Infor Enterprise Server User Guide for Triggering

Important Notices

Trademark Acknowledgements

Publication Information

Document code	tttriggeringug (U8762)
Release	10.5 (10.5)
Publication date	February 6, 2024

Table of Contents

About this document

This document describes the setup and use of the Triggering (TRG) module in the Enterprise Server Data Director (DA) package.

About this Guide

This document is a User's Guide that describes the setup and use of the Triggering (TRG) module in the Infor Enterprise Server Data Director (da) package.

This document contains the following chapters:

- Chapter 1, "Overview," provides an overview of the triggering solution.
- Chapter 2, "To set up triggering," describes how to set up the triggering.
- Chapter 3, "Event handling," describes what events look like and how a user can handle events.
- Chapter 4, "Examples," provides an example of how you can use the solution for a workflow business case.
- Appendix A, "Triggering sessions," provides and overview of the sessions of the Triggering (TRG) module.
- Appendix B, "Exchange sessions," provides and overview of the Exchange (daxch) sessions that are used in the Triggering solution.
- Appendix C, "External interface," provides a brief description of the Application Programming Interface (API) used for Triggering.

Note: This document describes the functionality of the Triggering module. This document does not provide a detailed description of the functionality of the Triggering sessions. For detailed information on these sessions, refer to the **Infor Web Help**.

Definitions, acronyms, and abbreviations

Term	Description
API	Application Programming Interface
Audit trail	A log that lists the changes that took place on the ERP database
BOI	Business Object Interface: an interface to invoke ERP business logic from outside ERP
da	Package code for Infor Enterprise Server Data Director
DLL	Dynamic Link Library

ERP	If used without a release number in this context, refers to the Infor LN enterprise resource planning product
Exchange	An ERP module to export and import data or data changes
OLTP	Online transaction processing: In this context, all transaction processing by users or application processes that result in database changes
TRG	Module code for the Triggering module from the Data Director (da) package
UDA	User Defined Attribute: In workflow, an attribute that is used in a process. Usually a primary key value to identify the business object instance to be used in a process.
XCH	Module code for the Exchange module from the Data Director (da) package.
XML	eXtensible Markup Language
XSD	XML Schema Definition

Other documentation

- The Web Help for the Exchange (XCH) and Triggering (TRG) modules. The Help consists of session Help and online manual topics.
- The specifications (dllusage) of the following libraries:
  - datrgevent (Event API)
  - datrgapi (Triggering API)

Note: You can retrieve the library specifications from the library objects. At operating-system level, use explode6.2 to locate the library object and subsequently use bic_info6.2 with the **-eu** options.

For example:

```
$ explode6.2 odatrgevent
/mybse/application/myvrc/odatrg/otrgevent

$ bic_info6.2 -eu /mybse/application/myvrc/odatrg/otrgevent
```

This shows the documentation of the event handling functions.

This method provides the prototypes, which includes the function name and type and parameters and their types, of the functions in the library, a description of the functions and their input and output, and the preconditions and post-conditions.

Send us your comments

We continually review and improve our documentation. Any remarks/requests for information concerning this document or topic are appreciated. Please e-mail your comments to documentation@infor.com

In your e-mail, refer to the document number and title. More specific information will enable us to process feedback efficiently.

Comments?

We continually review and improve our documentation. Any remarks/requests for information concerning this document or topic are appreciated. Please e-mail your comments to documentation@infor.com .

In your e-mail, refer to the document number and title. More specific information will enable us to process feedback efficiently.

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

If we update this document after the product release, we will post the new version on the Infor Support Portal. To access documentation, select **Search Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com .

Chapter 1
Overview

1

Overview

You can use Triggering if another site or application must be notified when an event occurs in LN.

For that purpose, the Triggering (TRG) module is available in the Data Director (da) package.

This module is a small component in LN that does the following:

- Receives an event.
- Checks whether the event meets certain conditions.
- Takes a predefined action.

Important terms

Term	Description
Event	An event is a message that states that something happened. An event can for example be the creation of a new sales order, the change of a price, or the receipt of a payment. In addition to the type of event, the event also contains data related to the event, such as the identifier of the object on which the event occurred, for example, an order number, and the data that was changed. For more information on events, refer to Chapter 3, "Event handling."
Condition	Conditions are used because not all events are valid. For example, notification of a newly created order might only be required if the order is entered

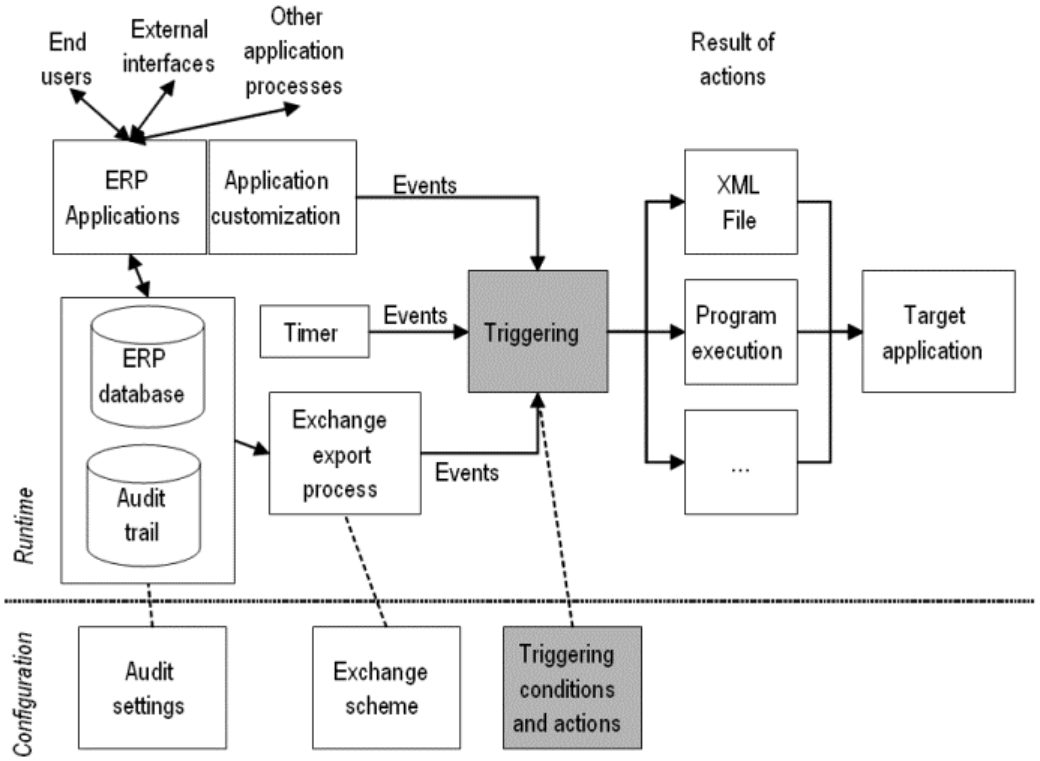
	manually. Notification of a change might only be required if a particular status is reached.
Action	The action describes what you must do if an event occurs. This can, for example, be the execution of a program or the creation of an XML file that the receiving application can pick up.

Origin of events

An event can be generated from multiple sources, such as the following:

- LN sessions can create events: This requires a customization of these sessions. This type of customization can be either visible to the user, for example, by adding a button to a session, or invisible, for example, by adding lines to a program script that creates an event if a specific situation occurs.
- Change events for created, changed, or deleted objects can be created from the Exchange module. In case of deleted objects, an exchange job runs regularly to collect changes on LN data and creates an event for each of these changes. The changes are collected from the audit trail.
- You can use the triggering mechanism to synchronize regularly complete data sets, because the Exchange module can regularly read a specific set of data and create the corresponding events that describe the current status of the data. You can use this method to notify another application or site from the current status of the data set, if the data set is small. For this reason, keeping track of the changes is not worthwhile.
- You can generate events using a timer. As a result, an event occurs regularly, irrespective of the LN application or LN data. In that case, the action must specify what must happen.

The following figure provides an overview of the solution.



Solution overview

Runtime actions

The following example illustrates what happens at runtime in the exchange-based scenario:

1. Changes are made to the OLTP (online transaction processing) database by users or by other processes that result in changes.
2. Changes on relevant tables are logged in the audit trail.
3. The exchange process runs regularly in a job.
4. The exchange process picks up the changes from the audit trail and processes the changes according to the defined scheme.
5. The exchange scheme creates XML events for the changes and forwards these to the triggering component.
6. The triggering component receives the event, checks what action must be carried out upon that event and performs this action.

