Customer Relationship Management

CRM Interface Administrator Guide

Version 14.1

Aurea

Unlimited Possibility

Notices

Table of Contents

Preface

For details, see the following topics:

About this documentation

This guide is part of the documentation set for Aurea CRM.

Notation conventions

This document uses the following notation conventions:

Convention	Meaning
`Fixed-width`	`Fixed-width` font indicates code, path names, file names, environment variable names, parameter names, command names, machine names, URLs.
`Bold Fixed-width`	**`Bold Fixed-width`** font is used to indicate user input or to emphasize certain lines of code.
Italic Fixed-width	*Italic Fixed-width* font indicates a placeholder for which you must supply a value.
Bold Sans serif	**Bold sans serif** typeface indicates the names of graphic user interface elements such as dialog boxes, buttons, and fields.
Italic serif	In text, *italic serif* typeface indicates the first use of an important term. The term is defined in the glossary.
Underlined	Underlined text in command lines and parameter descriptions indicate that you only have to enter the underlined part of the command or parameter name. For example, if you use the`-LOGFILE` parameter in a command, you only need to enter `-LOGF`.
[ ]	Brackets enclose optional arguments.
{ *a* | *b* | *c* }	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify a or b or c.

Convention	Meaning
...	Three consecutive periods indicate that you can repeat the immediately previous item. In code examples, they can be horizontal or vertical to indicate omissions.
Menu > Choice	An angle bracket between two menu items indicates that you should choose an item from a menu. For example, the notation **File** > > **Exit** means: "Open the **File** menu and choose **Exit**."
>>	Links to related information in other chapters or documents are indicated using the **>>** symbol.

Aurea global support

If you encounter a problem while using an Aurea product or require assistance with downloading the software or upgrading a product release, please open a ticket on Aurea Support Central. Preferably, search the articles on the Aurea Knowledge Base for solutions to your issues before opening a ticket.

Information about the support organization is available on Support Central. The product documentation is available athttps://help.aurea.com/crm/#.

For information about purchasing an upgrade or professional services, contact your account executive. If you do not know who your account executive is, or for other queries, contact us through our website.

1

Introduction

This document gives an overview of the architecture of CRM.interface and contains a full syntax reference for the interface XML file that holds the configuration.

This document does not cover the description of technologies such as XML or XSL/t, for details. refer to the documentation of these technologies.

Please contact the Aurea support center for feedback on this document. This document is published in the download section of the support area of https://support.aurea.com. You can also check for newer revisions.

Architecture Overview & Brief Description

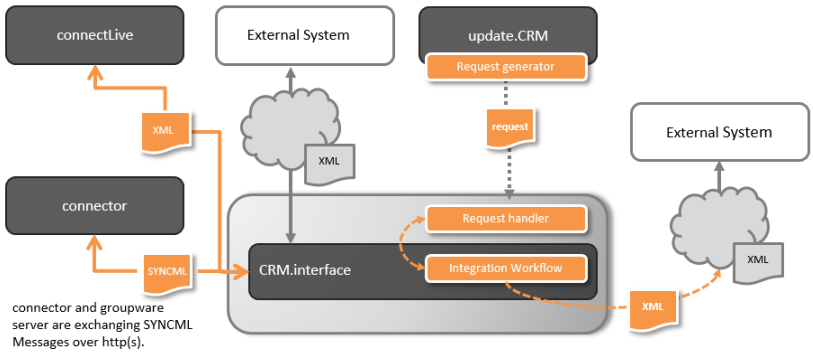
CRM.interface allows access to Aurea CRM business logic and data via XML messages.

Data can be exchanged bi-directionally with any application capable of passing XML requests via HTTP. A powerful transformation engine based on XSL/t allows interface to produce and process any given XML-based dialect.

CRM.interface is most commonly used for on-line integration of back- and front-end systems such as ERP, DMS, e-commerce, product configuration and legacy applications.

Additionally CRM.interface acts as the groupware Server, which is a central part of the Aurea CRM connector product family and connectLive.

Figure 1: Architecture Overview



Note: As of Service Pack 1 CRM.interface not only can act as an integration server, but also as an integration client and actively trigger interactions (e.g. sending http messages) with 3rd party systems.

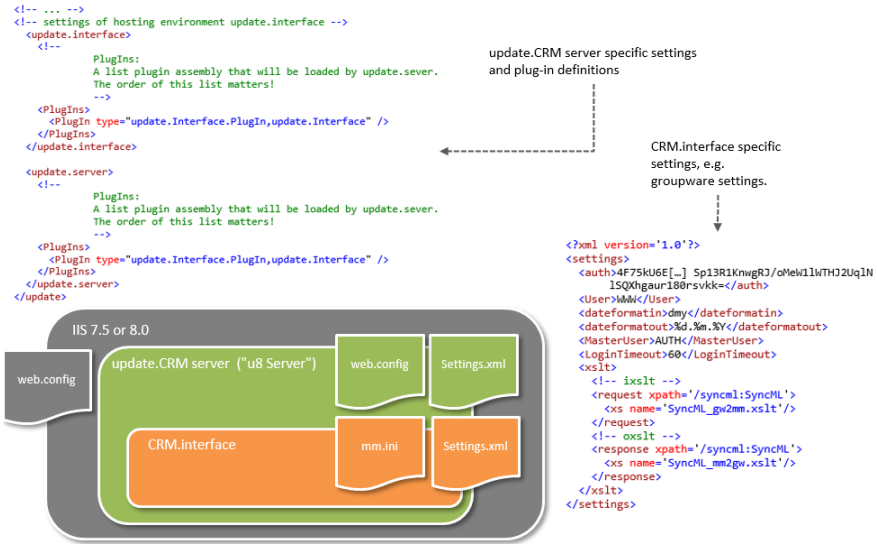
Note: See topic Integration Hub on page 16 for further details.

CRM.interface – Some Technical Details

CRM.interface is a plug-in in the Aurea.CRM server framework or CRM.interface is part of the homogenized server platform of Aurea.CRM.

As such a plug-in CRM.interface lives in IIS (and not as Aurea CRM interface in a special host application).

Figure 2: CRM.interface plug-in



Basically the behavior of CRM.interface is defined via two settings files – `settings.xml` (which can be configured via the CRM.interface configuration tool) and the `mm.ini` file holding some additional CRM.interface specific log settings.

- See also Configuration of CRM.interface for further information
- See also Main differences between CRM.interface and update.seven interface on page 11 for further information.

What's new

Find out what's new here!

New features with Service Pack 2

Integration with 3rd party applications requiring a login

While the initial release of the integration workflow only covered for simple integration scenarios, where either no authentication is required or the 3rd party system (e.g. web services) offers the possibility to pass along the credentials with every request, it is now possible to integrate with an application that requires an explicit login.

See topic How to integrate with a 3rd party system which requires a login on page 24 for further details.

New features with Service Pack 1

As of Service Pack 1 CRM.interface not only can act as an integration server, but also as an integration client and actively trigger interactions (e.g. sending http messages) with 3rd party systems.

See topic Integration Hub on page 16 for further details.

New features with version 8.1.7.433

Enhancements of the "synchronize" workflow

As of version 8.1.7.433 and higher the "synchronize" workflow natively supports exchanging messages with SOAP web services. You now can easily achieve integrations with SOAP web services endpoints. For further details see the topics Workflow Execution steps and Workflow settings and parameters. .

Additionally it is now possible to globally active logging for all workflow activities – see topic Logging on page 25 below.

Configuration of the technical user ("shadow user")

As of version 8.1.7.433 and higher the credentials (username and password) of the technical user (also known as "shadow user") are not longer stored in the `settings.xml`, but in an encrypted XML - `users.xml`. See topic Configuration of the technical user, creation of users.xml for further details.

Excursion_Synchronization of Contact Persons and Persons

As of this version it is possible to also synchronize person records in addition to company and contact person records.

Note: this feature required adoptions of `forms.xml` and `syncml_mm2gw.xslt` - so make sure that you incorporate these changes into your existing style sheets after installing the path. `forms.xlm`: `<table tablename='Human'>` is added to both `<form type='vevent'> and <form type='mmPerson'>.`

Note: `syncml_mm2gw.xslt`: the Person template is extended to `<xsl:template match='syncml:Person|syncml:Human'>`

Main differences between CRM.interface and update.seven interface

Learn about the differences between CRM.interface and update.seven interface.

