Automated Event Management

Configuration and Usage of the Event Management

Table of Contents

1 Introduction

1.1 Purpose of this Document

Xpert.NET is a very dynamic help desk solution that can be configured in various directions. Daily routine and experience have shown that users get acquainted with the most important modules and configurations quickly. Nevertheless, some questions regarding the functionalities of the *Xpert.NET* modules often remain. Thus, the individual modules cannot be used to their full potential.

1.2 Addressees of the Document

This document contains information, instructions, and examples for administrators of *Xpert.NET*.

1.3 Remarks on the Content of this Document

This document describes all the functions of the *Automated Event Management* module in *Xpert.NET*. Nevertheless, the range of functions may differ due to configuration, licensing, and version. Therefore, some of the options listed may not be available on the software provided.

1.4 Overview of the Automated Event Management

The *Automated Event Management* module is able to process large amounts of unfiltered signals or events in order to facilitate the general management of multiple heterogeneous data sources.

The primary benefit of this module - compared to other modules, like *Mail2Ticket* - is the option to filter incoming events on a lower level before they reach the workflow engine. Thus, the AEM can ensure complete and automatic filtering, processing, and collecting of signals, events, and tickets in your IT infrastructure. The AEM allows for converting filtered signals and events to tickets based on workflows as well as for processing signals and events on various protocols and data sources (e.g. SNMP, SQL, etc.). The simple handling of the AEM enables even employees without a technical background to monitor and configure the configuration on their own.

A typical example for the application of this module is the monitoring of system environments, like data centers, in which most of the events are indicators of normal operation. Other types of events that may indicate errors will result in the creation of a ticket.

Chapter 2 explains the configuration of the two different connectors used as interfaces to external data sources. Afterwards, Chapter 3 and 4 illustrate the use of filtering and value rules necessary for filtering incoming events. The event-to-ticket mappings are detailed in Chapter 5. These mappings are used for creating tickets based on events.

2 Configuring the Connectors

Connectors used as interfaces allow for connecting external data sources, services, and databases to the Event Management system.

The following connectors are currently available:

- **Service Broker Connector**: If MS SQL 2005 or later is used as the database server, this connector can be used. The advantage of this connection type is that the Service Broker is forwarding new events to the connector automatically without the need for repeated queries to the database. Thus, the the latency for polling queries can be significantly reduced. The configuration of this connector is simple and does not require a manual setup of the Service Broker queues in the database.
- **Service Broker Queue Connector**: If MS SQL 2005 or later is used as the database server, this connector can be used. This is a lower-level interface to the Service Broker infrastructure and it provides the maximum performance and lowest latency of all connectors. It connects directly to a queue of the Service Broker and retrieves notifications directly from it. The setup requires the administrator to configure the queues and notifications manually on the database.
- **SQL Connector**: If a database server older than MS SQL 2005 is used, or another database server, e.g. Oracle, or an ODBC-compliant database, this connector can be used. Unlike the Service Broker Connector, this connector only forwards new events whenever a poll is performed by the connector.

2.1 Service Broker Connector

The primary advantage of this connector compared to the SQL connector is its use of a "Subscription Query" to detect new events, which minimizes the data acquisition and the computation costs. A soon as a notification has been received, the connector performs another query to collect all of the new events' data. Thus, the lags that may occur by polling can be avoided. The notification of the Service Broker technology is instantaneous. Additionally, this connector makes the Service Broker infrastructure available for use without the need for a manual configuration of the notifications in the database.

The Service Broker requires the following information:

Name: This is the name of the connector. It is used for identification purposes.

Activated: If this checkbox is activated, the connector is in use.

Provider: In this field, the name of the .NET Framework Data Provider (ADO.NET) is to be specified. The provider is used to access the database. Depending on the system used, different values have to be entered here. Examples:

- "System.Data.SqlClient" for SQL servers,
- "System.Data.OracleClient" for Oracle,
- "System.Data.Odbc" for generic ODBC databases,
- or individual provider for different database systems.

Connection String: This connection string specifies the connection details. In most cases, this connection string and the "Subscription connection string" are identical.