Relation with Workflow

You can use the Triggering module to trigger business processes in Workflow. In that case, you can use the Workflow Management (tgwfm) sessions to generate the Exchange schemas and triggers.

For details, refer to the Workflow online Help and to the User's Manuals for Workflow for LN.

More information

For information on the scripts that are used in the triggering module, refer to the following topics in the Web Help:

- To configure an attached condition script
- To configure an attached action script
- Transformation script

Chapter 2
To set up triggering

2

To set up triggering

Preparations

Before you set up triggering, you must specify the requirements/design. Consider what must happen and when. For example, if a new sales order is created in LN, application X must be notified. Or, if a specific status field receives the value **Active**, application Y must be notified.

In addition, you must make clear what information must be communicated. In other words, what must be the content of the event message? For example, only an order number or other attributes from an order such as an order date or Sold-to Business Partner (customer) ID.

Choose how to implement the triggering, for example, by means of Exchange or by means of an application customization. In addition, you must define the action to be carried out.

While you specify these settings, you must consider performance-related questions. For example, how long must the process take before the target application is notified of an event? How frequently will events occur?

Subsequently, you can perform the actual implementation.

Overview of the actual implementation

The process to implement triggering consists of the following steps:

1. Configure a trigger source, such as an exchange scheme or an application customization. For more information, refer to "To set up a trigger source," later in this chapter.
2. Define the trigger, including the conditions and action. For more information, refer to "To define a trigger," later in this chapter.
3. Test the trigger: start the required processes and manage the processes at runtime. For more information, refer to "Runtime tasks," later in this chapter.

To set up a trigger source

Trigger sources

As described previously, you can generate an event in multiple ways, including the following:

- Application-based: Customize LN to generate trigger events.
- Exchange-based: Use the Exchange module to collect the changed database rows or a complete set of database rows from one or more tables.
- Timer-based: Generate an event regularly according to a time interval or calendar.

Application-based triggering

In the application, add a piece of code that generates an event message and invokes a trigger. You can attach this code, for example, to a library or to a program script.

The following two APIs are available to attach this code:

- API functions to create events are available in the *datrgevent* library.
- The API function to invoke a trigger is available in the *datrgapi* library.

For detailed information, refer to the specifications of these libraries.

Note

You can retrieve the library specifications from the library objects. At operating-system level, use explode6.2 to locate the library object, and subsequently use bic_info6.2 with the **-eu** options.

For example:

```
$ explode6.2 odatrgevent
```

```
/mybse/application/myvrc/odatrg/otrgevent
```

```
$ bic_info6.2 -eu /mybse/application/myvrc/odatrg/otrgevent
```

This command shows the documentation of the event handling functions.

This method provides the prototypes, including the function name and type and parameters and their types, of the functions in the library, a description of the functions and their input and output, and the preconditions and post-conditions.

The following is an example that shows how you can create an event from an application customization and to execute a trigger:

```
#pragma used dll odatrgevent    |event handling functions
#pragma used dll odatrgapi  |triggering api

#define CHECK_RETURN(retval)    if retval <> 0 then
^                                 ... |implementation depending|on context
^                               endif

long    my.event                |XML event used for triggering
long    retl                    |return value to be checked
```

```
string  exception.message(512) mb   |message if an error occurs
string  exception.details(512) mb   |details on exception message

if curr.inventory.on.hand < threshold.value and prev.inventory.on.hand >= threshold.value
then
    |low inventory, invoke trigger
    retl = datrgevent.simpleevent.create("Inventory", "LowInventory", my.event)
    CHECK_RETURN(retl)
    retl = datrgevent.simpleevent.set.value(my.event, "item", curr.item)
    CHECK_RETURN(retl)
    retl = datrgevent.simpleevent.set.value(my.event, "warehouse", curr.warehouse)
    CHECK_RETURN(retl)
    retl = datrgevent.simpleevent.set.value(my.event, "inventoryOnHand",
                        curr.inventory.on.hand)
    CHECK_RETURN(retl)
    retl = datrgevent.simpleevent.set.old.value(my.event, "inventoryOnHand",
                        prev.inventory.on.hand)
    CHECK_RETURN(retl)
    retl = datrgapi.trigger.do("mytrigger", my.event, exception.message, exception.details)
    CHECK_RETURN(retl)
endif
```

The simple event functions are offered to create events that consist of only one component. Specifying a class name (entity), event type (action), and the attribute name and value for each attribute is sufficient. In addition, if desired, you can also add the previous attribute value. Other event functions are available to create more complex events, such as multilevel events that consist of header and line components. For more information, refer to Chapter 3, "Event handling."

Exchange-based triggering (changes)

To use Exchange-based triggering, you must define an exchange scheme, including ASCII file and fields, batch, table relations (export), and field relations (export). In case of triggering, the ASCII file and fields might not be relevant in the sense that no physical ASCII files are created. However, these files and fields are relevant because they define the contents of the event. The ASCII file represents the object (or component) and the fields represent the attributes included in the event.

The mapping of the event, and the event’s required attributes, to the table and columns must be made clear. If a one-to-one mapping exists between columns and required attributes for the trigger, the exchange scheme can largely be generated as described in the following procedure. If required, you can add constant or calculated values to the output. Note that for calculated/transformed values, scripting is required.

You can easily set up an exchange scheme in the following way:

1. In the Exchange Schemes (daxch0501m000) session, create the main entity and define the exchange scheme attributes.
2. In the Exchange Schemes (daxch0501m000) session, on the **Specific** menu, click **ASCII Files** to start the ASCII Files (daxch0102m000) session.
3. While you use a table code for the ASCII file, create a new ASCII file. You can leave the **Definition File** field empty. If multiple tables are required, create multiple entries.
4. On the **Specific** menu, click **Create ASCII Files...** to start the Create ASCII File Fields and Relations (daxch0203m000) session.

5. Choose the required range of ASCII files.
6. Make sure the following check boxes are selected:
   - **Create Based on Table Definitions**
   - **Create Batch**. Enter the code and description of the batch to be created in the fields that correspond to the check box.
   - **Create Export Relations**
7. Start the process, which now generates the required exchange scheme contents.
8. Remove any unwanted fields by deleting the corresponding rows from the **Field Relations (Export)** session and from the ASCII File Fields (daxch0503m000) session.

After the exchange setup is complete, you can run the Create Export Programs (daxch0228m000) session to create a runtime program that implements the settings as defined in exchange scheme.

For more information on Exchange, refer to the Web Help.

The exchange scheme must be based on audit. Therefore you must generate an audit configuration for the tables in the exchange scheme. Use the Generate Audit Configuration (daxch1201m000) session for this. For more information on Audit Management, refer to the Web Help.

After you set up the exchange scheme and the triggers, you must define the relation between the two. In other words, one or more triggers must be linked to the exchange scheme. To link these triggers, you must use the Export Triggers (daxch0135m000). For details on this session, refer to the Web Help.

Exchange-based triggering (full export)

If a complete data set must be published regardless of what data was created, deleted, or changed, you can use a full exchange export.

In this case, the setup is the same as described in "Exchange-based triggering (changes)," later in this chapter, with the following exceptions:

- The exchange scheme must not be based on audit.
- No audit setup is required.

Timer-based triggering

No specific setup is required for timer-based triggering. You must define the trigger as described in "To define a trigger," later in this chapter. Subsequently, you can start the process as described in "Runtime tasks," later in this chapter.

To define a trigger

In the Triggering module, you must define the trigger, including the trigger’s conditions and actions, as required. You can create triggers in the Triggers (datrg1100m000) session. This session contains a configuration interface that enables you to generate the condition and action logic. Note that you can use the same trigger for multiple types of events, and even from multiple sources.

If applicable, you can define conditions for the trigger. A condition limits the number of events for which a trigger action is performed. You can define conditions by means of the Configure Trigger Conditions (datrg1110m000) session. In this session, you can create or update conditions on class, event type, or attribute value.

If the functionality available in this session is insufficient for the condition you want to create, you can attach a script that contains complex conditions. For details, refer to the "Configure attached condition script" online manual topic.

