TECHNICAL MANUAL

iProtect API description

- Date: 13-01-2020
- Version: 2.10

Publication

January 2020,
TKH Security
Paasheuvelweg 20
1105BJ Amsterdam
The Netherlands
https://tkhsecurity.com/
Tel.: +31-20-4620700

Table of content

List of images & tables

Tables

1 Introduction

This document describes the Application Programming Interface (API) for the iProtect Security Management System. It is applicable to iProtect version 7.05 or higher unless stated otherwise.

Chapter 2 of this document describes the XMLSQL interface. This interface allows direct access to the full iProtect database by means of SQL statements,

Chapter 3 describes a number of simple queries for the person, access key and card class tables of the iProtect system

Chapter 4 explains the iProtect event structure and gives a number of useful queries to obtain information from the iProtect events.

Chapter 5 of this document describes actions, procedures and alarm handling.

Chapter 6 of this document describes another web service that makes the collection and interpretation of iProtect events much easier.

Chapter 7 describes the event push mechanism that enables active pushing of transactions from server to client.

2 XMLSQL

To provide a general interface for connecting with iProtect we define a web service called XMLSQL. It's a XML wrapper around SQL statements that can be used to query, update and insert data in the iProtect data base. This interface gives full access to the iProtect functionality without using the iProtect user interface.

2.1 License

External services built on this interface should always have access to the iProtect system. To prevent that such a service cannot log into the iProtect system because all user licenses are in use, a special kind of user license has been introduced (License number **47: External Services**).
By indicating that a System User is an External Service in the System User properties, one can prevent that this license can be occupied by other System Users.

2.2 Authentication

The authentication currently used is form based (see Java™ Servlet Specification, v2.4)

The login procedure is a 3-step process.

Step 1: Obtaining the SESSIONID
The login is established over HTTP(S) by doing the following request:

```
https://iProtect/Webcontrols/xmlsql
POST /Webcontrols/xmlsql HTTP/1.1

[XML request (see XML QuerySQL request)]
```

Note: As from iProtect 9.10 all requests **must** be HTTPS and **must** be directed to the default https port (443). Ports 6060 and 8443 are not available any more.
Also note that the HTTP **POST** method is used, not the GET method so in the browser this request will return an error 400 because the GET method is not supported.

The result of the request is an HTML page. For version 8.4 and earlier the return code is HTTP status = 200 (OK)). For version 9 and later the return code is HTTP status = 302 (MOVED_TEMP).

In the resulting header “Set-Cookie” a JSESSIONID is defined. Use this ID for login and all subsequent requests in this session.

Step 2: Authentication
With the sessionId we can send the username and password to the server. The username and password must be configured in the iProtect system as iProtect users. The access rights of this user on the data elements of the iProtect database will determine the success of the SQL actions performed by this user. If the user has no read access to certain data, a select query on this data will fail.

Actual form login

```
POST /Webcontrols/j_security_check  HTTP/1.1
Connection: Keep-Alive
Cookie: JSESSIONID=[sessionid from Set-Cookie]

&j_username=[loginName]
&j_password=[password]
```

Note that the URL the request is send to is different from the URL in the first step. Now the URL /Webcontrols/j_security_check should be used.

When http return status = 302 (MOVED_TEMP) the login is successful

Step 3: Obtaining the result
After successful login a XML request must be send to the iProtect server (with Connection: Keep-Alive). The response to this request will be the response to the XML query in from the first step.

Actual request:

```
POST /Webcontrols/xmlsql HTTP/1.1
Connection: Keep-Alive
Cookie: JSESSIONID=[sessionid from Set-Cookie]

XML request
```

2.3 Open connection

After the initial authentication new requests can be submitted to the iProtect system without authentication as long as the correct sessionid is used in the request header.
The session remains active as long as requests are send to it. When no requests are submitted for a certain period (about 4 minutes) the iProtect system closes the connection. An open connection trough the XMLSQL interface uses one floating GUI license, so it can be important not to keep open connections unnecessarily.

2.4 Logout

The session can be terminated by sending a logout request to the server.

The logout is done by the following request:

```
POST /Webcontrols/xmlsql HTTP/1.1
Connection: Keep-Alive
Cookie: JSESSIONID=[sessionid from Set-Cookie]

<LOGOUT/>
```

2.5 XML QuerySQL request

The XML payload of a request should always begin with:

```
<?xml version="1.0" encoding="UTF-8"?>
```

For select the request has the following syntax:

```
<QUERY>
        <SQL>
                [sql command]
        </SQL>
</QUERY>
```

For insert/update/delete statements the request has the following syntax:

```
<UPDATE>
    <SQL>
        [sql command]
    </SQL>
</UPDATE>
```

The response:

when an error has occurred (error number != 0):

```
<RESULT error="[error number]">
    <ERROR>
        <DESCRIPTION>
            [Error description]
        </DESCRIPTION>
    </ERROR>
</RESULT>
```

