Verknüpfte Objekte in OTRS

Eintausendundeine Anfrage zu demselben Thema oder zumindest ähnlich gelagerte Fälle. Je mehr man mit einem Massengeschäft zu tun hat, umso eher wird einem genau das begegnen. Es ist dann natürlich sinnvoll, solche Anfragen zu verknüpfen. So kann man sich leicht durch die Anfragen "navigieren" und Lösungen von Kollegen finden.

OTRS bietet aber nicht nur die Möglichkeit, Anfragen zu verknüpfen, sondern beliebige Objekte miteinander zu verbinden. Das ist praktisch wenn die ITSM-Erweiterung im Einsatz ist und man Anfragen mit Configuration Items verknüpfen kann. So kann man schnell die notwendigen Zusatzinformationen für das Ticket finden.

In bestimmten Fällen reichen die eingebauten Verknüpfungsmöglichkeiten nicht aus. In diesem Artikel wird erst gezeigt, wie neue Verknüpfungsmöglichkeiten für existierende Objekte erschaffen werden können und danach wird darauf eingegangen, wie neue Objekte und damit neue Verknüpfungen eingeführt werden.

Zwischen Tickets können standardmäßig "Normale" Verbindungen geschaffen werden, damit werden gleichwertige Tickets verbunden. Als zweite Möglichkeit gibt es die Eltern-Kind-Beziehung. Damit schafft man gewisse Hierarchien und es gibt die Möglichkeit, dass im OTRS das Elternticket erst dann geschlossen werden kann, wenn alle Kindtickets geschlossen sind.

Jetzt soll die Verknüpfung "Massenstörung Eltern-Kind" eingeführt werden. Wenn z.B. der Mailserver ausfällt, werden sich in größeren Firmen garantiert einige hundert Benutzer melden und jeweils ein Ticket aufmachen. Es ist aber mühselig, sich um jedes einzelne Ticket zu kümmern. Darum soll es zu dem Vorfall ein Masterticket geben, in dem der Agent alle Informationen zusammenträgt. Die Tickets der Endanwender werden diesem Ticket untergeordnet und wenn das Masterticket geschlossen wird, werden auch gleichzeitig alle untergeordneten Tickets geschlossen und die Anwender benachrichtigt.

Dieses automatische Schließen der Kindtickets wird im Endeffekt über ein Ticket-Event-Modul gelöst. Das wird in einer weiteren Ausgabe von $foo besprochen. Aber damit dieses Schließen getriggert werden kann, muss es diesen neuen Verknüpfungstyp "Massenstörung Eltern-Kind" geben.

Da hier nur bestehende Objekte - nämlich Tickets - miteinander verknüpft werden, ist hier keinerlei Programmierarbeit notwendig. Aber der Vollständigkeit halber wird es hier dennoch gezeigt.

Der neue Verknüpfungstyp wird einfach über die SysConfig hinzugefügt (Listing 1). Möglichkeit 1 ist, es als Datei unter <OTRS_HOME>/Kernel/Config/Files/MassIncident.xml zu speichern und dann als OTRS-Administrator die SysConfig aufzurufen, damit die Änderungen aktiv werden.