In addition, you must define an action for the trigger. The type of action that you must define depends on the action template that you select for the trigger:

- A *Fan Out Action* Use this action template if the trigger must invoke multiple actions (subsequent triggers) in parallel.
- An *XML File Action* Contains the settings in case the trigger must act by creating a file containing the event.
- If the user interface functionality is insufficient to meet the requirements, the user can implement a specific configuration by attaching code (a script) that implements the trigger action. For details, refer to the "Configure attached action script" online manual topic.

If you define a trigger, the runtime program is automatically generated. The program is also regenerated automatically when you change the trigger, or the trigger's conditions or action. In addition, you can click **Generate Program** in the Triggers (datrg1100m000) session to regenerate the runtime program.

Runtime tasks

At runtime, the following happens: The triggering module offers an interface to initiate a trigger. Input in that case is an XML structure describing the event. The user can call this interface from the Exchange module, from an application, or from the Start Triggering via Job Timer (datrg1200m000) session. If you do not invoke this interface from an application or exchange process, the process checks whether an action is defined for the specified trigger and runs the corresponding trigger function.

Trigger management

For the trigger itself: no further action is required. The trigger is passive. Therefore, the trigger simply waits until one of the trigger sources invokes the trigger. If the trigger does not act as desired, you can debug the trigger. For more information, refer to the Triggers (datrg1100m000) session online Help.

Note

If a trigger does not exist, this will not be reported as an error. For example, assume that an application can run in multiple companies and only from one company must something be done upon a trigger. In this case, the same application can run if the trigger is only defined in one company.

Application-based triggering

For application-based triggering, no action is required at runtime either. The trigger invocation is included in the application logic, therefore, the trigger invocation will be performed automatically when the corresponding application state is reached.

Exchange-based triggering

To use exchange-based triggering, the export process must be running regularly. You can add the Export Data (on a Regular Basis) (daxch0234m000) session to a job that runs according to a calendar or a predefined time interval.

This job ensures that the exchange scheme regularly picks up the changes from the audit trail or the data from the database tables and invokes the corresponding triggers. For details, refer to the Exchange online Help.

Bear in mind that in practice, the time interval must not be shorter than the time the export process requires to be completed successfully. If a small interval, such as one minute or a few minutes, is required, check beforehand whether the selected interval is feasible. Note that the duration of the export process depends heavily on the amount of changes or data to be processed and the trigger action defined.

You must run all normal management activities required to run an exchange scheme. For example, occasionally, you might want to clean up data produced by the export process. Exchange offers logging facilities to check whether the process is running successfully.

If you use the audit mechanism to detect change events, the audit trail must be cleaned up regularly, if the data is no longer required.

Finally, note that the audit setup impacts performance. If the system sizing is correct and the audit is configured properly, the impact will be minimal, because the overhead is usually not greater than a few percent. Locating the audit data on a disk that is not a potential bottleneck is strongly advised. In addition, you can deploy disk striping to improve performance.

Timer-based triggering

Run the Start Triggering via Job Timer (datrg1200m000) session to regularly generate a predefined event. Refer to the Web Help for details.

Target application

Finally, the application that receives the event will require attention. For example, this application must clean up received files after processing.

Exchange-specific limitations

The following limitations apply to Exchange-based triggering.

Table vs. business object level

If you use events from Exchange, the triggers will be defined at table level rather than at Business Object level. Information from other tables can be included (scripting options are available), however, this requires programming. Note, however, that if the primary key of the object is the only attribute that is required, the same event can be generated from multiple tables. For example, a change on an order header and a change on an order line can both result in the same event, if required.

An example of an event that cannot be defined without programming a so-called condition script in the exchange scheme is as follows: "Initiate an approval process if the quantity of an order line is changed and the status of the order header is Open".

Unchanged table fields

If a row is created in or deleted from a table, the event generated by the Exchange export includes all ASCII fields as defined. However, if a row is changed, by default only the primary key fields and the changed fields will be included.

For example, if you handle events on or order line consisting of an order number, line number, item, quantity and price, if the price is changed, the item and quantity will not be available in the event. You must take this into account when you define the trigger and when you process the event in the application that finally receives trigger. A trigger condition on quantity will only be met if the quantity is available; in other words, if the row is created or deleted, or if the quantity value is changed.

To change this default behavior, you must change the audit type of the table fields in your exchange scheme’s audit profile.

Two audit types exist:

- **Always**: The field is logged each time when the content of the field, or the content of any other audited field, changes. This is the default setting for all primary key fields.
- **Changed**: The field is logged only when the content of the field itself changes. This is the default setting for all fields that do not belong to the primary key.

To make sure that a table field is always logged in the audit trail, you must change the field’s audit type to "Always". To do so, take the following steps:

1. Start the Audit Profiles (ttaud3110m000) session and select the audit profile that belongs to your exchange schema (the audit profile that you generated through the Generate Audit Configuration (daxch1201m000) session).
2. Click **Table Settings by Profile** on the **Specific** menu. The Audit Tables by Profile (ttaud3120m000) session is started. A list of tables is displayed.
3. Select the table for which you want to change the audit settings, and click **Audit Fields by Table** on the **Specific** menu. The Audit Fields by Table (ttaud3125m000) session is started. In this session you can change the audit type per table field. You can select the desired audit type (for example,. "Always") from the list.

4. Run the Create Runtime Audit Definitions (ttaud3200s000) session to generate new runtime audit definitions. You can start this session via the **Specific** menu in the Audit Profiles (ttaud3110m000) session.
5. Restart your virtual machine (bshell).

Note

For details about Audit Management, refer to the Web Help and to the *Infor10 ERP Enterprise Server (LN) Technical Manual (U8172 US)*.

Event types

Only events that can be defined in terms of a change in persistent data can be handled. The event must be described in terms of the following:

- Tables
- Event type (action):
  - Create
  - Change
  - Delete
- Attribute values.

Examples of events that can be generated from Exchange based on audit include the following:

- A new Item row is added.
- The quantity of an item row changes.
- The quantity of an item row is increased. Note that you cannot configure this condition directly in the triggering user interface. However, to configure this condition, you can customize the generated triggering implementation function, as described in the "Configure attached condition script" online manual topic.
- An item row with item group Hardware is created, changed, or deleted. However, as described previously, if an item row is changed, but the item group is not changed, the event will not meet the condition.

Examples of events that *cannot* be generated from Exchange include the following:

- An item row is changed by means of the **abcde1234m000** session
- An item row is change by user *john*.

Miscellaneous Exchange-specific limitations

- From Exchange, no (multiline) texts are included in the XML event. You can include text numbers.
- Events from Exchange use the ASCII file/field names as component/attribute names. This implies that the events will have a maximum of eight characters.

- Dates will be formatted according to the date format specified in the Exchange scheme properties in the Exchange Schemes (daxch0501m000) session, and/or ASCII file format properties in the ASCII File Fields (daxch0503m000) session.
- For enumerated constants, you must use the constant's numeric value instead of the constant's name. For example, in a trigger condition, you must use tdsls400.corg = 3 instead of tdsls400.corg = tdsls.corg.eop.
- No multibyte conversion is performed on multibyte string values stored in the XML event.

Chapter 3
Event handling

3

Event handling

Introduction

Event structure

An event essentially consists of the following two parts:

- Control area: Specifies the event type, or event action, and can include other elements such as the class, or entity, of the object that was changed, the event time, or the application that generated the event. Standard event types include:
  - Create: A new object instance was added.
  - Change: An object instance was updated.
  - Delete: An object instance was removed.
- The data area contains the data of the (business) object that was impacted by the event. The data area also contains *components* and *attributes*.

If the data area is filled, which is not mandatory, one component will always have one or more attributes. In addition, this component can have child components. A typical example is a sales order object, in which the main component is the sales order and the child components are the order lines.

The sales order component will have attributes such as order number, order date, and Sold-to business partner (customer). The order line component will have attributes such as line number, item, quantity, and price.