- The syntax of CRM.interface is fully compatible with the predecessor version Aurea CRM interface (SP8).
- CRM.interface now lives in IIS and no longer in Aurea CRM http server (which is no longer part of the product).
- CRM.interface includes the groupware Server, which is essentially for CRM.connector SE for Exchange and CRM.connector for Domino.
- CRM.interface is no longer a COM Object, therefore no registration of the component is required and therefore you are no longer limited to 10 instances (mmInterface1.dll to mmInterface9.dll) of interface.
- Aurea CRM http client has been substituted by the new CRM.interface integration client.
- The dictionary is no longer used in Aurea CRM; instead the XML names are read from the Aurea CRM data model (field "XML field name"). Therefore the `<dictionary />` command serves for informational purposes only.

2

Installation

Learn how to install CRM.interface and configure it. You can also find the post installation steps discussed here.

Setup Wizard

Aurea CRM software provides an installation wizard that guides you through the installation of CRM.interface.

CRM.interface also can act as groupware server, which is the essential server product for the CRM.connector products (CRM.connector SE for Exchange and CRM.connector for Domino) and CRM.connectLive.

Check the system requirements on https://support.aurea.com for further information on supported environments.

The setup does not require any manual input.

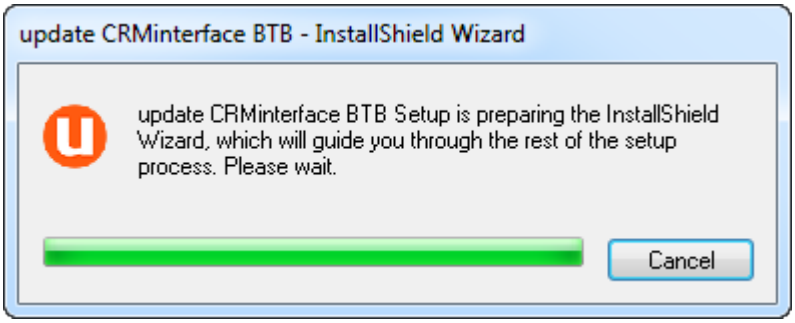
Note: If you are installing CRMinterface on a machine with under Windows 7 64Bit and IIS 7.0 a warning message is displayed and you have to ensure that you enable anonymous access in IIS.

During the course of the setup procedure an application pool (Aurea_CRM.interface) and a virtual directory in IIS is created.

If a matching Aurea CRM win installation is found, the database connection settings are copied from this installation to the CRM.interface installation. If such an installation is not found you have to define manually the connection string to your Aurea CRM database in the `mmdb.ini`.

Note: By default the `mmdb.ini` is located in `C:\Program Files\Aurea CRM\CRMinterface BTB\web\system\sys`.

Setup wizard welcome screen



Once started, the wizard guides you during the installation process.

Post installation steps

Learn about the post installation steps.

Creating a data base connection

If Aurea.CRM win is installed on the same computer where you are installing CRM.interface, setup has already copied the `..\sys` directory to the Aurea CRM web directory.

Otherwise you must copy the `..\sys` directory manually for CRM.interface to work or manually define the connection string in the `mmdb.ini`.

Note: By default the `mmdb.ini` is located in `C:\Program Files\Aurea CRM\ CR-Minterface BTB\web\system\sys` file.

Creating the Technical (Shadow) User in Aurea.CRM

You need to create the Technical ("Shadow") User in Aurea CRM. To run CRM.interface you must map the technical user to a Rep of the type "Employee". Please refer to the Aurea CRM win manual on how to create a user. By default this user is the "WWW" user, but you can also change the user name of the technical user.

Also see Configuration of the technical user, creation of users.xml and Shadow User on page 143 for further information.

Enabling Friendly Error Messages During Login

Friendly error messages displayed by CRM.Interface during login allows users with malicious intent to discern existing or valid users. Friendly error messages are disabled by default. To enable this feature set the `FriendlyLoginError` option to `1` in the `settings.xml` file, as shown in the sample below:

```
"FriendlyLoginError = 1"
```

Configuration of CRM.interface

Learn about the CRM.interface configuration.

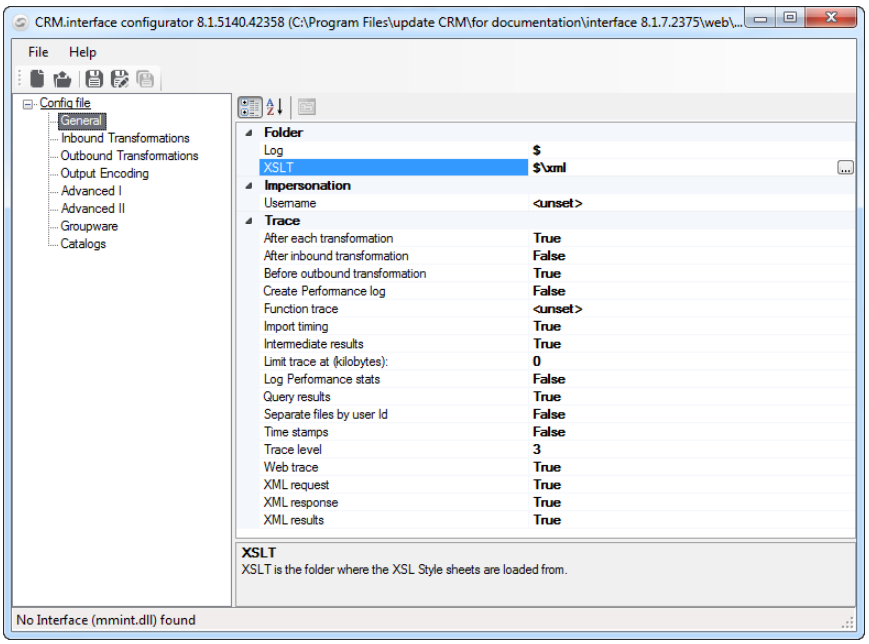
Configuration of the technical user, creation of users.xml

As of version 8.1.7.433 and higher the credentials (username and password) of the technical user are stored in an encrypted XML file - `users.xml`. Use `update.Users.exe` – which is installed in the \web\Bin subfolder of the installation directory to create or modify the `users.xml` file.

Also see Creating the Technical (Shadow) User in Aurea.CRM and Shadow User on page 143 for further information.

Configuration of CRM.interface

CRM.interface can be configured using a configuration tool (`update.Interface.Configurator.exe`), which is installed in the `\tools` folder of CRM.interface.



For all settings a help text is displayed in the "help pane" on the bottom.

Note: The configuration tool for CRM.interface supports creating and maintaining parent-child configurations allowing to separate general configurations from (instance) specific settings in scenarios, where multiple instances of CRM.interface are used.

Note: CRM.interface itself does not yet support such hierarchical (parent-child) configurations – this extension is planned for the future.

Logging

Learn about logging in CRM.interface.

CRM.interface is a plugin in the Aurea.CRM server framework and is part of the of the homogenized server platform of Aurea.CRM. Basically the behavior of CRM.interface is defined via two settings files:

- the `settings.xml` of the framework (`\web`), which holds the Aurea CRM server specific settings and
- the CRM.interface-specific `settings.xml` (`\web\system`).

Additionally CRM.interface is influenced by the `mm.ini` file in `\web\system\sys`.

Consequently CRM.interface generates two log files:

1. All logging of Aurea CRM server (e.g. regarding loading and unloading of the CRM.interface plug-in) is written to `u8_iis_interface.log`.
2. All logging about the actual processing of CRM.interface is written to `u8_interface.log`.

Note: See the Aurea WIKI articles *HOWTO Configure Logging for the Core* and *HOWTO configure diagnostic logging for CRM.interface* for more details.

