A3F02F99-B899-46BE-9125-66E4189F043F"}
 */
function onRecordDeleteInvoice(record) {

        if(record.is_posted)
                throw "Cannot delete a posted invoice";

        return true;
}
```

The next three events occur immediately following the commit to the database. This is an ideal mechanism to update the data model after data is known to have changed.

- **afterRecordInsert** - occurs subsequent to a new record being inserted into the table
- **afterRecordUpdate** - occurs subsequent to an existing record being updated in the database
- **afterRecordDelete** - occurs subsequent to an existing record being deleted from the database

An *afterRecordXXX* event is bound to an entity or a global method, which is invoked when the event occurs.

Parameters

JSRecord - the record object that was recently inserted, updated or deleted

Example

This is an example of an *afterRecordInsert* handler for a *projects* table. The data rule is that a new project record will be linked, via the *projects_users* table, to the current user.

```
/**
 * Record after-insert trigger.
 *
 * @param {JSRecord} record record that is inserted
 *
 * @properties={typeid:24,uuid:"92834B20-1CAC-472F-B022-DD97FEFEA792"}
 */
function afterRecordInsert(record) {
	if(record.projects_to_projects_users.newRecord())
{ 			// create a link record
		record.projects_to_projects_users.user_id = globals.
currentUserID;	// associate to current user
		databaseManager.saveData();
	}
}
```

The next three events occur just prior to the operation intended to be done on the foundset. The event handler has the opportunity to prevent the operation to take place. This is an ideal place to set fail-safe data rules.

- **onFoundSetRecordCreate** - occurs prior to a new record being created in the foundset
- **onFoundSetFind** - occurs prior to the foundset going into find mode
- **onFoundSetSearch** - occurs prior to executing a search on the foundset

An *onFoundSetXXX* event is bound to an entity or a global method, which is invoked when the event occurs. The method can veto the change by returning false or throwing an exception.

Parameters

The *onFoundSetSearch* event receives two parameters:

- clearLastResults - Boolean which tells whether or not to clear previous search
- reduceSearch - Boolean which tells to reduce (true) or extend (false) previous search results

Returns

Boolean - Return true from this method to allow the intended operation to take place on the foundset. Returning false will result in not allowing to process the operation.

> Throwing an exception in the event method will result in a Servoy Exception with a SAVE FAILED error code, which may be handled in the solution's onError event handler.

Example

This is an example of an *onFoundSetFind* handler. The data rule is that the foundset doesn't go into find mode if there are no records in the table.

```
/**
 * Foundset pre-find trigger.
 * When false is returned the foundset will not go into find mode.
 *
 * @returns {Boolean}
 *
 * @properties={typeid:24,uuid:"9FE55B79-E5D1-4B7D-82CD-D0E8CF6B6725"}
 */
function onFoundSetFind() {
    if(record_count == 0) // record_count is an aggregation defined for the
table, which counts the records
        return false;
    return true;
}
```

The last three events occur immediately following the operation executed on the foundset.

- **afterFoundSetRecordCreate** - occurs subsequent to the creation of a new record
- **afterFoundSetFind** - occurs subsequent to entering find mode
- **afterFoundSetSearch** - occurs subsequent to performing the search for a foundset

An *afterFoundSetXXX* event is bound to an entity or a global method, which is invoked when the event occurs.

Parameters

The *afterFoundSetRecordCreate* event receives parmeter JSRecord which is the record object that was recently crated.

Example

This is an example of an *afterFoundSetRecordCreate* handler for *book_text* table. The data rule is that every new record added to the foundset will have a predefined text on the *comment_text* column.

```
/**
 * Record after-create trigger.
 *
 * @param {JSRecord<db:/example_data/book_text>} record record that is created
 *
 * @properties={typeid:24,uuid:"CCDA9A02-7E4D-4A82-9815-B030BBDB1ED0"}
 */
function afterFoundSetRecordCreate(record) {
    record.comment_text = "Some predefined comment text.\n"
}
```