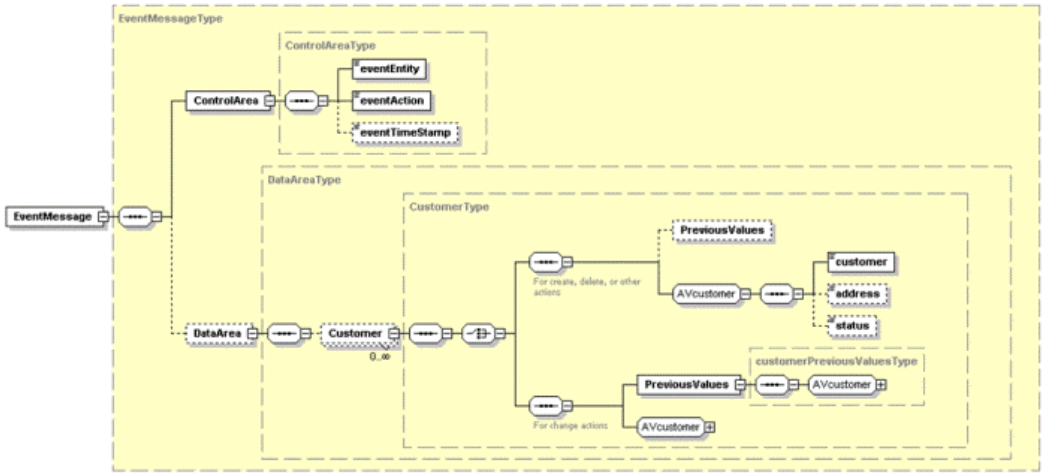
Attributes can have old values, new (current) values, or both. Create events typically do not have old values, while delete events do not have new values. Change events can have either old or new values.

Important!

Although the logical structure of the events is stable, the technical format can change. The XSDs and XMLs in this chapter are included to illustrate the logical structure. For best results, use the event handling API, as described in "API for event handling" later in this chapter, when you use the contents of an event, because that interface must not change the event even though the XML structure can change.

Example

The following figure illustrates the event used for a change, in this case, on a Customer object. Note that the component name, Customer, and the attributes are specific for this object. Other objects will have other values and can also have child components. Note, however, that events from Exchange will always be simple events, which are events that consist of only one component.



XSD for change event

An XSD file for this structure contains the following code:

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.baan.com" xmlns="http://www.baan.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
    <xs:element name="EventMessage" type="EventMessageType"/>
    <xs:complexType name="EventMessageType">
        <xs:sequence>
            <xs:element name="ControlArea" type="ControlAreaType"/>
            <xs:element name="DataArea" type="DataAreaType"
                minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ControlAreaType">
        <xs:sequence>
            <xs:element name="eventEntity" type="eventEntityDT"/>
            <xs:element name="eventAction" type="eventActionDT"/>
            <xs:element name="eventTimeStamp" type="eventTimeStampDT"
                minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="eventEntityDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="eventActionDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="eventTimeStampDT">
        <xs:restriction base="xs:dateTime"/>
    </xs:simpleType>
    <xs:complexType name="DataAreaType">
```

```
    <xs:sequence>
        <xs:element name="Customer" type="CustomerType"
            minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="CustomerType">
    <xs:sequence>
        <xs:choice>
            <xs:sequence>
                <xs:annotation>
                    <xs:documentation>For create, delete, or
                        other actions</xs:documentation>
                </xs:annotation>
                <xs:element name="PreviousValues"
                    type="PreviousValuesEmpty" minOccurs="0"/>
                <xs:group ref="AVcustomer"/>
            </xs:sequence>
            <xs:sequence>
                <xs:annotation>
                    <xs:documentation>For change actions
                    </xs:documentation>
                </xs:annotation>
                <xs:element name="PreviousValues"
                    type="customerPreviousValuesType"/>
                <xs:group ref="AVcustomer"/>
            </xs:sequence>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="actionType" type="eventActionDT"
        use="optional"/>
</xs:complexType>
<xs:complexType name="PreviousValuesEmpty"/>
<xs:complexType name="customerPreviousValuesType">
    <xs:sequence>
        <xs:group ref="AVcustomer"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="identifierDT">
    <xs:restriction base="xs:boolean"/>
</xs:simpleType>
<xs:group name="AVcustomer">
    <xs:sequence>
        <xs:element name="customer" type="customerType"/>
        <xs:element name="address" type="addressDT"
            minOccurs="0"/>
        <xs:element name="status" type="statusDT" minOccurs="0"/>
    </xs:sequence>
</xs:group>
<xs:complexType name="customerType">
    <xs:simpleContent>
        <xs:extension base="customerDT">
            <xs:attribute name="identifier" type="identifierDT"
                use="optional" fixed="true"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="customerDT">
    <xs:restriction base="xs:long">
        <xs:totalDigits value="10"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="addressDT">
    <xs:restriction base="xs:string">
```

```
            <xs:maxLength value="9"/>
            <xs:pattern value="\p{Lu}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="statusDT">
        <xs:restriction base="xs:string">
            <xs:enumeration value="active"/>
            <xs:enumeration value="inactive"/>
            <xs:enumeration value="historic"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

The component names and attribute names in this XSD are dependent on the specific trigger. In this example, the component is Customer and the attributes are Customer, Address, and Status. The other parts are generic. The structure will be the same for all triggers.

Note: The component class and attribute names will, in fact, be the ASCII files and ASCII file fields.

Some examples of a change events based on this structure include the following:

```
<EventMessage>
        <ControlArea>
                <eventEntity>customer</eventEntity>
                <eventAction>create</eventAction>
                <eventTimeStamp>2004-12-17T09:17:28Z</eventTimeStamp>
        </ControlArea>
        <DataArea>
                <Customer actionType="create">
                    <customer>2147483647</customer>
                    <status>inactive</status>
                </Customer>
        </DataArea>
    </EventMessage>
    <EventMessage>
        <ControlArea>
                <eventEntity>customer</eventEntity>
                <eventAction>change</eventAction>
                <eventTimeStamp>2004-12-17T09:30:47Z</eventTimeStamp>
        </ControlArea>
        <DataArea>
                <Customer actionType="change">
                    <PreviousValues>
                        <customer>2147483647</customer>
                        <status>inactive</status>
                    </PreviousValues>
                    <customer>2147483647</customer>
                    <status>active</status>
                </Customer>
        </DataArea>
    </EventMessage>

    <EventMessage>
        <ControlArea>
                <eventEntity>customer</eventEntity>
                <eventAction>delete</eventAction>
                <eventTimeStamp>2004-12-23T14:40:18Z</eventTimeStamp>
        </ControlArea>
        <DataArea>
                <Customer actionType="delete">
                    <PreviousValues>
                        <customer>2147483647</customer>
                        <status>active</status>
```

```
                    </PreviousValues>
                </Customer>
        </DataArea>
</EventMessage>
```

The "Examples" chapter provides additional examples.

Events from Exchange

Row events

A row event generated from the Exchange module will always have one of the standard event types: create, change, or delete. In addition, the data area will only have one single component because only one table is involved.

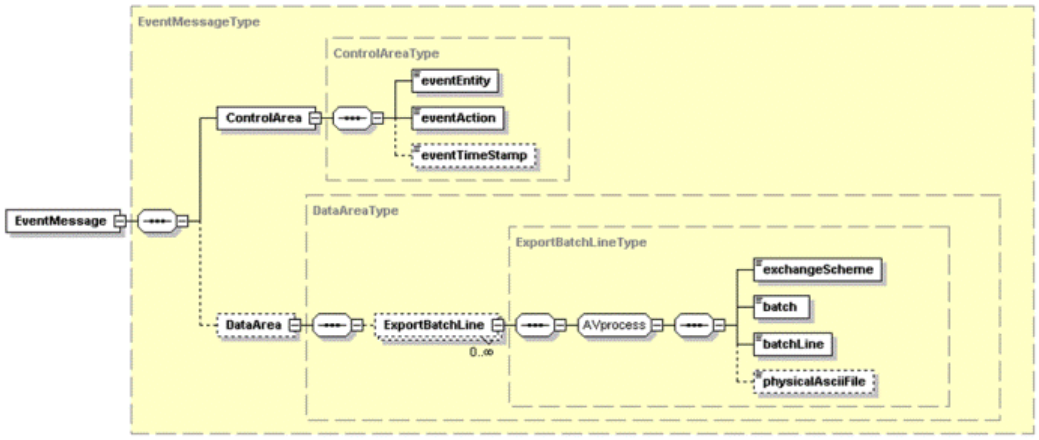
The ASCII file name is mapped to the component name in the event. The ASCII field names are mapped to the attribute names in the event.