3

Integration Hub

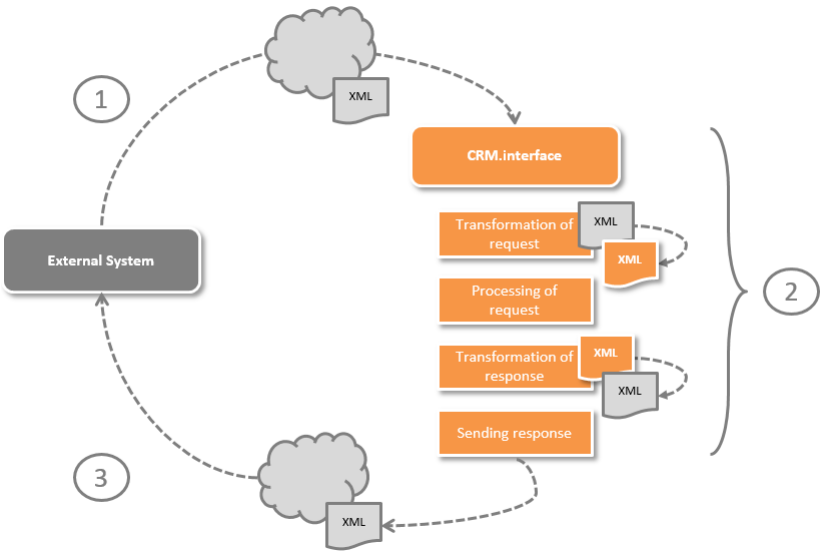
Learn about CRM.interface Integration hub client and server.

CRM.interface Integration Server

When CRM.interface acts as an integration server the process is initiated by the particular client.

The external system triggers the process via sending an XML message over http(s). This message is transformed and processed by CRM.interface, the response is transformed into the XML dialect of the 3rd party system and sent to the External System.

CRM.interface (server) – message flow



CRM.interface Test Client

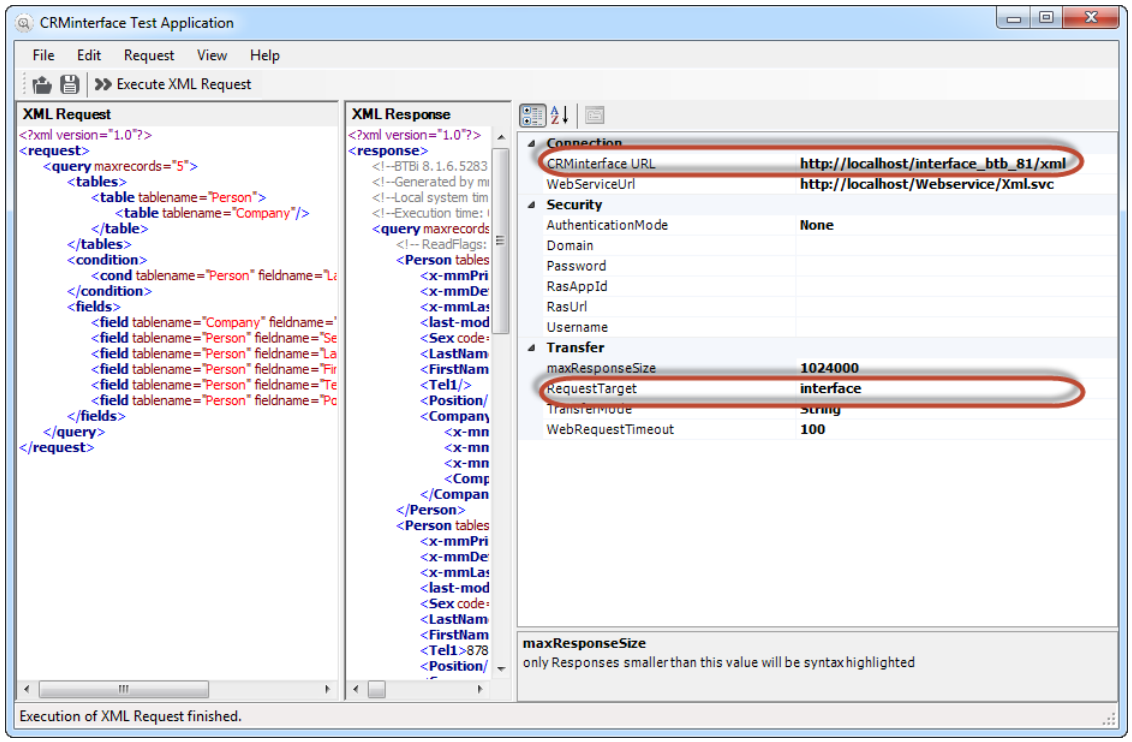
The CRM.interface (and CRM.webservices test) client allows for testing CRM.interface. The test client is installed into the \tools subfolder of the installation directory.

In the property pane you can basically define to which service endpoint the request should be sent and which authentication mode you want to use.

Note: the CRM.interface test client is provided "as-is". Some of the properties in the Property Pane are for internal use only, e.g. the "RAS" settings in the security hive.

In order to test CRM.interface with the test client set the value of the CRM.interface Url to the Url of your interface endpoint and set the Request Target to "interface".

CRM.interface Test Client

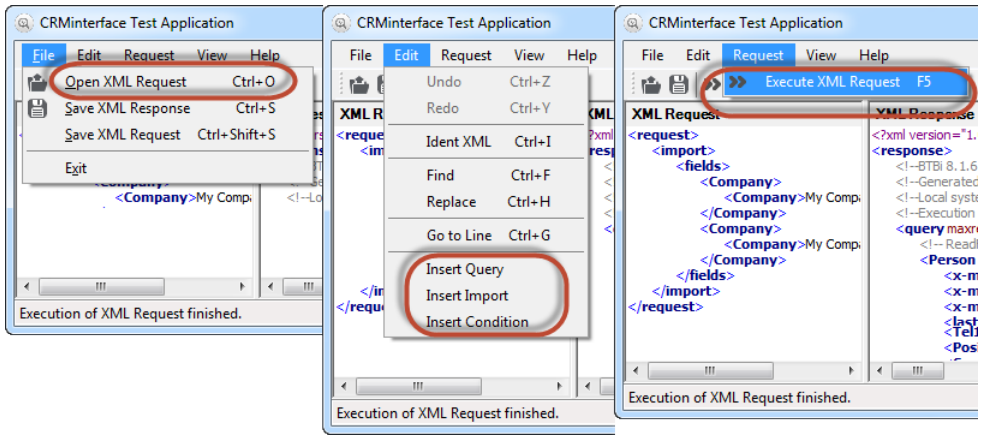


Note: In order to execute a XML request either press "Execute XML Request" or F5.

Note: Tip: the test client allows for saving of XML requests (and responses) and hence the execution of saved XML requests, which might be helpful, when testing more complex requests.

Note: Additionally it's possible to insert "templates" for `<query />`, `<import />` and `<condition />` commands.

Loading, saving and executing XML requests with the test client



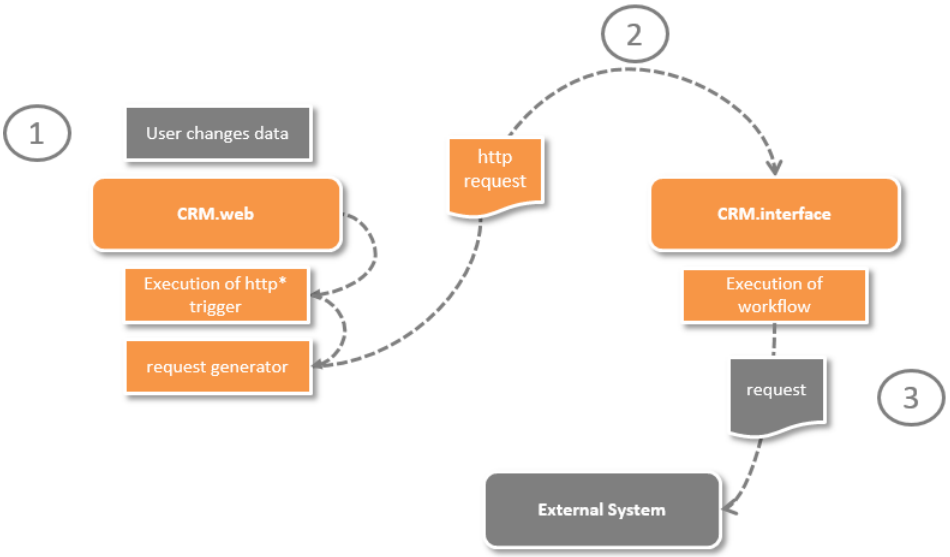
CRM.interface integration client

CRM.interface can also actively trigger the interaction with third-party systems, e.g., sending requests to an ERP system.