Example

```
Data Source=192.168.0.1;Initial Catalog=myDB;UserId=myLogin;
Password=myPassword
```

Query: Defines the SQL database query to be used for retrieving the full event data. This query includes all fields of the event. Ideally, this query should retrieve only a small subset of the data of the new events for performance reasons. To achieve this partial selection, expressions can be used in the query additionally. The following example shows a query to retrieve all entries added since the last query.

Example

```
SELECT
 {$ ServiceBrokerConnector.IdentityColumn $} AS IdColumn,
 {$ ServiceBrokerConnector.Provider $} AS Provider,
 {$ ServiceBrokerConnector.LastPoll.Time[yyyy/MM/dd hh:mm:ss] $}
 AS LastPollTime,
 {$ ServiceBrokerConnector.LastPoll.Identities.Count $} AS LastPollIDCount,
 {$ ServiceBrokerConnector.LastPoll.Identities.Contains[41] $}
 AS LastPollContains,
FROM myEvents
WHERE
 column1 IS NOT NULL
AND eventDate > COALESCE({$ ServiceBrokerConnector.LastDetectedRow[eventDate]
$}, CONVERT(nvarchar(50), GETUTCDATE()-10, 126))
ORDER BY eventDate ASC
```

Identity column: This column is used for recognizing possible data duplicates. It does not have to be the primary key column. If the table does not contain a column with distinct values, a respective column has to be added (e.g. `SELECT col1 + col2 + col3 as IdCol [...]`), which can be used to detect duplicates.

Source system: A name for the source can be entered here. This value will be attached to the respective event, in order to retrace the system used to generate an event. This field is only used for identification purposes after the event has been processed. For example, if *NAGIOS* is entered as a *Source system*, the ticket mapping will be configured to assign this value to a ticket field, and then tickets created after a trigger by a NAGIOS event will be handled differently.

Subscription connection string: Defines connection details for the subscription query. The subscription query is used for receiving notifications on new datasets. In most cases, the value of this field and the *Connection string* field will be identical.

Example

```
Data Source=192.168.0.1;Initial Catalog=myDB;UserId=myLogin;
Password=myPassword
```

Subscription query: This is the query used to subscribe to notifications. The connector will be informed as soon as there are changes that modify the query result.

Example

```
SELECT id FROM myEvents
```

2.2 Service Broker Queue Connector

This is a lower-level connector compared to the *Service Broker Connector*. It provides the best performance of all available connectors and has the lowest latency as well. This connector is also based on the Service Broker infrastructure, but it operates on a lower-level compared to the *Service Broker Connector*, giving the administrator maximum flexibility, but also requiring a more detailed knowledge for its use.

The primary advantage of this connector over the *Service Broker Connector* is that it does not require a repeated query in order to retrieve the new data. Instead, it receives a notification already containing all details and content for the event, resulting in an optimal query performance. This is the recommended connector for systems, in which a single database contains an extremely large amount of incoming events per second and performance is a critical factor.

This connector requires a SQL Server 2005 or newer to operate.

The Service Broker Queue requires the following information:

Name: This is the name of the connector used for its identification.

Activated: If this checkbox is activated, the connector is in use.

Connection String: This connection string defines the connection details. A SQL Server connection string is to be used here.

Example

```
Data Source=192.168.0.1;Initial Catalog=myDB;

User Id=myLogin;Password=myPassword
```

Queue name: Defines the name of the Service Broker queue to contain the events. This queue has to be setup manually in the database.