As a result, the example provided in "Introduction," previously in this chapter, is valid for events from Exchange, except that the component and attribute names will have a maximum of eight characters and will be lowercase. In addition, if you generate the exchange scheme as described in "To set up a trigger source" in Chapter 2, "To set up triggering," the ASCII file name, and consequently the component name, will be equal to the table code, while the ASCII field names, and consequently the attribute names, will be equal to the table field codes.

End of Data Set event

At the end of a batch line, an End of Set event is generated if an end of batch line trigger is defined in the export trigger.

The following figure shows the event used at the end of an export batch line:



XSD for end of set event

An XSD file for this structure contains the following code:

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.baan.com" xmlns="http://www.baan.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
    <xs:element name="EventMessage" type="EventMessageType"/>
    <xs:complexType name="EventMessageType">
        <xs:sequence>
            <xs:element name="ControlArea" type="ControlAreaType"/>
            <xs:element name="DataArea" type="DataAreaType" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ControlAreaType">
        <xs:sequence>
            <xs:element name="eventEntity" type="eventEntityDT"/>
            <xs:element name="eventAction" type="eventActionDT"/>
            <xs:element name="eventTimeStamp" type="eventTimeStampDT" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="eventEntityDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="eventActionDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="eventTimeStampDT">
        <xs:restriction base="xs:dateTime"/>
    </xs:simpleType>
    <xs:complexType name="DataAreaType">
        <xs:sequence>
            <xs:element name="ExportBatchLine" type="ExportBatchLineType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ExportBatchLineType">
        <xs:sequence>
            <xs:group ref="AVprocess"/>
        </xs:sequence>
        <xs:attribute name="actionType" type="eventActionDT" use="optional"/>
    </xs:complexType>
    <xs:simpleType name="identifierDT">
        <xs:restriction base="xs:boolean"/>
    </xs:simpleType>
    <xs:group name="AVprocess">
        <xs:sequence>
            <xs:element name="exchangeScheme" type="exchangeSchemeType"/>
            <xs:element name="batch" type="batchType"/>
            <xs:element name="batchLine" type="batchLineType"/>
            <xs:element name="physicalAsciiFile" type="physicalAsciiFileDT" minOccurs="0"/>
        </xs:sequence>
    </xs:group>
    <xs:complexType name="exchangeSchemeType">
        <xs:simpleContent>
            <xs:extension base="exchangeSchemeDT">
                <xs:attribute name="identifier" type="identifierDT" use="optional" fixed="true"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="batchType">
        <xs:simpleContent>
            <xs:extension base="batchDT">
                <xs:attribute name="identifier" type="identifierDT" use="optional" fixed="true"/>
            </xs:extension>
        </xs:simpleContent>
```

```
    </xs:complexType>
    <xs:complexType name="batchLineType">
        <xs:simpleContent>
            <xs:extension base="batchLineDT">
                <xs:attribute name="identifier" type="identifierDT" use="optional" fixed="true"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:simpleType name="exchangeSchemeDT">
        <xs:restriction base="xs:string">
            <xs:maxLength value="8"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="batchDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="batchLineDT">
        <xs:restriction base="xs:long"/>
    </xs:simpleType>
    <xs:simpleType name="physicalAsciiFileDT">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
</xs:schema>
```

An example of an end of set event based on this structure is as follows:

```
<EventMessage>
    <ControlArea>
        <eventEntity>exportBatchLine</eventEntity>
        <eventAction>endOfDataSet</eventAction>
    </ControlArea>
    <DataArea>
        <ExportBatchLine actionType="endOfDataSet">
            <exchangeScheme>workflow</exchangeScheme>
            <batch>BATCH</batch>
            <batchLine>1</batchLine>
            <physicalAsciiFile>/home/workflow/export/customer.S
            </physicalAsciiFile>
        </ExportBatchLine>
    </DataArea>
</EventMessage>
```

The physicalAsciiFile will not be set if no ASCII file is created.

Additional examples are provided in Chapter 4, "Examples."

API for event handling

The implementation of the event handling API is stored in the *datrgevent* library.

The library contains the following types of functions:

- Functions to create events You can use these functions from an application customization, but also from the Exchange module.
- Functions to read (use) events and to update (change) events The trigger implementation (conditions and action) can use these functions, if required. In addition, wrapper functions are available to read or create single-component events easily.
- A function to clean up events.

Important!

To read events, you must only use the functions in the *datrgevent* library. The implementation can change.

For details, refer to the specifications of the *datrgevent* library.

Note

You can retrieve the library specifications from the library objects. At operating-system level, use explode6.2 to locate the library object, and subsequently use bic_info6.2 with the **-eu** options.

For example:

```
$ explode6.2 odatrgevent
/mybse/application/myvrc/odatrg/otrgevent
$ bic_info6.2 -eu /mybse/application/myvrc/odatrg/otrgevent
```

This shows the documentation of the event handling functions.

This method provides the following:

- Prototypes Includes the function name and type, and parameters and their types, of the functions in the library
- A description of the functions and their input and output
- The preconditions and post-conditions.

Chapter 4
Examples

4

Introduction

This chapter provides several examples, based on the LN data model.

The sections in this chapter describe the following aspects:

- *The business cases (p. 33)*
- *The setup at implementation time (p. 35)*
- *The XML used at runtime (p. 46)*

Business cases

Sales order entry via EDI

From an external source, the customer's environment receives sales orders from EDI. These sales orders are added to the LN database. As soon as a new sales order is created, a workflow process is triggered to monitor the processing of the sales order.

Technical notes:

Only the creation of the sales order must be taken into account. For example, if the sales order header is updated or sales order lines are added later no new process is triggered. Note that sales orders from EDI are inserted with their sales order lines, either in the same database transaction or in two subsequent transactions.

Sales orders in LN are stored in the Sales Orders (tdsls400) table. Sales orders from EDI can be recognized by means of the **Origin** (tdsls400.corg) field with the value **EDI** (tdsls.corg.eop).

User-defined attributes to be passed on to the workflow engine include the following:

- Order Number (tdsls400.orno)
- Sold-to Business Partner (Customer) ID (tdsls400.ofbp)
- Order Type (tdsls400.sotp)

Inventory on-hand

The customer wants to take a more active approach towards inventory control. For that reason, a workflow process must be activated as soon as the inventory on-hand goes below a particular threshold value for an item that is stored in a warehouse. The process that is triggered contains the steps that must be taken to effectively react to the low inventory.

Technical notes:

In LN, the inventory data per item are stored in the Item Inventory by Warehouse (whwmd215) table. The process must be triggered when the inventory on hand (whwmd215.stoc) drops below a predefined value.

User-defined attributes to be passed on to the workflow engine are the following:

- Warehouse (whwmd215.cwar)
- Item (whwmd215.item)
- Inventory on Hand (whwmd215.stoc)
- Inventory Allocated (whwmd215.allo)

Credit limit (Example requiring application customization)

When you enter sales quotations or sales orders, a credit limit check is carried out. However, the user wants to check not only whether a credit limit is exceeded, but also wants to initiate a process if a credit limited is being approached, for example, if less than 20 percent of the credit limit remains. The process contains the steps to be taken to handle proactively the credit limit, rather than wait until orders can no longer be entered for a customer.

Technical notes:

The credit limit check is carried out in the program that maintains sales orders. When you enter or update a sales order, the tdsls4102 script contains logic to check the credit limit.

A customization is created for this script, which checks whether 80 percent of the credit limit is exceeded. If so, an event that contains the required data is created and a trigger is invoked.

User-defined attributes to be passed on to the workflow engine are, for example:

- Customer ID
- Order Number
- Credit Limit
- Credit in Use

Setup at implementation time

Introduction

The implementation described here is based on the following assumptions:

- In the Exchange scheme, the ASCII file name will be equivalent to the table code, and the ASCII field names will be equivalent to the column code.
- A create XML file type of action is defined. This action has a transformation script defined. In that transformation script, the following is programmed:
  - The event type is changed: The standard events are replaced by a specific event.
  - The class is changed: The table code is replaced with a logical name.
  - The attributes that are not required in the output, but that only had to check the condition, are removed.
  - The attributes that are required in the output are renamed.

Note that these assumptions are not mandatory. For example, the user can use other, more descriptive ASCII file and field names, which reduces the transformations in the trigger. However, in that case, the limitations of the Exchange module must be taken into account. More specifically, the ASCII file and field names cannot be greater than eight characters in length and the ASCII file names will always be in lowercase.

In addition, for the transformation script, you can follow one of the following two approaches:

- Adhere to the standard event structure. In this case, the incoming event is updated as described previously.
- Use a proprietary structure: In this case, the attribute values are picked up from the incoming event, and a new XML is created. In this case, any XML can be created and can be completely tuned to the needs of the workflow engine that receives the XML files.

Audit setup

Audit is switched on for the tables involved: whwmd215 and tdsls400. For details about Audit Management, refer to the Web Help and to the *Infor10 ERP Enterprise Server (LN) Technical Manual (U8172 US)*.

Important!

By default, unchanged columns are not audited in case of an update action. As a result, for example, if the *stoc* (inventory on hand) is changed in *whwmd215*, the values for the primary key fields (*cwar* and *item*) and the changed field (*stoc*) are included, but the unchanged fields are not included. Note that this can impact filtering by conditions, because you cannot check an attribute that is unavailable.

To avoid this limitation, use the Audit Fields by Table (ttaud3125m000) session to specify that unchanged fields must be audited (set the audit type to "Always"). For details, refer to "Unchanged table fields" in chapter 2, "To set up triggering".

Exchange scheme setup

A single exchange scheme is created to pick up the relevant changes. As a result, the job for each of these situations runs at the same frequency, which results in less overhead. If the frequency for checking the changes must differ for the tables, you can create multiple batches or exchange schemes.

The batch includes the two batch lines, covering the following table fields:

Table	tdsls400	whwmd215
Fields	orno	item
	ofbp	stoc
	sotp	allo
	corg	

Exchange scheme properties

Exchange Scheme	workflow
Description	Workflow Scheme for Triggering
Path for Exchange Objects	/home/exchange/workflow
Path for Condition Errors	/home/exchange/workflow
Path for Seq. Files	/home/exchange/workflow
Path for Definition Files	/home/exchange/workflow
Based on Audit	Yes
Based on Indicators	Yes
Default Date Format	YYYYDDMM
Control character y/n	No
Separator	|
Enclosing Character	
Parent Exchange Scheme	

Control character, separator, and enclosing character are irrelevant because no file is created.

Batch properties

Exchange Scheme	workflow (Workflow Scheme for Triggering)
Batch	BATCH
Sequence No.	10
Description	Workflow Export Batch
Company	996
Exchange Using Audit	Yes

ASCII file formats

- Exchange Scheme: workflow (Workflow Scheme for Triggering)
- ASCII file: Item Inventory by Warehouse (whwmd215)

Field No.	10	20	240	260
Field Name	Cwar	Item	Stoc	Allo
Description	Warehouse	Item	Inventory on Hand	Allocated Inventory
Field Type	Alphanumeric	Alphanumeric	Numeric	Numeric
Start Pos	1	4	246	286
Length	3	16	20	20
Date Format				
Floating Dec.	No	No	No	No
Dec.Pos.	0	0	0	0

- Exchange Scheme: workflow (Workflow Scheme for Triggering)
- ASCII file Sales Orders 9 (tdsls400)

Field No.	10	20	50	70
Field Name	orno	ofbp	corg	sotp
Description	Sales Order	Customer	Origin	Order Type
Field Type	Numeric	Alphanumeric	Numeric	Alphanumeric
Start Pos	1	7	22	28
Length	6	6	3	3
Date Format				
Floating Dec.	No	No	No	No
Dec.Pos.	0	0	0	0

Field Type, Start Position, Length, and so on are irrelevant, because no ASCII file is created.

Table and field relations (export)

- Exchange Scheme: workflow (Workflow Scheme for Triggering)
- Batch: BATCH (Workflow Export Batch)

Batch Line	10	20
ASCII File	whwmd215	tdsls400
Desc.	Item Inventory by Ware-house	Sales Orders
Table	whwmd215	tdsls400
ASCII File Name	whwmd215.S	tdsls400.S
Active	Yes	Yes
bdbpre	No	No
Range	No	No
Index	1	1
Cond.		

- Exchange Scheme: workflow (Workflow Scheme for Triggering)
- Batch: BATCH (Workflow Export Batch)
- Batch Line: 10 (whwmd215)

Serial No.	10	20	240	260
Field Name	cwar	item	stoc	allo
Table Field	cwar	item	stoc	allo
Array Element	0	0	0	0
Condition				
Fixed value				

- Exchange Scheme: workflow (Workflow Scheme for Triggering)

- Batch: BATCH (Workflow Export Batch)
- Batch Line: 20 (tdsls400)

Serial No.	10	20	50	70
Field Name	orno	ofbp	corg	sotp
Table Field	orno	ofbp	corg	sotp
Array Element	0	0	0	0
Condition				
Fixed value				

Export triggers

- Exchange Scheme: workflow (Workflow Scheme for Triggering)
- Batch: BATCH (Workflow Export Batch)

Batch Line	10	20
Trigger for Each Row	wf_inventory	wf_salesorder
Create ASCII File	No	No
Trigger for Batch Line End		

Optimizations

If required, to implement optimizations, you can move conditions from the Trigger to the Exchange scheme. For example, you can create a range for batch line 20 (*tdsls400*) on the *corg* (origin) column to only include sales orders created by means of EDI. This reduces the number of XML events that the Exchange module creates. Note, however, that the Exchange export cannot filter on action type, in this example, only include inserts, skip updates, and deletes.

Application customization

To perform the implementation, you must customize the function(s) that check the credit limit of a Business Partner, which is used in the sales order related program scripts. The customization adds a

check to see whether 80 percent of the credit limit is exceeded. If so, the function creates an event and invokes a trigger.

Note

You must check that the previous value of the Credit in Use did not exceed the 80 percent threshold. Otherwise, the process will be triggered multiple times for the same customer, which is undesirable.

If the trigger you invoke is not in use for any other events, no conditions must be defined. The action that you must take is to create a file that contains the XML event.

Implementation

Note

The information in this document is for educational purposes only, by way of example. The information is not intended to describe a complete and tested solution that will work in a live LN environment.

Assume that a library exists that contains a function to check the credit limit. Further assume that the following variables are available in that function:

- invoice.to.bp: the business partner (customer) to which the invoice must be sent
- order.number: The sales order being processed
- credit.limit: The business partner's credit limit
- old.credit.in.use: The credit already used before processing the sales order
- new.credit.in.use: The old.credit.in.use plus the order amount for the new sales order

The original version of the library function contains the following code:

```
if new.credit.in.use > credit.limit then
        return(true) | credit limit exceeded
    else
        return(false) | credit limit not exceeded
endif
```

In a customized version of library, if the usual credit limit check is passed, the 80 percent check is carried out:

```
if new.credit.in.use > credit.limit then
        return(true) | credit limit exceeded
    else
        if new.credit.in.use > 0.8 * credit.limit and
           old.credit.in.use <= 0.8 * credit.limit
        then
            generate.credit.trigger(invoice.to.bp,
                        order.number,
                        credit.limit,
                        old.credit.in.use,
                        new.credit.in.use)
        endif
        return(false) | credit limit not exceeded
endif
```

At the end of the DLL, the following function is added:

```
function generate.credit.trigger(
            domain  tccom.bpid  i.invoice.to.bp,
```

```
            domain  tcorno  i.order.number,
            domain  tcamnt  i.credit.limit,
            domain  tcamnt  i.old.credit.in.use,
            domain  tcamnt  i.new.credit.in.use)
{
#pragma used dll odatrgapi
#pragma used dll odatrgevent