Note: Excursion for readers familiar with update.seven: the integration client can cover for the same scenarios in which update.seven HTTP client was used in the past, but the new CRM.interface client is dramatically enhanced regarding functionality and flexibility.

Invocation is done by sending specific http "get" messages to CRM.interface - based on such a message a certain workflow is executed. The implementation of this workflow is based on the **Microsoft Workflow Foundation** framework. Out-of-the-box CRM.interface ships with one such predefined workflow, which is called "Synchronize".

CRM.interface (client) – message flow



Although CRM.interface is agnostic as to which application invokes the integration workflow (as long as the required parameters are provided) there is a built-in mechanism in Aurea CRM allowing for invoking the integration workflow when data is changed in Aurea CRM web (for example).

As shown in the figure above, when changing data in Aurea CRM web a program call trigger is fired; this http* trigger composes a http request which is sent to the defined CRM.interface endpoint. For further information about integration with client see https://support.aurea.com

The Synchronize Workflow

Learn about the CRM web interface synchronization steps.

As stated above, CRM.interface currently ships with one out-of-the-box workflow.

Activities

This workflow is called "synchronize" and can -

- create and execute XML queries against Aurea CRM
- transform the XML query response to 3rd party system format
- exchange data with 3rd party system, either via Web Requests or SOAP messages
- transform the response from the 3rd party system into CRM.interface XML messages
- create and modify data in Aurea CRM

The workflow is composed of the following activities:

- WorkflowSettings: This activity parses the incoming request for the settings parameters and constructs a list of parameters, which are then used while

executing the workflow. See Workflow settings and parameters for further details on how this parameter list is constructed.

- ReadFile: This activity loads a file specified by its parameter FileName and returns the content of this file. The “synchronize” workflow uses this activity to read query template and transformation schema files.
- XslTransformer: This activity transforms XML using transformation schemata. The “synchronize” workflow uses this activity for transforming CRM.interface XML messages into 3rd party messages and vice versa.
- RestAPI: This activity sends data to an URL specified in the URL parameter. The “synchronize” workflow uses this activity for communication with the particular 3rd party system.
- ProcessXml: This activity accesses Aurea CRM via CRM.interface. Basically this activity is used for both retrieving data form Aurea CRM based on the provided parameters in the invocation message as well as importing/updating data in Aurea CRM based on the responses from the 3rd party system.

Workflow Execution steps

1. An invocation message with the name of the workflow and a list of parameters is sent to CRM.interface.
2. The integration hub checks if the referenced workflow exists and after successful compilation an HttpRequest object is passed to the workflow.
3. At this point the workflow starts its sequence of activities:
   a. If the invocation message contains the settings parameter the workflow opens the referenced XML file and construct the list of specified parameters.
   b. The workflow loads the query template file from the path specified in the query parameter.
   c. Placeholders in query template which are marked with {parameter name} (e.g. {company_No}) are replaced by values provided in the URL (e.g. company_No=15)
   d. The XML query is executed against Aurea CRM (data is read).
   e. If outTransformation is provided the workflow transforms the response from CRM.interface using the transformation schema specified via the value of this parameter. Otherwise the execution is continued with step 3.k on page 21.
   f. If the useSoap parameter is set to true then the workflow continues with the step 7 otherwise it jumps to step 8.
   g. If useSoapTransformation is set to true then the workflow constructs the SOAP message by applying the XSLT transformation defined in soapOutTranformation. Otherwise it uses the XML template provided in the parameter soapTemplate to generate the SOAP message.
   h. The output of above transformation(s) is sent to the endpoint either defined by the value of targetUrl or soapUrl. If useSoap is false or no soapAction is provided then the endpoint is the URL provided in targetUrl.
   i. If the inTransformation parameter is provided, the workflow performs step 10, otherwise the execution jumps to step 3.k on page 21.
   j. The workflow transforms the response from the 3rd party system using the transformation schema specified via the parameter inTransformation and sends the output of the transformation to Aurea CRM
   k. The workflow returns an execution output xml and ends its sequence.

Workflow settings and parameters

Parameters can either be provided directly in the URL or as an XML file (in this case the URL has to contain the parameter settings with the name and path of that file).

Note: You can have multiple settings XML files for a single workflow.

Note: For example, you can have one settings XML file for exchanging Opportunities (e.g. `y1_settings.xml`) with the 3rd party system and another for exchanging contact persons (e.g. `kp_settings.xml`).

If a parameter both is specified in the URL string and in the settings XML file, the value defined in the URL string replaces the particular value of the settings file. If a parameter is provided in the URL and is not found in the settings XML file, then its value is added to the list of parameters. If a parameter is solely defined in the settings XML file, its value is added to the list of parameters.

Note: If you want to omit a parameter, which is defined in the settings XML file, you have to omit the parameter settings in the URL and provide all required parameters directly in the URL.

The following parameters are understood by the "synchronize" workflow

"synchronize" workflow parameters

Parameter	Meaning
loggingEnabled	Boolean indicator if the logging is on or off. This parameter is optional.
query	Path to the query template XML file
outTransformation	Path to the XSLT file for transforming CRM.interface XML into 3rd party system XML format. **Note:** if this parameter is not provided no message is sent at all.
inTransformation	Path to the XSLT file for transforming 3rd party system XML into CRM.interface format
useSoap	if true SOAP messaging is used. Otherwise a (normal) Web Request is sent to the 3rd party system.
targetUrl	URL of 3rd party system. A "normal" Web Request is sent to this URL if useSoap set to false.
soapUrl	URL of the 3rd party web service. A SOAP message is sent to this URL if useSoap set to true.
soapAction	method of the 3rd party web service, which should be executed. If useSoap set to true, but no soapAction is provided then a regular Web Request is sent to targetUrl as a fallback solution.
useSoapTransformation	if true the SOAP message is created by applying the XSLT stylesheet as defined in the parameter soapOutTranformation. Otherwise the SOAP message is generated based on the XML template as defined in the parameter soapTemplate.
soapTemplate	Path to XML template file used to create the SOAP message. If useSoapTransformation set to true this parameter is ignored.
soapOutTranformation	Path to XSLT used to create the SOAP message. Only used if useSoapTransformation is set to true.

Note: Support for SOAP messaging is available of version 8.1.7.433 and higher.

Note: If relative file paths are used these paths have to be relative to the root directory of the CRM.interface web application (and not relative to the location of the workflow XAML file).

The settings file of the "synchronize" workflow has the following structure.

Settings file of workflow

```
<?xml version="1.0"?>
<settings>
    <loggingEnabled></loggingEnabled> <!--  indicator if the logging is on or
off-->
    <query></ query> <!--  path to the query template xml file  -->
    <outTransformation></ outTransformation> <!--  path to XSLT file responsible
for transformation from Aurea CRM format into 3rd party system format        -->

   <inTransformation></ inTransformation> <!--  path to XSLT file responsible for
 transformation from 3rd party system into format Aurea CRM format -->
    <targetUrl></ targetUrl> <!--  URL of 3rd party system-->
    <useSoap></ useSoap> <!--  indicator if SOAP messaging will be used   -->
   <soapAction></ soapAction> <!--  method of the 3rd party web service to invoke
 -->
    <soapTemplate></ soapTemplate> <!--  XML template file used to create SOAP
message-->
    <soapUrl></ soapUrl> <!--  URL of web service endpoint -->
    <useSoapTransformation></ useSoapTransformation> <!-- indicator, if XSLT file
 (soapOutTranformation) or XML template (soapTemplate) should be used -->
   <soapOutTranformation></ soapOutTranformation> <!--  path to the XSLT file for
 transforming from Aurea CRM format into SOAP message -->
</settings>
```

How to integrate with a 3rd party system which requires a login

It is possible to integrate with 3rd party systems, which require an explicit login.

Note: the "Cookie-Handling" functionality described below requires service pack 2 of CRM.interface.

In order to set up the integration with such a system you only need to add a second instance of RestApi activity to your integration workflow. The first instance of RestApi activity is responsible for making the "Login" into the 3rd party system and retrieving the cookies. The second activity instance is responsible for exchanging the data between the integration hub and the 3rd party endpoint.