Example

```
Isonet_EventManagement_TargetMessageQueue
```

Example

The following scripts show an exemplary configuration of a queue named *Isonet_EventManagement_TargetMessageQueue* in the database:

```
IF EXISTS (SELECT * FROM sys.services
WHERE name = 'Isonet_EventManagement_InitiatorService')

BEGIN

 DROP SERVICE Isonet_EventManagement_InitiatorService

END

GO

IF EXISTS (SELECT * FROM sys.services
WHERE name = 'Isonet_EventManagement_TargetService')

BEGIN

 DROP SERVICE Isonet_EventManagement_TargetService

END

GO

IF EXISTS (SELECT * FROM sys.service_contracts
WHERE name = 'Isonet_EventManagement_Contract')

BEGIN

 DROP CONTRACT Isonet_EventManagement_Contract

END

GO

IF EXISTS (SELECT * FROM sys.service_message_types
WHERE name = 'Isonet_EventManagement_MessageType')

BEGIN

 DROP MESSAGE TYPE Isonet_EventManagement_MessageType

END

GO

IF OBJECT_ID('[dbo].[Isonet_EventManagement_MessageQueue]')
IS NOT NULL AND EXISTS(SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID('[dbo].[Isonet_EventManagement_MessageQueue]') AND
type = 'SQ')

BEGIN

 DROP QUEUE [dbo].[Isonet_EventManagement_MessageQueue]

END

GO
```

```
IF OBJECT_ID('[dbo].[Isonet_EventManagement_TargetMessageQueue]') IS NOT NULL
AND EXISTS(SELECT * FROM sys.objects WHERE object_id = OBJECT_ID
('[dbo].[Isonet_EventManagement_TargetMessageQueue]') AND type = 'SQ')

BEGIN

 DROP QUEUE [dbo].[Isonet_EventManagement_TargetMessageQueue]

END

GO

CREATE MESSAGE TYPE Isonet_EventManagement_MessageType
VALIDATION = WELL_FORMED_XML

GO

CREATE CONTRACT Isonet_EventManagement_Contract
(Isonet_EventManagement_MessageType SENT BY INITIATOR)

GO

CREATE QUEUE [dbo].[Isonet_EventManagement_TargetMessageQueue]

GO

CREATE QUEUE [dbo].[Isonet_EventManagement_MessageQueue]

GO

CREATE SERVICE Isonet_EventManagement_InitiatorService ON QUEUE
[dbo].[Isonet_EventManagement_MessageQueue]

GO

CREATE SERVICE Isonet_EventManagement_TargetService ON QUEUE
[dbo].[Isonet_EventManagement_TargetMessageQueue]
([Isonet_EventManagement_Contract])

GO
```

And once a queue is created, the notifications to that queue need to be setup. The following example creates a notification on insertion on a table called "IncomingEvents":

```
CREATE TRIGGER [dbo].[IncomingEvents_TriggerNotification] ON [dbo].[IncomingEvents]


AFTER INSERT AS

 DECLARE @conversationHandle UNIQUEIDENTIFIER

 DECLARE @message XML

 SET @message = (SELECT TOP 1 * FROM INSERTED FOR XML AUTO, ELEMENTS)

 BEGIN DIALOG CONVERSATION @conversationHandle

   FROM SERVICE Isonet_EventManagement_InitiatorService

   TO SERVICE 'Isonet_EventManagement_TargetService'

   ON CONTRACT Isonet_EventManagement_Contract WITH ENCRYPTION = OFF;
```

```
  SEND ON CONVERSATION @conversationHandle MESSAGE
TYPE Isonet_EventManagement_MessageType (@message)

GO
```

Source system: Specifies the name of the source. This value is attached to the respective event in order to retrace, which system has generated an event. It is only used for identification purposes, after the event has been processed. Example: If *Source system* is set to *NAGIOS*, then the event-to-ticket mapping is configured to assign this value to a ticket field; thus, tickets created by a NAGIOS event will be handled differently.

Threads: Defines the number of threads used to collect events. In most cases, a value of 1 is sufficient, due to the very light processing cost of collecting an event.

2.3 SQL Connector

The configuration of the SQL connector is analogous to the configuration of the Service Broker. The only difference is that this connection will not send instant notifications on data changes in the database automatically. In this case, the connector has to query the database at assigned intervals (polling).

Name: This is the name of the connector used for its identification.

Activated: If this checkbox is activated, the connector is in use.

Provider: The name of the .NET Framework Data Provider (ADO.NET) is to be specified here. The provider is used for accessing the database. Depending to the system used, different values have to be entered here. Examples:

- For SQL Server use “System.Data.SqlClient”,
- for Oracle use “System.Data.OracleClient”,
- for generic ODBC databases use “System.Data.Odbc”,
- or custom providers for other databases.

Connection String: This connection string specifies the connection details.

Example

```
Data Source=192.168.0.1;Initial Catalog=myDB;UserId=myLogin;
Password=myPassword
```

Query: Defines the SQL database query to be used to retrieve the full event data. This query covers all fields of the event. Ideally, this query should retrieve only a small subset of data on the new events for performance reasons. In order to achieve this partial selection, expressions can be used within the query additionally. The following example shows the query for retrieving all the new entries added since the last query.