Response for an SQL select statement with n columns, m rows:

```
<RESULT error="0">
    <ROWSET>
        <ROW>
            <[COLUMNNAME 1]>[Value column 1, row 1]<[/COLUMNNAME 1]>
            <[COLUMNNAME 2]>[Value column 2, row 1]<[/COLUMNNAME 2]>
            ...
            <[COLUMNNAME n]>[Value column n, row 1]<[/COLUMNNAME n]>
        </ROW>
        ...
        <ROW>
            <[COLUMNNAME 1]>[Value column 1, row m]<[/COLUMNNAME 1]>
            <[COLUMNNAME 2]>[Value column 2, row m]<[/COLUMNNAME 2]>
            ...
            <[COLUMNNAME m]>[Value column n, row m]<[/COLUMNNAME n]>
        </ROW>
    </ROWSET>
</RESULT>
```

For an SQL insert/update/delete statement:

```
<RESULT error="0">
    <AFFECTEDRECORDS>1</AFFECTEDRECORDS>
</RESULT>
```

2.5.1 Blob data

(Implemented in version 8.01 or higher)
When a field is of type blob (binary large object) normaly a reference id will be returned. To get the blob data in hex presentation (without leading 0x) use <QUERY BLOBASREFID='false'><SQL>[sql]</SQL></QUERY>

Example to obtain the photo of a person:

<QUERY BLOBASREFID='false'><SQL>SELECT PERSONID, BINARYOBJECTID from PERSON where PERSONID=xx</SQL></QUERY>

Example to obtain the passport scan of a person.

<QUERY BLOBASREFID='false'><SQL>SELECT BINARYOBJECTID from PERSONBLOB where PERSONID=xx</SQL></QUERY>

2.6 SQL Syntax

The iProtect API does not support the full SQL syntax. A subset of commands can be used. In this paragraph the supported SQL syntax is described.

2.6.1 Select

SELECT <ColomnListExpression> FROM <JoinListExpresion> WHERE <WhereListExpression> ORDER BY <OrderByListExpression> OFFSET <OffsetValue> LIMIT <LimitValue>

<ColumnListExpression> : <ColumnExpression> , <ColumnExpression>, ...

<ColumnExpression> : <ColumnNameExpression> | <ConstantExpression> | <WildCard>

<ConstantExpression> : '<StringValue>'

<WildCard> : [<TableName>.]*

<JoinListExpression> : <JoinListExpressionSQL89> | <JoinListExpressionSQL92>

<JoinListExpressionSQL89> : <JoinSQL89Expression>, <JoinSQL89Expression>, ...

<JoinSQL89Expression> : <TableNameExpression> | (<JoinListExpressionSQL92>)

<JoinListExpressionSQL92> : [(] [(] <TableNameExpression> <JoinExpressionSQL92> [)]
[<JoinExpressionSQL92> ... [)]

<JoinExpressionSQL92> : <JoinType> <TableNameExpression> ON
<JoinWhereListExpression>

<JoinType> : INNER JOIN | LEFT [OUTER] JOIN | RIGHT [OUTER] JOIN

<TableNameExpression> : <TableName> [[AS] <AliasName>]

<JoinListWhereExpression> : [(] [(] <JoinWhereExpression> [)] [AND
<JoinWhereExpression> ... [)]

<JoinWhereExpression> : <ColumnNameExpression> = <ColumnNameExpression> | <ColumnNameExpression> = <ConstantValue>

<WhereListExpression> : [(] [(] <WhereExpression> [)] [AND <WhereExpression> ... [)]

comment: OR is only allowed when <WhereExpression> = <ColumnNameExpression> = <ConstantValue> and every columnname is the same..

<WhereExpression> : <ColumnNameExpression> <CompareSymbol> <ColumnNameExpression> | <ColumnNameExpression> <CompareSymbol> <ConstantValue> | <ColumnNameExpression> IN (<ConstantValue>, <ConstantValue>, ...)

<CompareSymbol> : = | >= | <= | <

<OrderByListExpression> : <OrderByExpression>[[, <OrderByExpression>], ...]

<OrderByExpression> : <ColumnNameExpression> [ASC | DESC]

<OffsetValue> = < ConstantValue >

<LimitValue> = < ConstantValue >

2.6.2 Update

UPDATE <TableName> SET <UpdateSetListExpression> WHERE <UpdateWhereListExpression>

<UpdateSetListExpression> : <UpdateSetExpression> [, <UpdateSetListExpression>]

<UpdateSetExpression>: <ColumnNameExpression> = <ConstantValue>

<UpdateWhereListExpression> : <UpdateWhereExpression> [AND <UpdateWhereListExpression>]

<UpdateSetExpression> : <ColumnNameExpression> = <ConstantValue>

Remarks:

- The <UpdateWhereListExpression> must exist of a combination of columns that have a combined index or one of the columns is the primary key
- Only one record is updated per request.

2.6.3 Insert

INSERT INTO <TableName> (<InsertColumnListExpression>) VALUES (<InsertValueListExpression>)

<InsertColumnListExpression> : <ColumnNameExpression> [, InsertColumnListExpression]

<InsertValueListExpression> : <ConstantValue> [, InsertValueListExpression]
Delete
DELETE [*] FROM <**TableName**> [WHERE <**DeleteWhereListExpression**>]
<**DeleteWhereListExpression**> : <DeleteWhereExpression> [AND <DeleteWhereListExpression>

<DeleteWhereExpression> : <ColumnNameExpression> = <ConstantValue>

Remarks:

- The < DeleteWhereListExpression> must exist of a combination of columns that have a cobined index or one of the columns is the primary key
- Only one record is deleted per request when the WHERE keyword is used. Without WHERE all records are deleted.