Die unüblichere Variante ist es, die Daten direkt in das Modul Kernel::Config innerhalb der Subroutine `Load` zu schreiben:

```
$Self->{'LinkObject::Type'}
   ->{'MassIncident'} =  {
       'SourceName' => 'MassIncident Parent',
       'TargetName' => 'MassIncident Child'
};
$Self->{'LinkObject::PossibleLink'}
   ->{'0600'} =  {
       'Object1' => 'Ticket',
       'Object2' => 'Ticket',
       'Type' => 'MassIncident'
};
```

```
<ConfigItem Name="LinkObject::PossibleLink###0600-Massincident" Required="0" Valid="1">
    <Description Translatable="1">Links 2 tickets with a "MassIncident" link.</Description>
    <Group>Ticket</Group>
    <SubGroup>Core::LinkObject</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Object1">Ticket</Item>
            <Item Key="Object2">Ticket</Item>
            <Item Key="Type">MassIncident</Item>
        </Hash>
    </Setting>
</ConfigItem>
<ConfigItem Name="LinkObject::Type###MassIncident" Required="1" Valid="1">
    <Description Translatable="1">
        Defines the link type 'MassIncident'. If the source name and the target
        name contain the same value, the resulting link is a non-directional one;
        otherwise, the result is a directional link.
    </Description>
    <Group>Ticket</Group>
    <SubGroup>Core::LinkObject</SubGroup>
    <Setting>
        <Hash>
            <Item Key="SourceName">MassIncident Parent</Item>
            <Item Key="TargetName">MassIncident Child</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

Listing 1

Bei dem Konfigurationsparameter `LinkObject::PossibleLink` wird festgelegt, zwischen welchen Objekten diese Verbindung hergestellt werden kann. Der Typ nennt sich "MassIncident". Das ist der Wert, der in der Datenbank auch gespeichert wird. Über dieses "MassIncident" kann dann später identifiziert werden, auf welche Art und Weise die Objekte verbunden sind. Es besteht nämlich die Möglichkeit, verschiedene Verknüpfungen für ein und dasselbe Objekt zu etablieren.

Beide Objekte sind hier "Ticket", d.h. dass es zwischen "Ticket" und "ConfigItem" keine Verknüpfung vom Typ "MassIncident" geben kann.

Der zweite Konfigurationsparameter - `LinkObject::Type` - beschreibt den Verknüpfungstyp "MassIncident" genauer. Hier wird ausgesagt, dass der Ausgangspunkt der Verknüpfung das Elternticket der Massenstörung ist und der Endpunkt das Kindticket.

Wenn man jetzt auf den "Verknüpfung"-Link im Ticket (Abbildung 1) klickt und das Ticket mit einem anderen Ticket verknüpfen will, steht der neue Verlinkungstyp schon zur Verfügung (Abbildung 2).

Verknüpfungen mit neuen Objekten