Note: For the communication with SOAP endpoints the workflow now supports a new property Cookies. If the response header contains a Set-Cookie entry, its value is assigned to the Cookies property of the activity. In order to share the cookie between multiple activity instances, a workflow variable is used and assigned to the property of each instance. This way the first call to the 3rd party service fills the

variable with the value from the response and each consecutive RestApi instance sends it along with the request.

Note: For non-SOAP communications the property HttpClient is used to store the cookie. When this property is assigned to an instance of the HttpClient object with CookieContainer, it is used to make the call to the 3rd party system. If this instance of HttpClient is shared in similar fashion as described above (workflow variable passed to each RestApi instance), cookies retrieved from the first call to the 3rd party service is resent with every subsequent request.

Logging

Learn about logging of workflow activities.

Logging of workflow activities can be enabled or disabled via the parameter loggingEnabled in the settings file. If `<loggingEnabled>true</loggingEnabled>` is set all activities of the workflow are logged.

Invoking integration client via program call trigger

Learn how to invoice integration client via program call trigger.

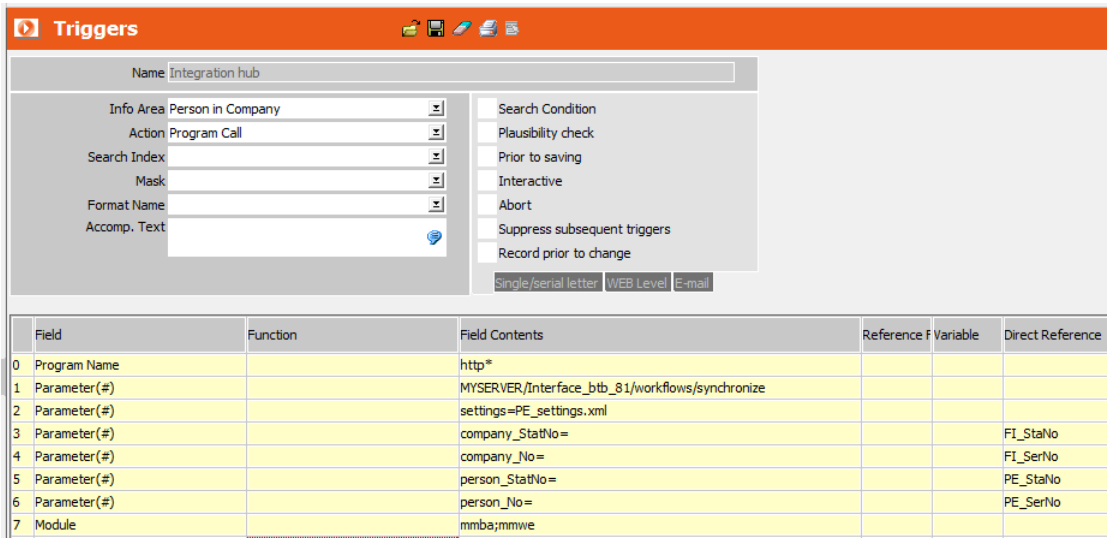
If you want to invoke CRM.interface client via Aurea CRM trigger you have to

- create a Program Call trigger
- set `Program Name` to `http*`
- specify the endpoint in the first parameter (endpoint always means the full path to the workflow which should be executed)

In the example below

- the endpoint of CRM.interface is `MyServer/Interface_btb_81/workflows/synchronize`
- the settings are defined in the `PE_settings.xml` file
- four fields (`FI_StatNo`, `FI_SerNo`, `PE_StatNo`, `PE_SerNo`) of the actual data record are passed as parameters to the workflow

Example of Program Call trigger (http* trigger)



The trigger in our example generates an invocation message which looks something like that:

```
http://MYSERVER/Interface_btb_81/workflows/synchronize?settings=PE_set-
tings.xml&company_ StatNo=&company_No=15&person_StatNo=100&person_No=834
```

The `PE_settings.xml` in our example references a query template file which looks something like that:

Example of a query template file

```
<?xml version='1.0'?>
<request>
<query maxrecords='100'>
  <tables>
 <table tablename='Company'>
   <table tablename='Person' flags='256'/>
 </table>
  </tables>
  <fields tablename='Company' fields='CoGrp,CoNo,Company,Country,FreeN1'/>
  <fields tablename='Person' fields='Sex,LastName,FirstName,FreeN1'/>
  <condition>
 <cond tablename='Company' fieldname='CoGrp' op='=' value='{company_StatNo}'/>
 <cond tablename='Company' fieldname='CoNo' op='=' value='{company_No}'/>
  </condition>
  <condition>
 <cond tablename='Person' fieldname='PeGrp' op='=' value='{person_StatNo}'/>
 <cond tablename='Person' fieldname='PeNo' op='=' value='{person_No}'/>
  </condition>
</query>
</request>
```

The parameters provided in the invocation message are merged into the query template file. During this merge the placeholders -which are marked with `{parameter name}` (e.g. `{company_No}`)- are replaced by the matching values.

In our example `value='{company_No}'` is replaced by `value='15'` (since the parameter provided in the invocation message is `company_No=15`).

Excursion-invoking CRM.interface via URL parameter

CRM.interface is agnostic as to which application invokes the integration workflow (as long as the required parameters are provided).

In order to invoke the "synchronize" workflow the request needs to contain the following information

- Name of the workflow
- Name of the settings file (or alternatively the settings as URL parameters)
- Fields of the actual data record, which are used to read data from Aurea CRM and prepare the outgoing message

For example, if you want to invoke CRM.interface client via a Button-click in Aurea CRM web, you would just need to call CRM.interface via URL parameter.

4

XML Syntax Reference

List of XML syntax references

Structure of Requests

```
<request attributes>
      <commands attributes />
      </request>
```

Commands

List of action XML syntax reference.

<request>

The `<request>` element contains the definition of an interface XML request. It is the root element of the document, and the whole document is defined to be namespace-neutral. (Example, see How to remove namespaces on page 163).

<request/>	
Appearance	<request> </request>
Attributes	ixslt log logmode noerrorlog oxslt any Session Attribute any Custom Attribute
Contents	(Any command element)+
May occur in	(No parent elements - defines the top-level element)
Remarks	Attributes on the <response> element that are not processed by CRM.interface (i.e. that are not in the attributes list above) are plainly copied to the <response> element of the XML response. This allows for easy pass-through of key-value pairs that can for example be used to connect the request to its corresponding response via a unique id.

<break>

The `<break>` command stops processing the request if the break condition is fulfilled. Commands not already executed are left unprocessed.

<break/>	
Appearance	<break> </break>
Attributes	if transaction any Custom Attribute
Contents	(No child elements)
May occur in	< request >
Remarks	none

```
<?xml version="1.0"?>
    <request xmlns:mp="http://www.update.com/xml/core/mp">
      <import>
        <fields>
          <Company>
            <Company matchup="true">break-dance</Company>
            <UndefinedField>undefined field</UndefinedField>
          </Company>
        </fields>
      </import>
      <!-- break on error -->
      <break if="/root/response/*[last()]//return[@type='error']"/>
      <!-- otherwise, do further processing -->
      <nop/>
    </request>
```

In the example above the execution of the request is stopped if an error occurs (`<UndefinedField>` doesn't exist in Aurea CRM; therefore the `<nop>` command following the import command is not executed, see response below).

If you would resolve the error in the request, the break condition is no longer fulfilled and the `<nop>` command would be executed.

```
<response xmlns:mp="http://www.update.com/xml/core/mp" xmlns="">
       <import>
       <return table="FI" id="0" type="error" func="C_Portal::ProcessImportNode">

           <ecode>-10027</ecode>
           <etext>Dictionary: Field not found</etext>
           <table table="FI" tablename="Company"/>
           <field>UndefinedField</field>
         </return>
       </import>
     </response>
```

<debug>

The `<debug>` command is for internal purposes only. Do not use except when explicitly instructed to do so.

<delete>

The `<delete>` command allows for deletion of multiple records from a single table (like in the Aurea CRM win service module).