2.6.4 General

<ColumnNameExpression> : [<TableName>.]<ColumnName> [AS <AliasName>]

<ConstantValue> : <StringValue> | <IntegerValue> | <TimeValue> | <HexValue> | NULL
<StringValue> : '<CHARSETCHARS>...'

<IntegerValue>:<DIGIT>...

<TimeValue> : #yyyy-MM-dd hh:mm:ss# | #yyyy-MM-dd# | #hh:mm:ss#

<HexValue> : 0x[HexByte][HexByte]… (version 8.01 or higher)

<HexByte> : 0 = 00, 1 = 01, ..., 255 = ff

<CHARSETCHARS> : [CHARSETCHAR][CHARSETCHAR]...

<CHARSETCHAR> : dependent on the character set op the database, 'and \ are 'escaped' by \' and \\

<DIGIT> : 0..9

2.7 Restrictions

Although a large part of the SQL89 and SQL92 syntax can be used, there are a number of restrictions:

- In the **select** statement there should be an index on the combination of **join**'s and **where**.

So:

SELECT person.name, employee.salarynr FROM person INNER JOIN employee ON employee.personid = person.personid WHERE employee.departmentid = 2

Is allowed because there is an index on *employee.departmentid* and employee.*personid*

SELECT person.name, employee.salarynr FROM person INNER JOIN employee ON employee.personid = person.personid WHERE person.name >= 'a' AND person.name <'azz'

Is allowed because there is an index on *employee.personid* and person.name.

SELECT person.name, employee.salarynr FROM person INNER JOIN employee ON employee.personid = person.personid WHERE person.name >= 'a' AND person.name <'azz' AND employee.balancestatus = 1

Is not allowed because there is no combined index on [employee.personid, employee.balancestatus] or on [person.personid, person.name].

Which indexes exist in the iProtect database can be found in iProtect:

Installation→Database→Table information

- Combinations of OR with AND are currently not allowed.. OR is only allowed when every columname is the same. This is really an alternative for WHERE <columnname> IN (value1, value2, ...)

- LIKE, COUNT keywords are not supported.

2.8 Example

Example of the first request:

```
POST /Webcontrols/xmlsql HTTP/1.1
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_05
Host: 192.168.1.1:443
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 91

<?xml version="1.0" encoding="UTF-8"?>
<query>
      <sql>
            select VERSION from IEDATA
      </sql>
</query>
```

The response for version 8.04 and earlier:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=56C06AB28033A4C0177A76F16ED36B9E; Path=/Webcontrols
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 3470
Date: Thu, 19 Jun 2008 13:39:02 GMT

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http:/
/www.w3.org/TR/html4/strict.dtd">
<html>
       (Stuf deleted)
</html>..
```

The response for version 9.0 and later:

```
HTTP/1.1 302 Found
Server: Apache-Coyote/1.1
Set-Cookie:                    JSESSIONID=C545DD2B29561276466CBC897B5C984E;
Path=/Webcontrols/; HttpOnly
Expires: 0
Pragma: no-cache
```

```
Content-Length: 0
X-XSS-Protection: 1; mode=block
Date: Thu, 19 Jun 2008 13:39:02 GMT
Location: https://212.123.220.176
/Webcontrols/login.html;jsessionid=C545DD2B29561276466CBC897B5C984E
```

Example of the second request:

```
POST /Webcontrols/j_security_check HTTP/1.1
Connection: Keep-Alive
Cookie: JSESSIONID=56C06AB28033A4C0177A76F16ED36B9E
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_05
Host: 192.168.1.1
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-type: application/x-www-form-urlencoded
Content-Length: 40

&j_username=ROOT
&j_password=Geheim
```

The response:

```
HTTP/1.1 302 Moved Temporarily
Server: Apache-Coyote/1.1
Location: https://192.168.1.1/Webcontrols/xmlsql
Content-Length: 0
Date: Thu, 19 Jun 2008 13:39:02 GMT
```

Example of the third request:

```
POST /Webcontrols/xmlsql HTTP/1.1
Cookie: JSESSIONID=56C06AB28033A4C0177A76F16ED36B9E
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.6.0_05
Host: 192.168.1.1
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 91

<?xml version="1.0" encoding="UTF-8"?>
<query>
      <sql>
            select VERSION from IEDATA
      </sql>
</query>
```

The response:

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
Content-Type: text/xml
Transfer-Encoding: chunked
Date: Thu, 19 Jun 2008 13:39:02 GMT

<?xml version="1.0" encoding="UTF-8"?>
<RESULT error="0">
       <ROWSET>
       <ROW><VERSION>7.02.03 - 08/05/30</VERSION></ROW>
       </ROWSET>
</RESULT>
```

3 Simple Person and AccessKey queries

As a simple example of the use of SQL-statements to obtain information from the iProtect database we give a short explanation of access key and person tables in iProtect.

3.1 Person

iProtect stores general information related to persons in the table PERSON. This table is used to store person-related information for card holders, visitors, employees, system users, etc.

The most important columns of the person table are:

- firstname
- prefix
- name: (lastname)
- homeaddress
- homezip
- homecity
- personid: a unique number that (internally) identifies a set of person data.