Jetzt aber zu dem Teil, bei dem Verknüpfungen mit neuen Objekten ermöglicht werden soll. Im OTRS von Perl-Services.de laufen auch Anfragen zu OPAR (OTRS Package ARchive) auf. Häufig sind das Sachen, die auch auf GitHub als Issues auftauchen. In solchen Fällen sollen die Tickets mit einer

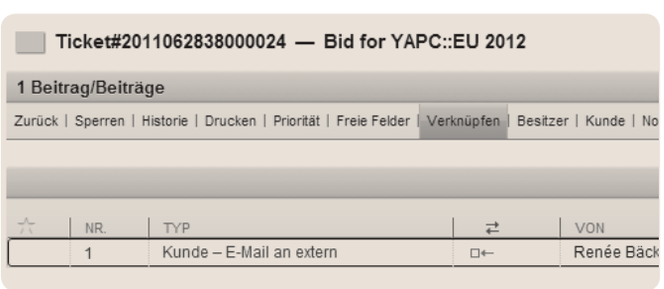


Abbildung 1: Verknüpfen-Link im Ticket-Menü

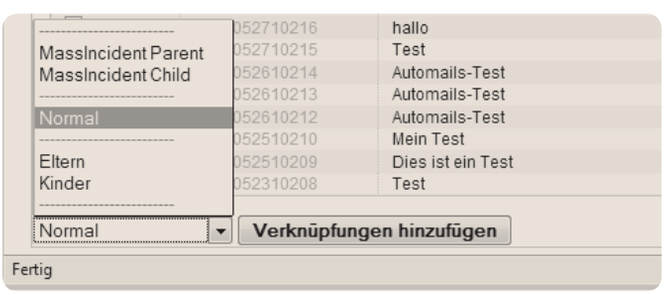


Abbildung 2: Massenstörung als Verknüpfungstyp

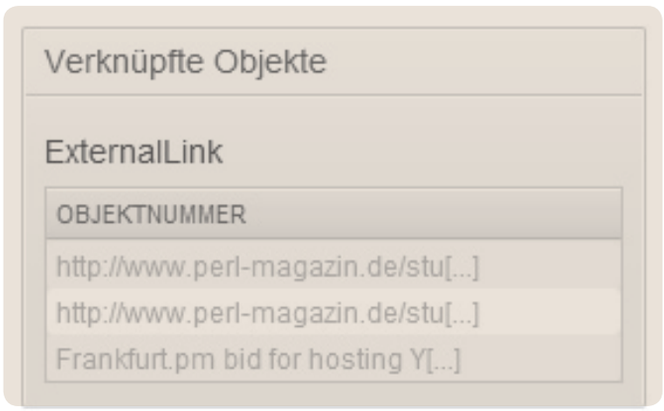


Abbildung 3: Verknüpfte URLs

URL verknüpft werden. Damit erscheint unter dem Ticket in der Tabelle der Link zu dem Issue auf GitHub (Abbildung 3).

Zunächst sollte man wissen, wie das System der verlinkten Objekte in OTRS funktioniert.

Zunächst wird betrachtet, wie es funktioniert, wenn eine Verlinkung erzeugt werden soll. Dazu muss OTRS erstmal wissen, welche Möglichkeiten es überhaupt gibt. Befinde man sich in der Ticketansicht, muss OTRS wissen, welche Verknüpfungstypen gibt es für Tickets. Ist gerade die Ansicht von ConfigItems geöffnet, werden die möglichen Verknüpfungstypen für ConfigItems benötigt.

Das zentrale Modul bei Verlinkungen ist Kernel::System::LinkObject. Das stellt alle notwendigen Funktionen zur Verfügung. Sollen die möglichen Verknüpfungstypen herausgefunden werden, liest das Objekt des LinkObject-Moduls die Konfiguration aus und wertet die Einträge von "LinkObject::PossibleLink" (siehe auch Beispielkonfiguration von oben) aus. Möglich sind die Werte, die bei `Object1` und `Object2` eingetragen sind.

Anschließend wird noch geprüft, ob ein entsprechendes Backendmodul in Kernel/System/LinkObject/ liegt, bei der Beispielkonfiguration also Kernel/System/LinkObject/Ticket.pm

Auch die Anzeige der Verlinkungen wird über das zentrale Modul Kernel::System::LinkObject gemacht. Über das Objekt des LinkObject-Moduls werden die Daten aus der Datenbank geholt und ein Hash mit allen möglichen Informationen erstellt. Daraus lassen sich die Informationen darüber welcher Objekttyp das Ziel ist, ob das Ticket Quelle oder Ziel ist und welcher Art die Verlinkung ist. So ein Hash ist in Listing 2 aufgeführt

In der Datenbank stehen nur IDs - siehe Listing 3.

Die Informationen an sich werden dann wieder über die Backendmodule geholt.

Mit diesem Wissen kann es jetzt an die Arbeit gehen. Da die Steuerung über das zentrale Modul Kernel::System::LinkObject geht, werden nur die Konfiguration und die Backendmodule benötigt. Wie diese aussehen, werden in den folgenden Abschnitten gezeigt.

Als erstes werden wieder Konfigurationsparameter wie oben benötigt. Diesmal ist das Quellobjekt ein Ticket und das Zielobjekt eine URL - siehe Listing 4.