Example

```
SELECT
{$ ServiceBrokerConnector.IdentityColumn $} AS IdColumn,
{$ ServiceBrokerConnector.Provider $} AS Provider,
```

```
{$ ServiceBrokerConnector.LastPoll.Time[yyyy/MM/dd hh:mm:ss] $} AS LastPollTime,
{$ ServiceBrokerConnector.LastPoll.Identities.Count $} AS LastPollIDCount,
{$ ServiceBrokerConnector.LastPoll.Identities.Contains[41] $} AS LastPollContains,
FROM myEvents
WHERE
column1 IS NOT NULL
AND eventDate > COALESCE({$ ServiceBrokerConnector.LastDetectedRow[eventDate]
$}, CONVERT(nvarchar(50), GETUTCDATE()-10, 126))
ORDER BY eventDate ASC
```

Identity column: This column is used for recognizing possible data duplicates. It does not necessarily have to be the primary key column. If the table does not contain a column with unique values, a respective column has to be added (e.g. SELECT col1 + col2 + col3 as IdCol [...]), which can be used to detect duplicates.

Source system: A name for the source can be entered here. This value will be attached to the respective event, in order to retrace the system used to generate an event. This field is only used for identification purposes after the event has been processed. For example, if *NAGIOS* is entered as a *Source system*, the ticket mapping will be configured to assign this value to a ticket field, and then tickets created after a trigger by a NAGIOS event will be handled differently.

Poll interval: This is the time between two database queries. Is is to be specified in seconds.

3 Filtering Rules

One of the central goals of this module is to provide a way to easily define filtering criteria for incoming events. Filtering rules allow for defining exact criteria that determine whether an event is to be converted into a ticket.

The task list for managing the filtering rules and their conditions can be found on the left.

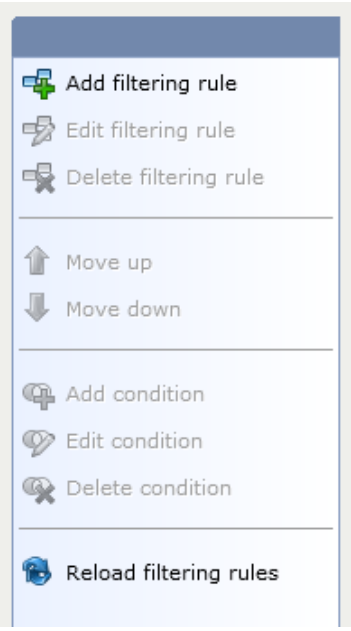

Figure 3.1: Tasks for filtering rules

Add filtering rule/Edit filtering rule/Delete filtering rule	These three buttons allow for creating, editing, and deleting filtering rules.
Move up/down	These buttons are used to move entire rules upwards or downwards in the list. The rules will applied top down one after the other.
Add condition/Edit condition/Delete condition	These three buttons allow for creating, editing, and deleting conditions for the filtering rules.
Reload filtering rules	After all filtering rules have been set up correctly, the list has to be reloaded via this button. Subsequently, they will be used by the system. This button allows for changes to the rules while the system is still running. These changes will only be applied after a click on this button.

3.1 Managing Filtering Rules

In order to create a new rule, click on the button *Add filtering rule* in the task menu. Existing rules can be opened via a double-click or the button *Edit filtering rule*.

Subsequently, a dialog opens, in which the following information has to be entered:

Name: The name of the filtering rule. If multiple languages have been activated, the name can be localized for each language.

Action type: Defines the action to be executed as soon as the rule is activated. The actions available are *Create ticket* and *Discard event*. The action *Create ticket* requires the selection of a ticket mapping!

Valid from/until: Via these options, the validity period of filtering rules can be set. Either *Always* or a specific date and time can be selected here.

Active: Whenever the filtering rule is to be deactivated temporarily, e.g. for revision, this box can be unchecked.

After all information has been entered, a click on the *OK* button creates the new rule or saves the changes in an existing one.

Afterwards, the conditions for the filtering rule have to be set. The following *Section 3.2* illustrates all conditions available at present.