If the personid is known the above information can be obtained from the database using the following query:

```
SELECT firstname, prefix, name, homeaddress, homezip, homecity FROM person WHERE personid=x
```

3.2 AccessKey

In the iProtect system different types of access keys can be used at the same time (e.g. card and license plate). This means that a card number (RCN) does not have to be unique in the system. Every type of access key is defined by a so-called CARDCLASS. To identify a unique key both the card number (RCN) and the CARDCLASS are required.

All defined cardclasses can be obtained with the command:

```
SELECT cardclassid, cardclassname FROM cardclass
```

The cardclassid is a number that is dependent of the specific iProtect implementation and different for each installation. To avoid this problem we provided a cardclass parameter “code”, which is a user-defined value that also can be used in queries (it is indexed).

NOTE: The code parameter can be used in many tables (READER, OUTPUT, ALARMACTION, etc) to identify in a database independent way the objects in the iProtect database. This code is specifically designed for external interfaces.

For example, if you give the cardclass “license plate” in iProtect the code “5” than we can obtain the cardclassid from the license plate card class with the query:

```
SELECT cardclassid, cardclassname FROM cardclass WHERE code=5
```

If you know the card number (=RCN=string variable !!) and the cardclassid of a certain access key, you can obtain all information about this access key with a query like:

```
SELECT accesskeyid, personid, valid, startdate, enddate, unlimited FROM
accesskey WHERE cardclassid=x and rcn= 'y'
```

We obtain the following information:

- accesskeyid: internal unique ID of the access key (long)
- personid: internal ID of the person that owns the access key. If this value is NULL the access key has not been assigned to a person.
- valid: indicates whether the access key is valid (boolean)
- startdate: start date of the validity of the access key.If this value is NULL than the access key is valid until the end date (date)
- enddate: end dat of the validity of the access key. If the value is NULL the access key is not valid.
- unlimited: if this value is TRUE the startdate and enddate are ignored and the access key is always valid(boolean)

To make an access key invalid, the following SQL command can be used:

```
UPDATE accesskey SET valid=0 where accesskeyid=x
```

3.3 Keygroup

The access rights in the iProtect system are controlled by combinations of card groups, time zones and reader groups (Admittance). Usually, these admittance combinations are defined during the configuration of the system.
Giving a certain card/key specific access rights is done by adding this key to the correct keygroup.

If the id of the key is *x* and the id of the keygroup the key is added to is *y* the command to add the key to this keygroup is:

```
INSERT INTO keykeygroup (accesskeyid, keygroupid) VALUES (X,Y)
```

3.4 Uploading a photo

A photo of a person can be uploaded as a hexadecimal binary object with the update statement.

Example:

```
update PERSON set BINARYOBJECTID=0x8950A3…FF where personid=X
```

`0x8950A3…FF` is the hexadecimal representation of the photo.

3.5 Uploading a passport scan

It is possible in iProtect to link addition images to a person. This mechanism is handled with so-called FREEBLOB in the PERSONBLOB table.
Which FREEBLOB instance gets updates (you can have up to 4 for a person) is determined in the CUSTOMTEXT table via the CUSTOMTEXTID.

Example:

We have defined the passport scan as FREEBLOB1 and linked this to the person table. This definition is in the table CUSTOMTEXT with CUSTOMTEXTID=26.
We now can add the passport image to person with PERSONID 13 with the following update statement:

```
Insert into PERSONBLOB (PERSONID, CUSTOMTEXTID, BINARYOBJECTID)
values (13,26,0x89AA0D..FF)
```

Or alternatively (for 8.02 and higher):

```
<UPDATE>
  <SQL>update personblob set personid=13, customtextid=26,
binaryobjectid=?</SQL>
  <BLOBS>
    <BLOB>[base64 encoded blob1 data]</BLOB>
  </BLOBS>
</UPDATE>
```

4 Event structure

Events (transactions) in iProtect™ Aurora have to contain many kinds of information. Therefore the event mechanism has been designed in a flexible fashion. Information about an event is stored in several different tables of the iProtect™ Aurora database. This implies that for retrieving information about a specific event from the database you have to take different steps.

This document describes the structure of the events in iProtect™ Aurora for enabling you to reconstruct events.

4.1 Event notification definition (‘TRANSDEF’)

First of all there is the table TRANSSETUP (if language specific setting: ‘Event settings’) which defines different *event types* (transaction types). The textual meaning of an event type can be found in the table USERTEXT (if language specific setting: ‘Language specific text’) and can be retrieved with query:

```
SELECT * FROM USERTEXT WHERE NAME='TRANSTYPE'
```

In iProtect™ Aurora you can define for each event type separately which information will be stored with the event. It will be determined in table TRANSDEF (‘Event notification definition’). This table obtains for each event type several rows in which is determined, how the information can be retrieved. For each row ONE parameter will be described by a reference to an other event table. For the real storage of event data 3 tables are used: TRANSTABLE, TRANSDATA and TRANSNUMBER.

If a row refers to a TRANSNUMBER or TRANSDATA table, one will find the value in DATAID. The textual meaning can be retrieved with

```
select * from usertext where name=transvalueid
```

For explaining this mechanism we have to look at the (default) event notification definition of a common event: "Access for subscriber card", event type 1.