<delete/>	
Appearance	<delete> </delete>
Attributes	any Command Attribute if any Custom Attribute
Contents	tables links? fields? condition? sortlist? custom_sortlist?
May occur in	< request >
Remarks	The <delete> command is executed as a <query> and the resulting records are deleted. Be sure that the set of records to be deleted are properly limited by conditions and/or links, otherwise you might delete too many or even all records in a table (including all its child records).

```
<?xml version="1.0"?>
      <request>
        <delete>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="="
value="delete-test*"/>
          </condition>
        </delete>
      </request>
```

In the example above all company records that match the condition (Company starts with "delete-test") is deleted. The response returns the record id of each deleted record.

```
  <response>
        <delete>
          <return table="FI" tablename="Company" id="4294979919" type="delete"/>
          <return table="FI" tablename="Company" id="4294979937" type="delete"/>
          <!-- … -->
        </delete>
      </response>
```

In the example below person records that match the condition (LastName = "Maier") within the linked company record (Company = "delete-test") is deleted.

The response is similar to the example before.

```
<?xml version="1.0"?>
    <request>
      <delete>
        <tables>
          <table tablename="Person"/>
        </tables>
        <condition>
          <cond tablename="Person" fieldname="LastName" op="=" value="Maier"/>
        </condition>
        <links>
          <Company>
            <Company>delete-test</Company>
          </Company>
        </links>
      </delete>
    </request>
```

<dictionary>

Note: The dictionary is no longer used in Aurea CRM; instead the XML names are read from the Aurea CRM data model (field "XML field name"). Therefore the `<dictionary />` command serves for informational purposes, only.

The `<dictionary>` command generates a new `dictionary.xml` in the `\xml` subfolder of the installation directory - an existing file is renamed to `dictionary_YYYYMMDDhhmmss.xml`.

<dictionary/>	
Appearance	<dictionary> </dictionary>
Attributes	complete if transaction any Custom Attribute
Contents	(No child elements)
May occur in	<request>
Remarks	**Note:** Deprecated. The dictionary is no longer used in Aurea CRM.

Attribute	Meaning
text_class = CustomTexts	custom text is defined in the dictionary
text_class = CoreTexts	text is defined in data model
text_class = MissingTexts	neither in the data model, nor in the dictionary a custom text is defined, text is generated automatically based on the field number

<getcat>

The `<getcat>` command exports all values for a catalog. If the catalog is dependent, the parent values are also generated.

<getcat/>	
Appearance	<getcat> </getcat>
Attributes	extkey id include_locked_values if transaction any Custom Attribute
Contents	(No child elements)

<getcat/>	
May occur in	<request>
Remarks	The <getcat> command returns the language-dependent catalog texts. Furthermore, sort number and tenant are also included in the response. If extkey='true', external keys are also returned. The catalog can be given by number/name, catalog number or the core catalog number.

```
<?xml version="1.0"?>
<request>
  <getcat tablename="Company" fieldname="Country"/>
</request>
<response>
  <getcat tablename="Company" fieldname="Country">
    <catalog index="2" ucatindex="-1">
      <cat index="1" locked="false">Österreich</cat>
      <cat index="2" locked="false">Deutschland</cat>
      <cat index="3" locked="false">Schweiz</cat>
    </catalog>
  </getcat>
</response>
```

In this example the catalog texts of a dependent catalog is returned. Additionally its respective parent catalog text is returned.

```
<?xml version="1.0"?>
<request>
  <getcat tablename="ProblemResolution" fieldname="Product"/>
</request>
<response>
  <getcat tablename="ProblemResolution" fieldname="Product">
    <catalog index="28" ucatindex="29">
      <ucat index="0" locked="false">
        <cat1>Product Group 1</cat1>
        <cat index="0" locked="false">Product 1-1</cat>
      </ucat>
      <ucat index="1" locked="false">
        <cat1>Product Group 2</cat1>
        <cat index="0" locked="false">Product 2-1</cat>
        <cat index="1" locked="false">Product 2-2</cat>
        <cat index="2" locked="false">Product 2-3</cat>
      </ucat>
    </catalog>
  </getcat>
</response>
```

The example below illustrates request and response using extkey and id attribute.

```
<?xml version='1.0'?>
<request>
  <getcat id='2' extkey='true' include_locked_values='true'/>
  <getcat name='Firma-Frei 3'/>
  <getcat table='IT' fieldname='Interest'/>
</request>
<?xml version='1.0'?>
<response>
  <getcat id="2" extkey="true" include_locked_values="true">
    <catalog id="2" cid="20000" name="Land">
      <cat code="2" extkey="GER" value="Deutschland"/>
      <cat code="1" extkey="AUT" value="Österreich"/>
      <cat code="4" locked="true" mnr="69" sort="666" value="test"/>
      <cat code="3" extkey="THA" value="Thailand"/>
    </catalog>
  </getcat>
```

```
      <getcat name="Firma-Frei 3">
        <catalog id="61" cid="20021" name="Firma-Frei 3">
          <cat code="1" sort="1001" value="FI free3 1001"/>
        </catalog>
      </getcat>
      <getcat table="IT" fieldname="Interest">
        <catalog id="46" pid="45" cid="20035" name="Interesse">
          <cat code="2" value="leer"/>
          <cat code="1" value="test">
            <cat code="1" value="test1"/>
            <cat code="2" value="test2"/>
            <cat code="3" value="test3"/>
          </cat>
        </catalog>
      </getcat>
    </response>
```

<getdoc>

The `<getdoc>` command reads documents from the D1 or D2 document tables and embeds them base64-encoded in the response. XML documents can optionally be embedded as inline XML.

<getdoc />	
Appearance	<getdoc> </getdoc>
Attributes	decipher decrypt forceBase64 verify any Command Attribute if any Custom Attribute
Contents	tables links? fields? condition? sortlist? custom_sortlist?
May occur in	<request>
Remarks	The <getdoc> command is executed as a <query> and the resulting records are then processed as follows. If it is a document record (D1 or D2), its document is used. Otherwise, all fields read are interpreted as a document reference and the corresponding document is used.

In this example all records from D1 document table that match the condition are read and the document itself is returned as base64-encoded string in the response. The contents of the fields declared in the `<fields>` section is also returned.

To read documents from the D2 table use `tablename="CustomerDocuments"` instead of "Documents".

```
<?xml version="1.0"?>
      <request>
        <getdoc>
          <tables>
            <table tablename="Documents"/>
          </tables>
          <fields>
            <!-- Declare any additional fields to be read from D1 record -->
```

```
            <field tablename="Documents" fieldname="Keyword"/>
          </fields>
          <condition>
            <cond tablename="Documents" fieldname="Name" op="=" value="test*"/>
          </condition>
        </getdoc>
      </request>
      <response>
        <document table="D1" id="4294967495">
          <docid>A1-199</docid>
          <Name>test.txt</Name>
          <ContentType>text/plain</ContentType>
          <Size>4</Size>
          <Keyword>example</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </document>
        <document table="D1" id="4294967497">
          <docid>A1-201</docid>
          <Name>test2.txt</Name>
          <ContentType>text/plain</ContentType>
          <Size>4</Size>
          <Keyword>example</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </document>
      </response>
```

The base64-encoded document is within the rawdata element. The elements `<docid>`, `<Name>`, `<Size>` and `<ContentType>` are returned by default.

In the next example below all contact records that match the condition (current date) are read. The fields declared in the `<fields>` section are interpreted as a document reference (e.g.: "A1-200", "K100-5") and the corresponding document is returned as base64-encoded string in the response.

```
<?xml version="1.0"?>
      <request>
        <getdoc>
          <tables>
            <table tablename="Contact"/>
          </tables>
          <fields>
            <field tablename="Contact" fieldname="Document1"/>
          </fields>
          <condition>
            <cond tablename="Contact" fieldname="Date" op="=" value="$curDay"/>
          </condition>
        </getdoc>
      </request>
      If the field is empty the following error will be returned:
      <response>
        <getdoc>
          <return table="MA" id="4295006414" type="error"
func="C_Portal::XmlProcessGetDocument">
            <ecode>-10054</ecode>
            <etext>Field is empty</etext>
            <table>MA</table>
            <field>Document1</field>
          </return>
        </getdoc>
      </response>
     If the field contains a wrong or not existing document reference the following
 error will be returned:
      <response>
        <getdoc>
          <return type="error" func="C_Portal::XmlProcessGetDocumentEx">
            <ecode>-10055</ecode>
            <etext>Document not found</etext>
            <code>0</code>
```

```
            <docid>wrongref</docid>
          </return>
        </getdoc>
      </response>
```

<history>

The <history> command exports the record history (that is stored in the H0 table).