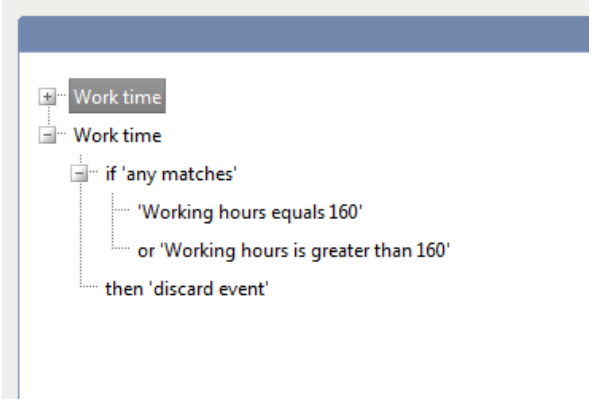

Figure 3.2: An example for filtering rules

If more than one condition is created for a filtering rule, these conditions will be automatically linked to the *Group* condition using an "AND" operator. When other conditions are added afterwards, these conditions will be created dependent on the currently selected condition in the tree view. If a condition is to be created below the group condition, this group condition has to be selected. If a contained condition is selected, a new grouping will be created.

Please note

During creation of the conditions, please mind that grouped conditions cannot be moved!

3.2 Condition Types

The following section details the available types of conditions:

Configuration Item

This conditions checks whether the field content matches the Visible ID of a CI contained in the CMDB.

Visible ID field name: Name of the event field to contain the Visible ID of a CI.

Event Source Suspension Condition

This condition checks whether an event source (the system causing the event) is suspended.

Event source ID field: Defines the name of the event field. In case of an SQL Connector, this would be the name of the result column.

Is suspended?: If this option is activated, the system checks whether the event source has been suspended.

Field Value Comparison

This condition type compares an event field to a predefined value.

Field name: The name of the event field the predefined value is to be compared to. Field names are generally assigned by the connector. If, for example, an SQL database is defined as event source, the field name equals the column name of the SELECT query result.

Comparison: Defines the comparison type between the event value and the comparative value. The available options are *Is equal*, *Is not equal*, *Is less than*, *Is greater than*, *Starts with*, *Ends with*, and *Contains*.

Value: This is the predefined value that is to be compared to the event field.

Ignore upper/lower case: If this box is checked, case sensitivity is ignored in the comparison.

Geographic Location

This condition checks whether a geographic location is located within a certain radius around a point of comparison.

Latitude field name: Defines the name of the event field the predefined value is to be compared to, which contains the latitude in decimal notation. (Example: eventLat)

Longitude field name: Defines the name of the event field the predefined value is to be compared to, which contains the longitude in decimal notation. (Example: eventLong)

Latitude: The latitude of a point of comparison in decimal notation. (Example: 51.338312)

Longitude: The longitude of a point of comparison in decimal notation. (Example: 12.376339)

Radius in meters: The radius the coordinates are allowed to be located in around the point of comparison, specified in meters.

Outside POI?: If this option is activated, the system checks whether the coordinates are located outside of the defined radius around the point of comparison.

Group

Existing conditions can be grouped via the *AND* and *OR* operators.

List

This condition allows for comparing field contents to a list of values.

Field name: The name of the field as assigned by the connector. (Example: ServerName)

Operator: Defines whether an element on the list is to be contained in the field or not.

List of values: Multiple values can be entered one after another, separated by a delimiter. (Example: isoserver01;isoserver04;isoserver82)

Delimiter: The delimiter to be used in the list of values can be specified here. (Example: ;)

Time Period

This condition checks whether the field value is located within a specific time period.

Field name: The name of the field that contains the time value as assigned by the connector. (Example: eventDateTime)

Field format: The format used for parsing the value of the field into a valid date/time. (Example: yyyy-MM-ddTHH:mm:ss.fff)

From/To: Defines the date interval, specified in the 24-hour format (Example: 18:00)

Formatting locale: The locale used for parsing region-specific data, e.g. the date in the event (Example: en-US). If not specified, a generic one will be used.

Value Rule

This condition allows for embedding an existing value rule (Settings -> Event Management -> Value Rules) into a filtering rule. This allows the reuse of complex condition sets.

Weekday

This condition checks whether a date is on a specific weekday.