The rows in TRANSDEF which define this event are:

TRANSDEFID	TRANSTYPE	NAME	TABLENAME	DATAID	SYSTABLESID
104	1	RCN CARD NR	TRANSDATA	29	(null)
103	1	PERSON	TRANSTABLE	(null)	19
102	1	ACCESSKEY	TRANSTABLE	(null)	26
101	1	READER	TRANSTABLE	(null)	15
100	1	KEYUSE	TRANSTABLE	(null)	25
106	1	CAMERA	TRANSTABLE	(null)	229
105	1	CARDCLASS	TRANSTABLE	(null)	222

Table 1 Event definition

The first row (104) indicates that the RCN card number belonging to this event can be retrieved in the TRANSDATA table with DATAID "29".

You can retrieve the other data in the TRANSTABLE table. The TRANSTABLE table obtains references to other tables of the iProtect™ Aurora database (via SYSTABLEID) where the necessary data are stored. The persons in the table with SYSTABLEID=19 and the cards in the table with SYSTABLEID=26.

The link between SYSTABELID's and the table names are put into the TABLENAMES table.

For "Access for subscriber card" events no references to the TRANSNUMBER table are used. In this table numeric values are saved which belong to the event.

4.2 Event reconstruction

With above mentioned information one can retrieve all information from a specific event.
The events are saved as unique entities in the TRANSACTION table.
As example we can take a " Access for subscriber card " event (TRANSTYPE=1) of 3rd of august 2007 at 15:08 (wintertime). The row in the TRANSACTION table is:

TRANSID	TRANSTYPE	TIME
11423389	1	2007-08-03 15:08:10.0

Table 2 Obtaining transid

With TRANSID=11423389 it is possible now to retrieve the necessary information in the tables TRANSDATA and TRANSTABLE:

```
SELECT * FROM TRANSDATA WHERE TRANSID=1143389
```

results in:

TIME	DATAID	VALUE	TRANSID
2007-08-03 15:08:10.0	29	0001	11423389

Table 3 Obtaining transaction data

```
SELECT * FROM TRANSTABLE WHERE TRANSID=1143389
```

results in:

TIME	TABLEID	PRIMKEY	TRANSID
2007-08-03 15:08:10.0	15	32	11423389
2007-08-03 15:08:10.0	19	434	11423389
2007-08-03 15:08:10.0	25	1	11423389
2007-08-03 15:08:10.0	26	105	11423389
2007-08-03 15:08:10.0	222	5	11423389

Table 4 Obtaining table data

This indicates the location of the data in the different tables.

- The reader which is involved in this event, is the reader defined with primary key "32" in table "15" (=READER table).
- The person which is involved in this event, is the person defined with primary key "434" in table "19" (=PERSON table).
- The card usage which is involved in this event, is the card usage defined with primary key "1" in table "25" (=KEYUSE table).
- The card which is involved in this event, is the card defined with primary key "105" in table "26" (=ACCESSKEY table).
- The card coding which is involved in this event, is the card coding defined with primary key "5" in table "222" (=CARDCLASS table).

4.3 Event types

For retrieving the event types of different events in iProtect™ Aurora you can view the event list in iProtect™ Aurora:

General->Settings->Events

In this list you can find the event type (number) and the event notification definition.

4.4 Useful queries

The following query retrieves the point of time and person's id of certain events younger than a certain date (transtype=111 time booking; tableid=19 Person table).

```
SELECT TRANSACTION.TIME, TRANSTABLE.PRIMKEY FROM TRANSTABLE INNER JOIN
TRANSACTION ON TRANSACTION.TRANSID=TRANSTABLE.TRANSID WHERE
TRANSACTION.TRANSTYPE=111 AND TRANSTABLE.TABLEID=19 AND TRANSTABLE.TIME >=
#2008-02-24#
```

For retrieving the booking categories (tableid=13):

```
SELECT TRANSACTION.TIME, TRANSTABLE.PRIMKEY FROM TRANSTABLE INNER JOIN
TRANSACTION ON TRANSACTION.TRANSID=TRANSTABLE.TRANSID WHERE
TRANSACTION.TRANSTYPE=111 AND TRANSTABLE.TABLEID=13 AND TRANSTABLE.TIME >=
#2008-02-24 00:00:00#
```

If the PersonID is known retrieving the name and the department:

```
SELECT PERSON.FIRSTNAME, PERSON.NAME, DEPARTMENT.NAME FROM PERSON INNER
JOIN DEPARTMENT ON PERSON.DEPARTMENTID = DEPARTMENT.DEPARTMENTID WHERE
PERSON.PERSONID=personid
```

If the PersonID is known retrieving the employee number:

```
SELECT PERSON.FIRSTNAME, PERSON.NAME, EMPLOYEE.SALARYNR FROM PERSON
INNER JOIN EMPLOYEE ON PERSON.PERSONID = EMPLOYEE.PERSONID WHERE
PERSON.PERSONID=personid
```

Above mentioned queries also can be combined:

```
SELECT PERSON.NAME, EMPLOYEE.SALARYNR, DEPARTMENT.NAME  FROM PERSON
INNER JOIN EMPLOYEE ON PERSON.PERSONID = EMPLOYEE.PERSONID INNER JOIN
DEPARTMENT ON PERSON.DEPARTMENTID = DEPARTMENT.DEPARTMENTID  WHERE
PERSON.PERSONID=681
```

If the events are displayed in text like in the iProtect™ Aurora event list (table TRANSVIEW):

```
SELECT TRANSVIEW.TIME, TRANSVIEW.TEXT FROM TRANSVIEW INNER JOIN
TRANSACTION ON TRANSACTION.TRANSID=TRANSVIEW.TRANSID WHERE
TRANSACTION.TRANSTYPE=111 AND TRANSVIEW.TIME >= #2008-02-24#
```

5 Actions, Procedures and Alarms

5.1 Actions

In iProtect Actions are elementary operations that the iProtect system can execute. This can be operations like sending a message to a user, activating an output, setup an intercom connection, start an URL, etc,etc…

All types of actions can be retrieved by the query:

```
SELECT * FROM USERTEXT WHERE NAME='ACTIONTYPE'
```

We can define actions of a certain type with extra parameters that control the action that will be carried out. For example: the id of the intercom stations that has to be connected, or the id of the output that has to be activated.