#define MY_TRIGGER  "credit"
#define ERR_IF_NONZERO(value)
^           if value <> 0 then
^               | here some error logging can be implemented,
^               | for example
^               return
^           endif

    long    event       |tree containing trigger event
    long    retl            |return value to be checked
    string  error.mess(256) |error message from trigger
    string  error.details(256)  |error details from trigger

    |add event to trigger
    retl = datrgevent.simpleevent.create("Customer",
                    "creditLimitIsNear", event)
    ERR_IF_NONZERO(retl)
    retl = datrgevent.simpleevent.set.value(event,
            "customerID", i.invoice.to.bp)
    ERR_IF_NONZERO(retl)
    retl = datrgevent.simpleevent.set.value(event,
            "orderNumber", str$(i.order.number))
    ERR_IF_NONZERO(retl)
    retl = datrgevent.simpleevent.set.value(event,
            "creditLimit", str$(i.credit.limit))
    ERR_IF_NONZERO(retl)
    retl = datrgevent.simpleevent.set.value(event,
            "creditInIse", str$(i.new.credit.in.use))
    ERR_IF_NONZERO(retl)
    retl = datrgevent.simpleevent.set.old.value(event,
            "creditInUse", str$(i.old.credit.in.use))
    ERR_IF_NONZERO(retl)

    |invoke trigger
    retl = datrgapi.trigger.do(MY_TRIGGER, event,
                    error.mess, error.details)
    ERR_IF_NONZERO(retl)
}
```

Trigger setup

Triggers

Assume that the following three triggers are defined:

- wf_inventory (for whwmd215)
- wf_salesorder (for tdsls400)
- wf_creditlimit (for the application trigger)

Trigger conditions

In the trigger conditions, you do not have to check the class or table code. If a single trigger was used for multiple classes, you must include the class condition, and you must have the same action performed for each class.

The following conditions are defined:

Trigger wf_salesorder:

Seq Nr	10	20
And/Or		And
Condition Type	Event Type	Attribute
Attribute		corg
Operator	=	=
Value	create	3
Becomes		No

Trigger wf_inventory:

Seq Nr	10
And/Or	
Condition Type	Attribute
Attribute	stoc
Operator	<
Value	20
Becomes	Yes

Trigger wf_creditlimit will not have any conditions specified, because the conditions are checked in the application.

Trigger actions

For each trigger, a create XML file action is specified. The local path, file transfer program, and target path are defined.

Additionally, if required, a transformation script is defined that translates the generic event into an event that the workflow engine can understand.

The minimum requirement will likely be simply to rename the event type into a specific event to enable the workflow engine to detect what event occurred. Note that the class and standard event type are insufficient. The event type will always be Create, Change, or Delete, and the class is not unique because two changes on the same table/class can result in two different workflow processes. If you do not rename the event, the workflow engine must duplicate the conditions as defined for the triggers.

In addition, the attributes must be renamed to UDAs that the workflow engine knows. The previous values can be moved to give the values a unique name. For more information, refer to "XML used at runtime," later in this chapter for examples.

Finally, rather than use the standard event structure, the transformation script can build up a new event structure.

XML used at runtime

Introduction

In ERP, a job is created to regularly run the Exchange export process, for example, every ten minutes. Based on this setup standard create, change and delete events will come out.

The event XML is defined in Chapter 3, "Event Handling." Below this XML structure is applied to the business case. Note that the end of data set event is not used for the business case.

The following two transformation examples illustrate each of the two cases. In the first example, the transformation script renames the event type, renames the attributes, and moves the previous values. In the second example, the transformation script replaces the event with a workflow-specific structure.

Optionally, the user can add an event time to the control area, such as:

```
<eventTimeStamp> 2006-04-06T08:34:52Z </eventTimeStamp>
```

Sales order entry with EDI

Example 1: Create event (relevant)

The Exchange scheme produces the following event:

```
<EventMessage>
        <ControlArea>
            <eventEntity> tdsls400 </eventEntity>
            <eventAction> create </eventAction>
        </ControlArea>
        <DataArea>
            <tdsls400 actionType=create>
                <orno> 1234 </orno>
                <ofbp> SMITH </ofbp>
                <sotp> A1 </sotp>
                <corg> 3 </corg>
            </tdsls400>
        </DataArea>
</EventMessage>
```

After you process this information in the transformation script, the output is as follows:

```
<EventMessage>
        <ControlArea>
            <eventEntity> SalesOrder </eventEntity>
            <eventAction> newEDIOrder </eventAction>
        </ControlArea>
        <DataArea>
            <SalesOrder actionType=create>
                <orderNumber> 1234 </orderNumber>
                <customerID> SMITH </customerID>
                <orderType> A1 </orderType>
            </SalesOrder>
        </DataArea>
</EventMessage>
```

If you use an alternative transformation script that rebuilds the event, the output can be, for example, as follows:

```
<WorkflowEvent>
        <eventType> newEDIOrder </eventType>
        <UDAs>
            <orderNumber> 1234 </orderNumber>
            <customerID> SMITH </customerID>
            <orderType> A1 </orderType>
        </UDAs>
</WorkflowEvent>
```

Example 2: create event (irrelevant)

The Exchange scheme produces following event:

```
<EventMessage>
        <ControlArea>
            <eventEntity> tdsls400 </eventEntity>
            <eventAction> create </eventAction>
        </ControlArea>
        <DataArea>
            <tdsls400 actionType=change>
                <orno> 2345 </orno>
                <ofbp> SMITH </ofbp>
                <sotp> A2 </sotp>
                <corg> 1 </corg>
            </tdsls400>
        </DataArea>
</EventMessage>
```

The condition rejects this event.

Example 3: Delete event (irrelevant)

The Exchange scheme produces the following event:

```
<EventMessage>
        <ControlArea>
            <eventEntity> tdsls400 </eventEntity>
            <eventAction> delete </eventAction>
        </ControlArea>
        <DataArea>
            <tdsls400 actionType=delete>
                <orno> 1234 </orno>
                <ofbp> SMITH </ofbp>
                <sotp> A2 </sotp>
                <corg> 3 </corg>
            </tdsls400>
        </DataArea>
</EventMessage>
```

The condition rejects this event.

Inventory on hand

Example: Change event (relevant)

The Exchange scheme produces the following event:

```
<EventMessage>
        <ControlArea>
            <eventEntity> whwmd215 </eventEntity>
            <eventAction> change </eventAction>
        </ControlArea>
        <DataArea>
            <whwmd215 actionType=change>
                <PreviousValues>
                    <cwar> W01 </cwar>
                    <item> BIKE001 </item>
                    <stoc> 28 </stoc>
                </PreviousValues>
                <cwar> W01 </cwar>
                <item> BIKE001 </item>
                <stoc> 13 </stoc>
            </tdsls400>
        </DataArea>
</EventMessage>
```

After you process this in the transformation script, the output is as follows:

```
<EventMessage>
        <ControlArea>
            <eventEntity> Inventory </eventEntity>
            <eventAction> lowInventory </eventAction>
        </ControlArea>
        <DataArea>
            <Inventory actionType=change>
                <warehouse> W01 </warehouse>
                <item> BIKE001 </item>
                <inventoryOnHand> 13 </inventoryOnHand>
                <inventoryAllocated> 50 </inventoryAllocated>
                <previousValuewarehouse> W01 </previousValuewarehouse>
                <previousValueitem> BIKE001 </previousValueitem>
                <previousValueinventoryOnHand> 28
                </previousValueinventoryOnHand>
            </Inventory>
        </DataArea>
</EventMessage>
```

If you use an alternative transformation script that rebuilds the event, the output can be, for example, as follows:

```
<WorkflowEvent>
        <eventType> lowInventory </eventType>
        <UDAs>
                <warehouse> W01 </warehouse>
                <item> BIKE001 </item>
                <inventoryOnHand> 13 </inventoryOnHand>
                <previousValueInventoryOnHand> 28
                </previousValueInventoryOnHand>
        </UDAs>
</WorkflowEvent>
```

Credit limit check from application

The application customization produces the following event:

```
<EventMessage>
    <ControlArea>
        <eventEntity> Customer </eventEntity>
        <eventAction> creditLimitIsNear </eventAction>
    </ControlArea>
    <DataArea>
        <Customer>
            <PreviousValues>
                <creditInUse> 78000.00 </creditInUse>
            </PreviousValues>
            <customerID> SMITH </customerID>
            <orderNumer> 4567 </orderNumer>
            <creditLimit> 100000.00 </creditLimit>
            <creditInUse> 83000.00 </creditInUse>
        </Customer>
    </DataArea>
</EventMessage>
```

In this case, no transformation is required.

If you use an alternative transformation script that rebuilds the event, the output can be, for example, as follows:

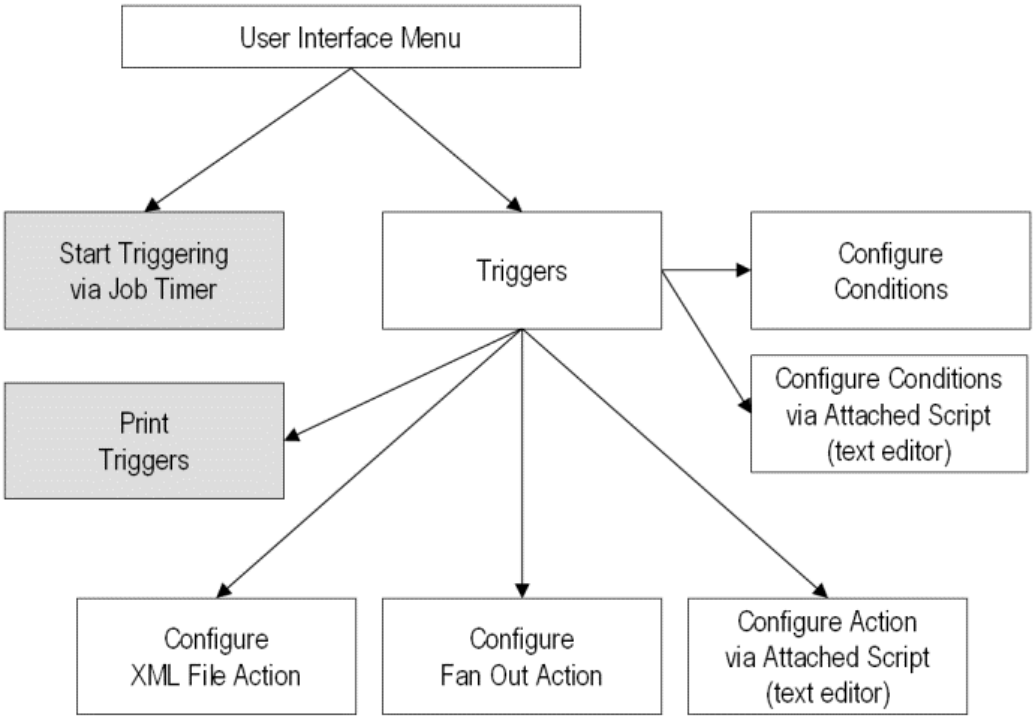
```
<WorkflowEvent>
        <eventType> creditLimitIsNear </eventType>
        <UDAs>
            <customerID> SMITH </customerID>
            <orderNumer> 4567 </orderNumer>
            <creditLimit> 100000.00 </creditLimit>
            <creditInUse> 83000.00 </creditInUse>
        </UDAs>
</WorkflowEvent>
```

Appendix A
Triggering sessions

A

Triggering sessions

The following figure provides an overview of the Triggering sessions and the sessions' relations:



User interface structure

Triggering menu

The **Triggering (mdatrg1000m000)** menu, which is included in the **Integration Tools (mtttls1100m000)** menu, contains the following:

- Sessions to maintain, display, and print triggers

- The Start Triggering via Job Timer (datrg1200m000) session.

From the Triggers (datrg1100m000) session, you can start the configuration sessions for conditions and actions.

Triggering sessions

The following sessions are available in the Triggering (TRG) module:

- Triggers (datrg1100m000)
- Print Triggers (datrg1400m000)
- Configure Trigger Conditions (datrg1110m000)
- Configure Fan Out Action (datrg1120m000)
- Configure XML File Action (datrg1225m000)
- Start Triggering via Job Timer (datrg1200m000)

For a detailed description of the functionality of these sessions, refer to the Web Help.

Appendix B
Exchange sessions

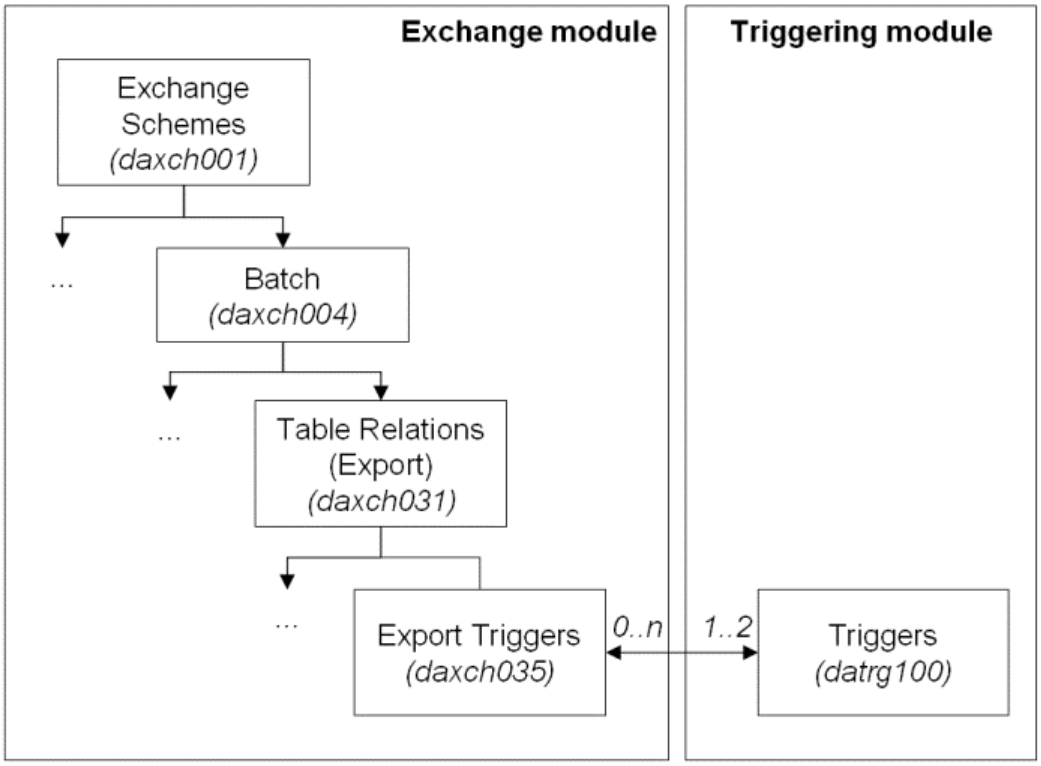
B

Exchange sessions

This chapter specifies the sessions in the Exchange module that use the triggering functionality. For an overview of Exchange, and how to use Exchange, refer to the Exchange online Help and to the *Infor10 ERP Enterprise (LN) Exchange - User`s Guide (U8405 US)*.

After you define an exchange scheme with one or more table relations (export), you can link triggers to those table relations. To do so, you must create so-called export triggers. An export trigger indicates that for a table relation (export), one or more triggers must be invoked.

The following figure shows the relation between Export Triggers and the existing Table Relations (Export) entity on one hand, and the Triggering module on the other hand. An arrow denotes a ‘many’ relationship, while no arrow denotes a ‘one’ relation ship.



Export triggers and their context

The **Export Module (mdaxch3003m000)** menu contains the following session to maintain, display, and print the export triggers:

- Export Triggers (daxch0135m000)

For a detailed description of the functionality of this session, and the Print Export Triggers (daxch0435m000) session that the session incorporates, refer to the Web Help.

Appendix C
External interface

C

External interface

Triggering API

This section briefly describes the application programming interface (API) for Triggering.

The Triggering API is stored in the *datrgapi* library. This library contains functions to:

- Create a trigger
- Check whether a trigger exists to be used from the Exchange module
- Invoke a trigger
- Delete a trigger

The library does not contain functions to change an existing trigger, except functions for adding components such as conditions or actions.

Points of attention

- Triggers are run synchronously
- Component and attribute names cannot exceed 40 characters in length

More information

For detailed information on the Triggering API, refer to the specifications of the *datrgapi* library.

You can retrieve the library specifications from the library objects. At operating-system level, use explode6.2 to locate the library object, and subsequently use bic_info6.2 with the **-eu** options.

For example:

```
$ explode6.2 odatrgevent

/mybse/application/myvrc/odatrg/otrgevent

$ bic_info6.2 –eu /mybse/application/myvrc/odatrg/otrgevent
```

This shows the documentation of the event handling functions.

This method provides the following:

- Prototypes, including the function name and type and parameters and their types, of the functions in the library
- A description of the functions and their input and output
- The preconditions and post-conditions.

Index