Field name: The name of the field containing the date as assigned by the connector. (Example: eventDateTime)

Field format: The format used for parsing the value of the field into a valid date/time. (Example: yyyy-MM-ddTHH:mm:ss.fff)

Day of week: The day of the week the field is to be compared to can be specified here. Multiple days can be selected by holding the Ctrl- or Shift-key and then selecting the desired values.

Formatting locale: The locale used for parsing region-specific data, e.g. the date in the event (Example: en-US). If not specified, a generic one will be used.

4 Value Rules

Value rules are generally used for simplification. Complex conditions, which need to be used several times, for example, can be configured in value rules in order to be used in filtering rules at a later date.

On the left side, the tasks for the value rules can be found.

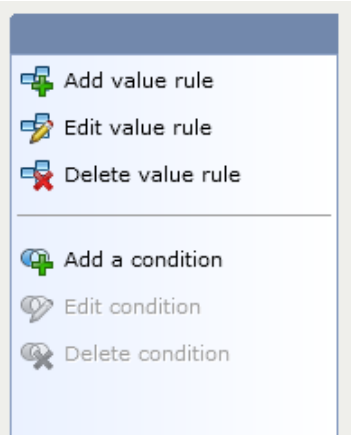

Figure 4.1: The value rules menu with the individual entries

Add value rule/Edit value rule/Delete value rule	These three buttons allow for creating, editing the name of, and deleting value rules.
Add a condition/Edit condition/Delete condition	These three buttons allow for creating, editing, and deleting value rule conditions.

Please note

Value rules will not be used by the system before they are used in the filtering rules.

4.1 Managing Value Rules

In order to create a new value rule, the button *Add value rule* can be used in the task menu on the left. In the following dialog, a name for the new rule is to be entered. If multiple languages have been activated on the system, the name has to be localized for each language. A click on *OK* creates the new rule. If an existing rule is to be modified, this very dialog can be accessed via a double-click on the rule.

Please note

Rules that contain invalid conditions or have not been fully configured will be displayed in red and cannot be used in the filtering rules before all conditions are valid.

In the next step, the conditions for the value rules have to be set. *Section 3.2* details all available conditions.

After the configuration of the value rule has been finished, this rule can be used in the filtering rules. The condition *Value rule* allows for embedding existing valid value rules into the existing filtering system.

5 Event-to-Ticket Mappings

As soon as a filtering rule triggers the creation of a ticket, the event-to-ticket mappings can be used in order to write data available in the event source into the ticket.

The event-to-ticket mappings allow for assigning event source data to ticket fields. The configuration page for event-to-ticket mappings can be accessed via Settings -> Event Management -> Event to Ticket Mapping.

5.1 Mapping Configuration

A new mapping can be created via the button *New mapping*. Existing mappings can be edited by double-clicking on them. A dialog window for entering the following information will open:

Details

Name: The name of the mapping.

Ticket Schema

Ticket schema: Defines the ticket schema that is used to create tickets as soon as a filtering rule is complied with.

> **Please note**
>
> After saving the mapping, the ticket schema cannot be changed anymore.

If the Service Portfolio has been activated on the system, service transactions can be used to create tickets additionally to ticket schemas. For this purpose, the entry *Service Portfolio* has to be selected from the ticket schema drop-down list. Below this list, a service transaction selection field will appear, in which either the name of the transaction can be entered or a service transaction can be selected via the Service Portfolio browser.

After the desired transaction has been selected, the description for this service transaction is displayed and the respective linked configuration item (CI) can be selected below.

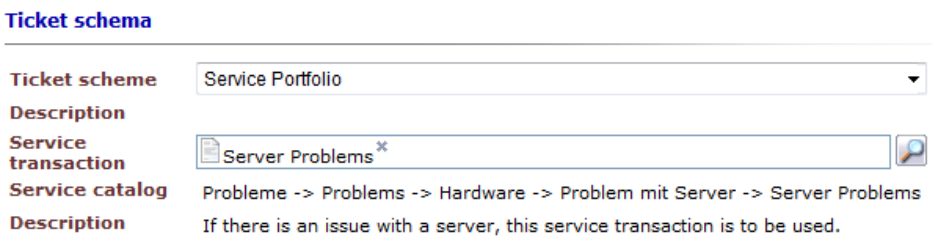

Figure 5.1: A mapping via the Service Portfolio

Creator

Mapping type: The following options are available here:

- **Use specific user**: The user set in the user browser as default user will be the ticket creator.
- **Match by object GUID**: An ObjectGuid from the source will be compared to the ObjectGuids of the users in the *Xpert.NET* system. If a matching ObjectGuid is found, the respective user will be set as the ticket creator. Otherwise the default user will be set.
- **Match by user name**: A value from the source will be compared to the user names in the *Xpert.NET* system. If a matching name is found, the respective user will set as the ticket creator. Otherwise the default user will be set.
- **Match by user field**: A value from the source will be compared to the content of a user field of all users in the *Xpert.NET* system. If a matching value is found, the respective user will set as the ticket creator. Otherwise the default user will be set.

For all mapping types but *Use specific user*, there are additional options available, as it is assumed that the event will contain an identifier of a user. The identifier will vary depending on the selected *Mapping type*:

Source: This allows for defining where to look for the user identifier in the event's content. The available options are the *Header* field or any other standard *Field* of the event.

- **Header field**: Only available, if *Header* has been selected as a source. It displays a list of the predefined header fields available. Although a rare case, a header field might contain enough information to identify a user.
- **Field**: Only available for the source *Field*. This text box is used for determining the field name of the event that contains the user identifier. For example, enter "reporter" in this textbox, in order to access a field with the same name in the event that will contain the name of the user to be set as ticket creator.
- **Custom attribute**: Only available for the mapping type *Match by user field*. This mapping type will try to locate the user based on a custom attribute. For example, if there are users with a secondary e-mail address, which is to be used as the identifier, the value "e-mail internal", for example, has to be entered into the text field, in order to define this field for the mapping.

Default user: The default user is set as the creator if the creator mapping does not have a result or if the mapping type is set to *Specific user*.

Affected user

The options for mapping affected users are identical to those of the ticket creator mapping.

After a click on *Save*, the mapping will be generated and the ticket field mappings can be created.

5.2 Field Mapping Configuration

The tab *Field mappings* allows for assigning source data fields to ticket fields.

A new mapping of event source data to ticket fields can be performed via a click on the *New mapping* button in the overview. If at least one mapping is already on the list, the *Actions* button for deleting existing mappings will be displayed.

If a new mapping is created via the *New mappings* button, a dialog opens and the following information has to be entered:

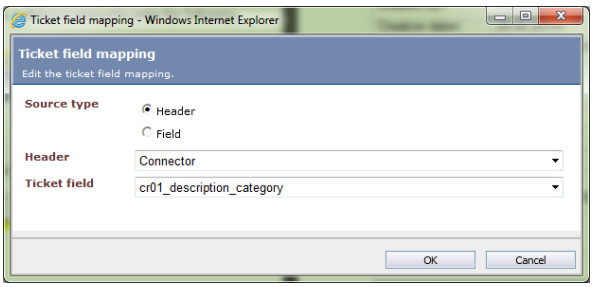

Figure 5.2: A ticket field mapping with header data

Source type: Defines whether header or field data of the source are to be used for filling in the ticket field.

Header/Field: Depending on the selection of the source type, this field allows for selecting a certain header data field or entering a name for the field.

Ticket field: Defines the ticket field of the ticket schema, into which all field or header data of the event will be written.

After the configuration has been finished, the ticket field mapping can be saved via a click on *OK* and further mappings can be added.

As soon as all mappings have been created, the desired mapping can be used in the respective filtering rule (*Section 3.1*).

6 Statistics and Change Management

6.1 Statistics

Created by	Maik Wisatzke
Creation date	20.01.2012
Doc-ID	DOC-190612-010
Version	2015-1
Status	Approved
Replaces version	3.10-3
Release date	12.06.2015
Valid from	Immediately
Valid until	Cancellation
Document name	Automated Event Management 2015-1_EN

6.2 Change Control

Version	Date	Executed by	Comment
3.8-1	02.04.2012	Maik Wisatzke/ Gonzalo Casas/ Steffi Kurnot	Created
3.8-2	31.07.2013	Maik Wisatzke/ Gonzalo Casas/ Anna Hajduk	Updated with new Service Broker Queue connector
3.8-3	22.01.2014	Maik Wisatzke	New design adapted, new figures
2015-1	12.06.2015	Alexander Schmidt	Updates for 2015