All Actions are stored in the **ALARMACTION** table.

Actions can be automatic: they are executed immediately, or they have to be confirmed: they are only executed if the user confirms the action.

5.2 Procedures

One or more Actions can be grouped into Procedures. When a procedure is executed all action in that procedure are executed one by one.

There are 2 different ways in iProtect that procedures can be started.

- Procedures can be started if certain conditions are met (conditional)
- Procedures can be started if a certain type of event occurs.

The conditional procedures are stored in the table **ALARMDEF.**

The connection between with actions are used in which procedures is defined in the table **ALARMDEF_ACTION**.

Example: We have a procedure with ID=18. To find all the names of the actions that will be executed by this procedure we can do the query:

```
SELECT ADA.ACTIONNUMBER, AD.NAME, AA.LAN1TEXT FROM
ALARMDEF_ACTION as ADA INNER JOIN
ALARMDEF as AD on ADA.ALARMDEFID=AD.ALARMDEFID INNER JOIN
ALARMACTION as AA on ADA.ACTION=AA.ALARMACTIONID
WHERE ALARMDEFID=18 ORDER BY ADA.ACTIONNUMBER
```

The sequence of actions that are executed when an event occurs are stored in the table **TRANSSETUP_ACTION**.

5.3 Alarm handling

Does a procedure contain actions that require user input or confirmation, a socalled alarm is generated and an entry is added to the ACTIVEALARM table referring to this procedure. In the iProtect user interface the associated alarm shows up in the alarm handler.

The user can then execute the actions and acknowledge the different steps in the procedure and thus handle the alarms.

After the alarms have expired or have been handled they are removed from the ACTIVEALARM table.

With the iProtect API it is also possible to do the alarm handling from an external system, instead of in the user interface. For this we use a (virtual) table: **CORRECTION**. Records of a certain type and with additional parameters can be added to this table and this results in the execution of commands by the database.

In general the statement is:

```
INSERT INTO CORRECTION
 (CORRECTIONTYPE,ALARMDEFID,
ACTIVEALARMID, ALARMACTION, TEXTARG)
VALUES (X,Y,Z…)
```

The different correction types correspond to different actions that can be performed on the alarms.

CORR_ALARM_ACCEPT=3100
This is to indicate to system that this operator will 'handle' this procedure [for multi operator environment]. Parameter needed activealarmid. Note that in the 'tcpconnectionid' column of the activealarm table the "connection/user" is recorded who will handle the alarm. If the user logs out or connection is dropped the alarm will be 'released' again (see also unaccept).

CORR_ALARM_UNACCEPT=3101
This is to indicate to system that this operator will NO longer 'handle' this procedure [for multi operator environment] and should be available for the 'general' pool again. Parameter needed activealarmid.

CORR_ACK_HARD=10001
With this operation the operator acknowledge a hard alarm (see reset column active alarm [32-63]). This we change the status to softalarm. Parameter needed activealarmid.

CORR_RESET_ALARM=10002
This enables the operator to reset this alarm. All pending action(s) will be 'cancelled' and the alarm is reset. Parameter needed activealarmid.

CORR_RESET_ALL=10004
This reset all pending procedures [normally only used after system upgrades etc].

CORR_DONE_ACTION=10003
To indicate that an action (has/)should be done (by operator). Parameter needed activealarmid, alarmaction

CORR_CANCEL_ACTION=10005
To indicate that an action will not be done. Parameter needed activealarmid, alarmaction.

CORR_FAILED_ACTION=10006
To indicate the action is tried but failed. Parameter needed activealarmid, alarmaction.

CORR_INPROG_ACTION=10010

To indicate that the operator has started with performing this task/action. Parameter needed activealarmid, alarmaction.

CORR_DONE_ACTION_PHONE=10007

See also action done, but this one enables to add an extra parameter (phonenr) from the name field of the table.

CORR_FAILED_ACTION_PHONE=10008

See also failed action. but this one enables to add an extra parameter (phonenr) from the name field of the table.

CORR_INPROG_ACTION_PHONE=10009

See also inprog(ress) action. but this one enables to add an extra parameter (phonenr) from the name field of the table.

5.4 Executing a procedure

With the correction mechanism as described in the previous section it is also possible to trigger a procedure from an external system via this API.

First step (Preparation)

Add a unique code to the procedure in iProtect to get a reference that is independent of the database primary index.