```
$LinkList = {
    Ticket => {
        Normal => {
            Source => {
                12  => 'Ticket 1',
            },
        },
        ParentChild => {
            Source => {
                5 => 'Ticket2',
            },
            Target => {
                4  => 'Ticket 3',
            },
        },
    },
    URL => {
        ExternalLink => {
            Source => {
                1 => 'http://perl.org',
            },
        },
    },
};
```

Listing 2

```
mysql> select * from link_relation;
+------------------+------------+------------------+------------+---------+
| source_object_id | source_key | target_object_id | target_key | type_id |
+------------------+------------+------------------+------------+---------+
|                1 | 120        |                1 | 122        |       2 |
|                1 | 130        |                1 | 131        |       2 |
+------------------+------------+------------------+------------+---------+
```

Listing 3

```
<ConfigItem Name="LinkObject::PossibleLink###0600-ExternalLink" Required="0" Valid="1">
    <Description Translatable="1">Links a ticket with an URL.</Description>
    <Group>LinkExternalURL</Group>
    <SubGroup>Core::LinkObject</SubGroup>
    <Setting>
        <Hash>
            <Item Key="Object1">Ticket</Item>
            <Item Key="Object2">URL</Item>
            <Item Key="Type">ExternalLink</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

Listing 4

```
<ConfigItem Name="LinkObject::Type###ExternalLink" Required="1" Valid="1">
    <Description Translatable="1">
        Defines the link type 'ExternalLink'. If the source name and the target
        name contain the same value, the resulting link is a non-directional one;
        otherwise, the result is a directional link.
    </Description>
    <Group>LinkExternalURL</Group>
    <SubGroup>Core::LinkObject</SubGroup>
    <Setting>
        <Hash>
            <Item Key="SourceName">ExternalLink</Item>
            <Item Key="TargetName">ExternalLink</Item>
        </Hash>
    </Setting>
</ConfigItem>
```

Listing 5

Da die Verknüpfung "ungerichtet" ist, wird für Quelle und Ziel nur ExternalLink als Typ angegeben, siehe Listing 5.

Sind es "gerichtete" Verknüpfungen - wie oben bei den Massenstörungen, muss man sich überlegen, was die Quelle und was das Ziel darstellt.

Neben der Konfiguration werden noch drei weitere Dateien benötigt. Das klingt erstmal nach viel Arbeit, aber bei zwei Modulen kann man viel von bestehenden Modulen übernehmen. Die drei Module, die benötigt werden sind

- Kernel::System::URL
- Kernel::Output::HTML::LinkObjectURL
- Kernel::System::LinkObject::URL

Die erste dieser Dateien ist für die Datenbankaktionen bezüglich der URLs da. Wenn URLs also hinzugefügt, gelöscht oder gesucht werden sollen. Der Code dieses Moduls ist an dieser Stelle uninteressant, für diesen Artikel wichtiger sind dagegen die beiden letztgenannten Module.

Kernel::Output::HTML::LinkObjectURL

Dieses Modul legt fest, wie die Daten angezeigt werden sollen wenn es um die Verknüpfung geht. Gibt es zu einem Ticket verknüpfte Daten, werden diese in einer Tabelle am Ende des Tickets angezeigt (Abbildung 3). Dabei werden für unterschiedliche Objekte auch unterschiedliche Informationen angezeigt. Bei Tickets sind dies z.B. die Ticketnummer, ... und bei externen URLs soll nur der Link angezeigt werden, wobei der User den Titel der Webseite sehen soll.

OTRS kennt für die Anzeige der Verknüpfungen zwei Sorten von Tabellen. In Abbildung 3 ist die einfache Tabelle zu sehen. Es gibt noch eine komplexe Tabelle (Abbildung 4), in der mehr Informationen angezeigt werden können und die auch im Hauptbereich zu sehen ist und nicht nur in der schmalen Spalte. Welche Art der Tabelle angezeigt wird, ist über die SysConfig einstellbar. Unter Framework->Core::LinkObject gibt es die Einstellung "LinkObject::ViewMode". Standardmäßig ist es auf "Simple" gestellt.

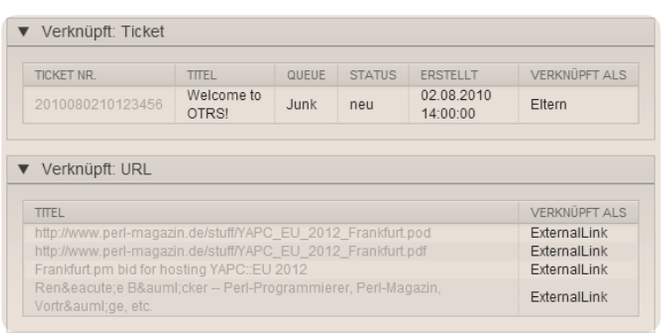


Abbildung 4: Verknüpfungen als komplexe Tabelle

Die Informationen für die Tabellen werden in den Subroutinen `TableCreate*` gesammelt und zur Verfügung gestellt. Für die komplexe Tabelle müssen viele Informationen bereitgestellt werden:

```
# define the block data
my %Block      = (
    Object    => 'URL',
    Blockname => 'Externe Links',
    Headline  => [
        {
            Content => 'Title',
        },
    ],
    ItemList => \@ItemList,
);
```