<history />	
Appearance	<history> </history>
Attributes	any Command Attribute any Custom Attribute
Contents	tables links? fields? condition? sortlist? custom_sortlist? history_condition?
May occur in	<request>
Remarks	The <history> command is executed as a <query> with the resulting records being output with their history added. The record history can also be output by reading the H0 table as a dependent table.

```
<request>
        <query>
          <tables>
            <table tablename="Company">
              <table tablename="History"/>
            </table>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="update
software AG"/>
          </condition>
          <fields tablename="Company" fields="Company"/>
          <fields tablename="History" fields="1"/>
        </query>
        <history>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="update
software AG"/>
          </condition>
        </history>
      </request>
```

In the query response detailed information is returned, whereas in the history response the history information is generated in a "query result"-structure.

```
<response>
        <query>
          <Company table="FI" id="296352743952" recId="x0000004500000210">
            <Company>update software AG</Company>
            <history>
              <revisions>
```

```
                <revision id="1">
                  <header version="7060" dm="67">
                    <date_time>2009-09-17T12:46:11.241</date_time>
                    <module id="11">XML</module>
                    <StatNo>69</StatNo>
                    <Flags name="ModuleFlags"/>
                    <UserId>6900002</UserId>
                    <mode value="2">update</mode>
                    <function value="0"/>
                    <Flags name="NulFlags">Mapping</Flags>
                    <Flags name="Flags">fields</Flags>
                    <internal len="122"/>
                  </header>
                  <fields>
                    <field id="0" fid="60" type="string" hdb_type="2"
hdb_len="8">prev</field>
                    <field id="1" fid="4011" type="date" hdb_type="15"
hdb_len="4">2009-09-17</field>
                    <field id="2" fid="4016" type="time_msec" hdb_type="19"
hdb_len="4">12:46:11.209</field>
                    <field id="3" fid="4203" type="unsigned_short" hdb_type="9"
hdb_len="2">69</field>
                    <field id="4" fid="4204" type="rep" hdb_type="21"
hdb_len="4">WWW</field>
            <field id="5" fid="4205" type="static_catalog" hdb_type="6"
hdb_len="1">XML</field>
                    <field id="6" fid="4206" type="unsigned_int" hdb_type="11"
hdb_len="4">67108864</field>
                  </fields>
                </revision>
                <revision id="2">
                  <header version="7060" dm="67">
                    <date_time>2009-09-17T13:08:32.213</date_time>
                    <module id="11">XML</module>
                    <StatNo>69</StatNo>
                    <Flags name="ModuleFlags"/>
                    <UserId>6900002</UserId>
                    <mode value="2">update</mode>
                    <function value="0"/>
                    <Flags name="NulFlags">Mapping</Flags>
                    <Flags name="Flags">fields</Flags>
                    <internal len="92"/>
                  </header>
                  <fields>
                    <field id="0" fid="61" type="string" hdb_type="2"
hdb_len="8">prev</field>
                    <field id="1" fid="4016" type="time_msec" hdb_type="19"
hdb_len="4">13:08:32.198</field>
                  </fields>
                </revision>
              </revisions>
            </history>
          </Company>
        </query>
        <history>
          <Company table="FI" id="296352743952" recId="x0000004500000210">
            <FreeC1>prev</FreeC1>
            <Upd>2009-09-17</Upd>
            <UpdTime>12:46:11.209</UpdTime>
            <x-UpdStatNo>69</x-UpdStatNo>
            <x-UpdUserId>WWW</x-UpdUserId>
            <x-UpdModule>XML</x-UpdModule>
            <x-UpdFlag>67108864</x-UpdFlag>
          </Company>
          <Company table="FI" id="296352743952" recId="x0000004500000210">
            <FreeC2>prev</FreeC2>
            <UpdTime>13:08:32.198</UpdTime>
          </Company>
        </history>
      </response>
```

<import>

The <import> command defines a write operation (insert/update/match) over one or more tables.

<import/>	
Appearance	<import> </import>
Attributes	allow_deleted catnew force_update mode no_insert write_cursor_flags any Command Attribute if any Custom Attribute
Contents	links? fields
May occur in	<request>
Remarks	See Matchup on page 147 and especially the note in the topic internal/external matchup.

In this example two company records are imported into the database. The <import> command considers the matchup logic (similar to the import module, see Matchup on page 147). In case of the first company record in this example:

if matchup finds a company record with the imported external key, the record is updated.

if no company with this key exists, the company is inserted as a new record.

```
<?xml version="1.0"?>
      <request>
        <import>
          <fields>
            <Company>
              <ExtSystem>External</ExtSystem>
              <ExtKey>1508</ExtKey>
              <Company>My Company 1</Company>
              <Synonym>external</Synonym>
              <!– any other fields from company -->
            </Company>
            <Company>
              <Company>My Company 2</Company>
              <Synonym>internal</Synonym>
              <!– any other fields from company -->
            </Company>
            <!– any other company records-->
          </fields>
        </import>
      </request>
```

The response returns the id of each imported record and the type of the write operation (insert/update/match)

```
<response>
        <import>
          <return table="FI" tablename="Company" id="4294979917" type="update"/>
```

```
          <return table="FI" tablename="Company" id="4294979924" type="insert"/>
        </import>
      </response>
```

In the next example records from more tables is imported. One company with tow Persons and one contact for the second person.

```
<request xml:lang="de">
        <import>
          <fields>
            <Company>
              <ExtSystem>SAP</ExtSystem>
              <ExtKey>1508</ExtKey>
              <Company>My Company</Company>
              <Synonym>external</Synonym>
              <!-- any other fields from company -->
              <Person>
                <FirstName>John</FirstName>
                <LastName>Doe</LastName>
                <!-- any other fields from person-->
              </Person>
              <Person>
                <FirstName>Jane</FirstName>
                <LastName>Doe</LastName>
                <!-- any other fields from person-->
                <Contact>
                  <Contact>Telefon</Contact>
                  <Subject>Imported via CRM.interface</Subject>
                  <!-- any other fields from contact-->
                </Contact>
                <!-- any other contact records-->
              </Person>
              <!-- any other person records -->
            </Company>
            <!-- any other company records -->
          </fields>
        </import>
      </request>
      <response>
        <import>
          <return table="FI" tablename="Company" id="4294979917" type="update"/>
          <return table="KP" tablename="Person" id="4294977462" type="insert">
            <links>
              <link table="FI" tablename="Company" id="4294979923" linkId="-1"/>
            </links>
          </return>
          <return table="KP" tablename="Person" id="4294977463" type="insert">
            <links>
              <link table="FI" tablename="Company" id="4294979923" linkId="-1"/>
            </links>
          </return>
          <return table="MA" tablename="Contact" id="4295044743" type="insert">
            <links>
              <link table="KP" tablename="Person" id="4294977463" linkId="-1"/>
            </links>
          </return>
        </import>
      </response>
```

In this example the company record declared in the `<links>` section is read and – if found - set as the link record of the imported contact. The structure of the response is the same shown as in the examples above.

```
<request xml:lang="de">
        <import>
          <fields>
            <Contact>
              <links>
                <Company>
                  <ExtSystem>SAP</ExtSystem>
                  <ExtKey>1508</ExtKey>
```

```
            </Company>
          </links>
          <Contact>Brief</Contact>
          <Subject>Imported via CRM.interface</Subject>
        </Contact>
      </fields>
    </import>
  </request>
```

If the link record does not exist, the following error is returned:

```
<response>
    <import>
      <return type="error" func="C_Portal::ReadLinks">
        <ecode>-10041</ecode>
        <etext>Link record not found</etext>
        <link>
          <Company>
            <ExtSystem>External</ExtSystem>
            <ExtKey>1508</ExtKey>
          </Company>
        </link>
        <code>0</code>
      </return>
    </import>
  </response>
```