Second step

Find the alarmdefID of the procedure, in this case with code 123

```
SELECT alarmdefid from alarmdef where code=123
```

Example response alarmdefid =95

Third step

Start the procedure by inserting the proper correction (10025) in the correction table.

```
insert into correction (correctiontype, alarmdefid) values (10025, 95)
```

6 Service requests

The XMLSQL service as described in chapter 2 is a very powerful and flexible interface that gives full access to the iProtect database. But for some data structures (like Events) this can become rather complex.
Therefore, another mechanism has been added to the interface: Service Requests. This interface will resolve all the references to other data structures and allows the user to obtain the relevant parameters directly.

6.1 Syntax

For service request (i.e. predefined data requests) the request has the following syntax:

```
<SERVICE name='[Service name]'>
    <PARAMETERS>
        <PARAMETER name='Parameter 1 name']>[Parameter 1 value]</PARAMETER>
        <PARAMETER name='Parameter 2 name']>[Parameter 2 value]</PARAMETER>
        ...
        <PARAMETER name='Parameter N name']>[Parameter N value]</PARAMETER>
    </PARAMETERS>
</SERVICE>
```

The **Service name** defines the type of data request that is send to the iProtect service. Currently, only the transactions (events) are supported by the Service name “'transactionshared.transactionView”. For iProtect 8.0 and higher the service name should be: “dataservice/transaction.getView”

The **Parameter names** define the filters for the data that is returned by this service request. The following parameters are supported for 'transactionshared.transactionView:

- columnList: this is a list with column names of the data elements that will be return. Only rows will be returned that contain elements with this columnname.
- filterTransTypeList: this is a list with transaction types that will be returned.
- fromDate: start date for the filter
- endDate: end date for the filter
- descending: reverse to order of the transactions returned
- offset: skip the first number of transactions
- limit: limit the number of transactions returned

6.2 Example

The following example returns the transaction time, type and unique id (standard), full name, card number and reader name of all successful entries of a certain day.

```
Full name=PERSON.FIRSTNAME,PERSON.PREFIX,PERSON.NAME
Card number= ACCESSKEY.RCN
Reader name=READER.NAME
Access transaction types= 1,2

<SERVICE name=’dataservice/transaction.getView’>
    <PARAMETERS>
        <PARAMETER name='columnList'>
            PERSON.FIRSTNAME,
            PERSON.PREFIX,PERSON.NAME,
```

```
                ACCESSKEY.RCN,READER.NAME
        </PARAMETER>
        <PARAMETER name='filterTransTypeList'>
                1,2
        </PARAMETER>
        <PARAMETER name='fromDate'>
                2012-11-14 00:00:00
        </PARAMETER>
        <PARAMETER name='toDate'>
                2012-11-15 00:00:00
        </PARAMETER>
    </PARAMETERS>
</SERVICE>
```

The result of this request will be something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<RESULT error="0">
<ROWSET>
    <ROW>
        <TRANSID>28876431</TRANSID>
        <TYPE>2</TYPE>
        <TIME>2012/11/14 06:35:15</TIME>
        <PERSON.FIRSTNAME>Marco</PERSON.FIRSTNAME>
        <PERSON.PREFIX>van der</PERSON.PREFIX>
        <PERSON.NAME>Linden</PERSON.NAME>
        <ACCESSKEY.RCN>0186</ACCESSKEY.RCN>
        <READER.NAME>1.3.1 - Hoofdentree IN</READER.NAME>
    </ROW>
    <ROW>
        <TRANSID>28876521</TRANSID>
        <TYPE>2</TYPE>
        <TIME>2012/11/14 07:44:24</TIME>
        <PERSON.FIRSTNAME>Sander</PERSON.FIRSTNAME>
        <PERSON.PREFIX></PERSON.PREFIX>
        <PERSON.NAME>Zonneveld</PERSON.NAME>
        <ACCESSKEY.RCN>0133</ACCESSKEY.RCN>
        <READER.NAME>1.3.1 - Hoofdentree IN</READER.NAME>
    </ROW>
    <ROW>
        <TRANSID>28878083</TRANSID>
        <TYPE>2</TYPE>
        <TIME>2012/11/14 14:18:12</TIME>
        <PERSON.FIRSTNAME>Wilfred</PERSON.FIRSTNAME>
        <PERSON.PREFIX></PERSON.PREFIX>
        <PERSON.NAME>Janssen</PERSON.NAME>
        <ACCESSKEY.RCN>0216</ACCESSKEY.RCN>
        <READER.NAME>1.1.1 - Hal &gt; Verkoop bg</READER.NAME>
    </ROW>
</ROWSET>
</RESULT>
```

7 Event Push

The interfaces described in chapter 2 and 5 are interfaces where the client side has to poll for the information and the iProtect server replies with the requested data.
For events, in certain situations, it is better to use a push mechanism for the data exchange.
In iProtect 8.0 this event push mechanism has been implemented.

This mechanism has been implemented conform the HTML5 Server-Sent Event method (http://www.w3schools.com/html/html5_serversentevents.asp?output=print).

7.1 Push NewTransaction

The first service in push will be the push of new transaction arrived in de database. This service can be used to either 'life' replicate the transaction database or to supply the client application with the events happening in the iProtect system.

This servlet/service is started with (using an authenticated session, see chapter 2.1):
https://iProtect/Webcontrols/PushEvent?command=NewTransaction[&parameters]

If no parameters are defined, all transactions will be pushed that are generated by the system after the connection is established. The transactions are send in textual form as they are represented in iProtect.

The **Parameter names** define the filters for the data that is returned by this service request. The following parameters are supported for:

- Last-Event-ID: When this value is positive it indicates the ID of the first transaction that will be pushed to the client. When the value is negative it indicates the most recent transactions that will be pushed before new transactions will be pushed. (so Last-Event-ID=-2 means that the 2 most recent transaction will be pushed before new transactions will be pushed).
- columnList: this is a list with column names of the data elements that will be returned. Only events will be pushed that contain elements with this columnname.
- filterTransTypeList: this is a list with transaction types that will be pushed.

The response consists of meta data and transactions in XML format. The meta data start with ":". These are (conform HTML5 standard):

:retry: = retry interval in ms for the connection
:keep-alive = periodic heart beat to keep the connection alive
:id = the unique identified of an transaction (iProtect event)
:event: Type of event. Now only NewTransaction is supported
:data: The XML representation of the event.

7.2 Examples

The following request opens a connection that pushes all transactions/events that are generated after the connection is established.