Die Angabe `Object` bestimmt das Backendmodul, der Blockname ist die Überschrift zu der Teil-Tabelle und bei `Headline` werden die Überschriften der Tabelle angegeben. Für jede Spalte muss ein anonymer Hash übergeben werden. Dabei ist "Content" immer der Schlüssel und der Wert ist die Überschrift.

In der `ItemList` sind dann die einzelnen Verknüpfungen zu finden. In der Liste muss pro Verknüpfung ein anonymer Hash genommen werden:

```
{
    Type    => 'Link',
    Key     => $URLID,
    Content => $URL->{Title},
    Link    => $URL->{URL},
},
```

Als `Type` kann gewählt werden zwischen "Text" und "Link". Ist das Element ein Link, wird die bei `Link` übergebene URL als Ziel und der bei `Content` übergebene Text als Linkbeschreibung herangezogen. Der `Key` dient der Sortierung.

Bei der einfachen Tabelle gibt es keine einzelnen Blöcke, sondern es landet alles in einem Block - wie auf Abbildung 3 zu sehen ist. Um dennoch eine Unterscheidung treffen zu können, zwischen welchen Objekttypen die Verknüpfung besteht, muss das im Hash - der zurückgegeben wird - enthalten sein. In dem Hash werden

```
%LinkOutputData = (
    ExternalLink::Source => {
        URL => [{
           Type    => 'Link',
           Content => 'Perl',
           Title   => 'Die Webseite ...',
           Link    => 'http://perl.org',
        }],
    },
);
```

Der Title ist der Alternativtext, der bei einem `MouseOver` über dem Link angezeigt wird.

In der Funktion `SearchOptionList` wird festgelegt, welche Felder für die Suche nach den Zielobjekten es im Formular (Abbildung 5) gibt. Nimmt man ein bestehendes Modul aus dem Standard als Vorlage, muss nur das Array `@SearchOptionList` angepasst werden.

```
my @SearchOptionList = (
    {
        Key  => 'URL',
        Name => 'URL',
        Type => 'Text',
    },
    {
        Key  => 'Title',
        Name => 'Title',
        Type => 'Text',
    },
);
```

Jedes Element in diesem Array ist ein Suchfeld. Der `Key` ist das Label und der `Name` eben der Name für das Eingabefeld. Als Typen stehen hier "Text" und "List" zur Verfügung. "Text" erzeugt ein einfaches Texteingabefeld, während "List" ein DropDown erzeugt. Für DropDown-Elemente muss in `SearchOptionList` vor dem Aufruf von `BuildSelection` der Wertebereich des DropDowns definiert werden. Im Modul für Tickets heißt es:

```
my %ListData;
if ( $Row->{Key} eq 'StateIDs' ) {

    # get state list
    %ListData =
        $Self->{StateObject}->StateList(
            UserID => $Self->{UserID},
        );
}
```

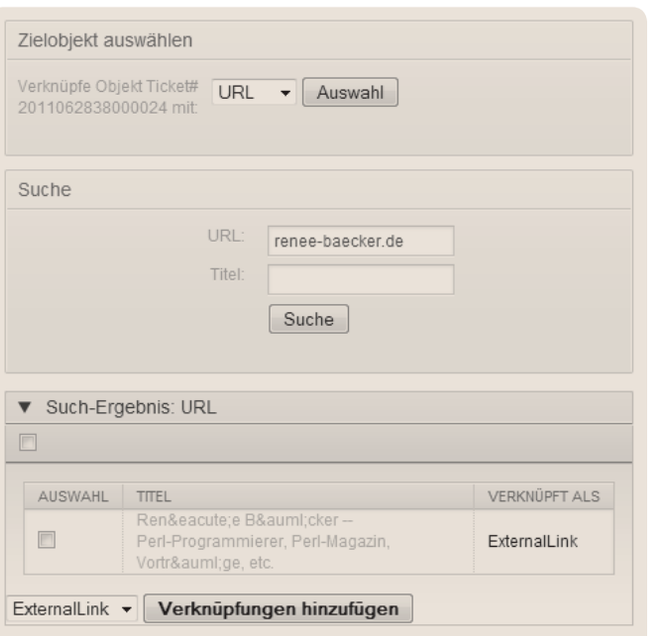
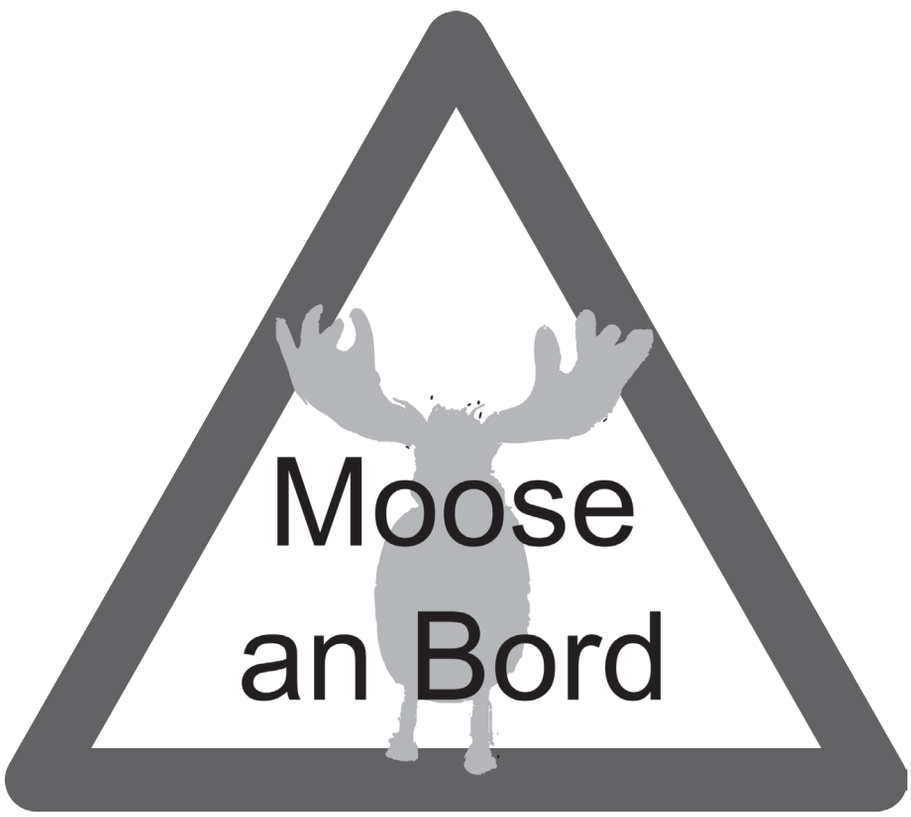


Abbildung 5: Formular um Verknüpfungen herzustellen

Kernel::System::LinkObject::URL

Im Backendmodul für die URLs wird dann die Suche nach den URLs implementiert. Diese Methode muss implementiert sein, da man im Formular nach Objekten suchen kann. Zusätzlich gibt es hier noch die Möglichkeit, auf LinkAdd*- bzw. LinkDelete*-Events zu reagieren. Das kann dann sehr sinnvoll sind, wenn noch zusätzliche Aktionen notwendig sind. Z.B. wenn der Abteilungsleiter informiert werden soll, wenn mit den Verknüpfungen neue Massenstörungen definiert werden.

Das Paket, das URLs als Zielobjekt von Verknüpfungen erlaubt, ist auch auf OPAR unter http://opar.perl-services.de zu finden. Einfach nach "LinkExternalURL" suchen.



Perl-Services.de

Programmierung - Schulung - Perl-Magazin

info@perl-services.de