```
https://iprotect/Webcontrols/PushEvent?command=NewTransaction
```

The response will be something like:

```
retry: 3000

id: 28913533
```

```
event: NewTransaction

data: <ROW>
      <TRANSID>28913534</TRANSID>
      <TRANSTYPE>29</TRANSTYPE>
      <TIME>2012/11/19 19:40:33</TIME>
      <TEXT>Input actief 17.2.1 - Uitrit DA </TEXT>
</ROW>

id: 28913535

event: NewTransaction

data:
<ROW>
      <TRANSID>28913535</TRANSID>
      <TRANSTYPE>131</TRANSTYPE>
      <TIME>2012/11/19 19:40:33</TIME>
      <TEXT>Alarmactie Zet Uitrit Blok A (KP) ConTZ Trigger: DA Uitrit Actief
      Auto-UIT </TEXT>
</ROW>

id: 28913536

event: NewTransaction

data:
<ROW>
      <TRANSID>28913540</TRANSID>
      <TRANSTYPE>131</TRANSTYPE>
      <TIME>2012/11/19 19:40:33</TIME>
      <TEXT>Alarmactie Puls open Slagboom Uitrit ConTZ Trigger: DA Uitrit Actief
      Auto-UIT </TEXT>
</ROW>

id: 28913547

event: NewTransaction

data:
<ROW>
      <TRANSID>28913547</TRANSID>
      <TRANSTYPE>30</TRANSTYPE>
      <TIME>2012/11/19 19:40:37</TIME>
      <TEXT>Input inactief 17.2.1 - Uitrit DA </TEXT>
</ROW>
```

The following request opens a connection that pushes all transactions successful access transactions (type 1,2), starting with the 2 most recent transactions before the connection was opened. For all transactions the reader name, and the person name are returned (besides the full text).

```
      https://iprotect/Webcontrols/PushEvent?command=NewTransaction&
      Last-Event-ID=-2&language=&columnList=reader.name,
```

```
      person.firstname,person.prefix,person.name&filterTransTypeList=1,2
```

The response will be something like:

```
retry: 3000

id: 28913522

event: NewTransaction

data:
<ROW>
      <TRANSID>28913522</TRANSID>
      <TYPE>2</TYPE>
      <TIME>2012/11/19 19:39:17</TIME>
      <TEXT>Naar Buiten voor Toegang bij 1.8.1 - Hoofdentree UIT voor Kees
      Kraakmans met taq 2: JQQ-KP  0120 Keyuse Medewerker </TEXT>
      <READER.NAME>1.8.1 - Hoofdentree UIT</READER.NAME>
      <PERSON.FIRSTNAME>Kees</PERSON.FIRSTNAME>
      <PERSON.PREFIX></PERSON.PREFIX><
      PERSON.NAME>Kraakmans</PERSON.NAME>
</ROW>

id: 28913472

event: NewTransaction

data:
<ROW>
      <TRANSID>28913472</TRANSID>
      <TYPE>2</TYPE>
      <TIME>2012/11/19 19:24:03</TIME>
      <TEXT>Naar 01-BG-R bij 1.1.1 - Hal &gt; Verkoop bg voor Kees
      Kraakmans met taq 2: JQQ-KP  0120 Keyuse Medewerker </TEXT>
      <READER.NAME>1.1.1 - Hal &gt; Verkoop bg</READER.NAME>
      <PERSON.FIRSTNAME>Kees</PERSON.FIRSTNAME>
      <PERSON.PREFIX></PERSON.PREFIX>
      <PERSON.NAME>Kraakmans</PERSON.NAME>
</ROW>

:Keep-Alive

id: 28913602

event: NewTransaction

data:
<ROW>
      <TRANSID>28913602</TRANSID>
```

```
      <TYPE>2</TYPE>
      <TIME>2012/11/19 20:05:51</TIME>
      <TEXT>Naar Buiten voor Toegang bij 1.8.1 - Hoofdentree UIT voor
      Maarten  Postbode met taq 16: Oxy  0011 Keyuse Sec </TEXT>
      <READER.NAME>1.8.1 - Hoofdentree UIT</READER.NAME>
      <PERSON.FIRSTNAME></PERSON.FIRSTNAME>
      <PERSON.PREFIX>Maarten</PERSON.PREFIX>
      <PERSON.NAME>Postbode</PERSON.NAME>
</ROW>

:Keep-Alive
```