If more than one link record is found, the following error is returned:

```
<response>
    <import>
      <return type="error" func="C_Portal::ReadLinks">
        <ecode>-10042</ecode>
        <etext>Link record not unique</etext>
        <link>
          <Company>
            <Company>Test</Company>
          </Company>
        </link>
        <code>-2</code>
      </return>
    </import>
  </response>
```

In order to be independent from default xml names within an `<import/>` request you can also reference a field by its unique field id (attribute fid) and a table by its abbreviation (attribute table).

```
<import>
    <fields>
      <call table="KM">
        <links>
          <businesspartner table="FI">
            <name fid="2">update software AG</name>
          </businesspartner>
        </links>
        <key fid="88">ABC10000</key>
        <parentcat fid="4">Software</parentcat>
        <childcat fid="5">Call</childcat>
        <owner fid="50">Alexander Jüttner</owner>
        <owner2 fid="51">Christoph Macheiner</owner2>
      </call>
    </fields>
  </import>
```

The above example is the equivalent to the following one (using the default syntax):

```
<import>
    <fields>
      <ProblemResolution>
        <links>
          <Company>
```

```
            <Company>update software AG</Company>
            </ Company>
          </links>
        <No>ABC10000</No>
        <ProblemGroup>
          Software</ ProblemGroup>
          <Problem>Call</Problem>
          <ProcessedBy>Alexander Jüttner</ProcessedBy>
          <CompletedBy>Christoph Macheiner</CompletedBy>
        </ProblemResolution>
    </fields>
  </import>
```

<insert>

The <insert> command defines a write operation over one or more tables where only insert is permitted.

<insert/>	
Appearance	<insert> </ insert >
Attributes	allow_deleted catnew force_update mode no_insert write_cursor_flags any Command Attribute if any Custom Attribute
Contents	links? fields
May occur in	<request>
Remarks	The <insert> command is executed as an <import> with matchup disabled.

```
<request>
  <insert>
    <fields>
      <Company>
        <Company>My Company 1</Company>
        <Synonym>internal</Synonym>
        <!– any other fields from company -->
      </Company>
      <Company>
        <Company>My Company 2</Company>
        <Synonym>internal</Synonym>
        <!– any other fields from company -->
      </Company>
      <!– any other company records-->
    </fields>
    </ insert>
  </request>
```

The only difference to the <import> command is that there is no matchup logic considered. All records are inserted as new records.

```
<response>
        <insert>
```

```
        <return table="FI" tablename="Company" id="4294979923" type="insert"/>
        <return table="FI" tablename="Company" id="4294979924" type="insert"/>
        </ insert >
    </response>
```

<login>

The `<login>` command changes the current login user and/or language.

<login/>	
Appearance	<login> </login>
Attributes	any Session Attribute if transaction any Custom Attribute
Contents	(No child elements)
May occur in	<request>
Remarks	The <login> command changes the current active login user and/or language for the remainder of the request. In contrast, when credentials and/or language are specified on any other command, they are only used for that single command. See also Shadow User on page 143. CRM.interface uses a technical user – by default the WWW user – to access Aurea CRM if no login context is provided. Use update.Users.exe – which is installed in the installation of interface - to create or modify the `users.xml` file. Because `users.xml` contains the usernames and passwords of a CRM user, it is highly recommend to encrypt the contents of the file via the option "Use Xml Encryption". See also, Authentication on page 143 and Impersonation on page 144.

```
    <request>
      <login user="MyUser" pwd="secret_password"/>
    </request>
    If the login is successful the response returns information about the user,
the corresponding rep user and the login language.
    <response>
      <!-- … -->
      <login user="MyUser">
        <login userId="2">
          <user>MyUser</user>
          <rep repId="103944" grpId="105473">My Rep Name</rep>
          <language>de</language>
        </login>
```

```
      <language>ger (de/German) 0,1,deu</language>
    </login>
  </response>
  If the password is incorrect, the following error is returned (code -17):
  <response>
    <login user="MyUser">
      <return type="error" func="C_Portal::XmlDoLogin">
        <ecode>-10043</ecode>
        <etext>Login error</etext>
        <code>-17</code>
        <user>MyUser</user>
        <language>0</language>
      </return>
    </login>
  </response>
```

Note: If the user does not exist the code = -13 is returned. For other error codes, see the

list of errors
in the Appendix.

In this example the first `<query>` command is executed in the context of user "MyUser" (declared at request element). The first `<login>` command changes the context to user "User1". The next two `<query>` commands are executed in this user's context.

The two `<query>` commands after the second `<login>` command are then executed in the context of user "User2".

```
<request user="MyUser" pwd="my_secret_password">
        <query>
          <!-- query 1 -->
        </query>
        <login user="User1" pwd="password1"/>
        <query>
          <!-- query 2 -->
        </query>
        <query>
          <!-- query 3 -->
        </query>
        <login user="User2" pwd="password2"/>
        <query>
          <!-- query 4 -->
        </query>
        <query>
          <!-- query 5 -->
        </query>
      </request>
```

<matchup>

The `<matchup>` command executes the internal or external matchup for the specified record data.

<matchup/>	
Appearance	<matchup> </matchup>
Attributes	any Command Attribute if any Custom Attribute
Contents	match query ?
May occur in	<request>
Remarks	see also Matchup on page 147.

For each matching record one `<MatchupRecord>` element is returned within the `<match>` element, containing score, text and description.

When the internal matchup is used, score is always 100, text and description are empty - in this case the record ID is returned as well.

When the external matchup is used, the information returned depends on the implementation.

In this example no explicit `<query>` command is specified in the request, so only the Default-Reference of the matching records are returned.

```
<request>
        <matchup>
          <match>
            <fields>
              <Company>
                <Company>update software</Company>
                <Country>Österreich</Country>
                <Street>Operngasse 17-21</Street>
                <ZipCode>1040</ZipCode>
              </Company>
            </fields>
          </match>
        </matchup>
      </request>
      <response>
        <matchup>
          <match>
            <MatchupRecord table="FI" tablename="Company" id="4294967300">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
            <MatchupRecord table="FI" tablename="Company" id="4294975546">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
            <MatchupRecord table="FI" tablename="Company" id="4294975547">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
          </match>
          <query>
```

```
            <Company tableshort="FI" id="4294967300">
              <x-mmDefaultReference>update software AG</x-mmDefaultReference>
            </Company>
            <Company tableshort="FI" id="4294975546">
              <x-mmDefaultReference>update software Germany
GmbH</x-mmDefaultReference>
            </Company>
            <Company tableshort="FI" id="4294975547">
              <x-mmDefaultReference>update software (Switzerland)
GmbH</x-mmDefaultReference>
            </Company>
          </query>
        </matchup>
      </response>
      To read additional fields, specify a <query> after the <match>.
      <request hello="world">
        <matchup>
          <match>
            <fields>
              <Company>
                <Company>update software</Company>
                <Country>Österreich</Country>
                <Street>Operngasse 17-21</Street>
                <ZipCode>1040</ZipCode>
              </Company>
            </fields>
          </match>
          <query>
            <tables>
              <table tablename="Company"/>
            </tables>
            <fields tablename="Company" fields="Company,Synonym,Country"/>
          </query>
        </matchup>
      </request>
      <response hello="world">
        <matchup>
          <match>
            <MatchupRecord table="FI" tablename="Company" id="4294967300">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
            <MatchupRecord table="FI" tablename="Company" id="4294975546">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
            <MatchupRecord table="FI" tablename="Company" id="4294975547">
              <Score>100.00</Score>
              <Text1/>
              <Text2/>
            </MatchupRecord>
          </match>
          <query>
            <Company tableshort="FI" id="4294967300">
              <Company>update software AG</Company>
              <Synonym>update</Synonym>
              <Country>Österreich</Country>
            </Company>
            <Company tableshort="FI" id="4294975546">
              <Company>update software Germany GmbH</Company>
              <Synonym> update</Synonym>
              <Country>Deutschland</Country>
            </Company>
            <Company tableshort="FI" id="4294975547">
              <Company>update software (Switzerland) GmbH</Company>
              <Synonym>update</Synonym>
              <Country>Schweiz</Country>
            </Company>
          </query>
```

```
    </matchup>
  </response>
```

<merge>

The <merge> request allows the merging of Company and Person records via CRM.interface.

The merge mode defines what approach is taken to field-level conflict resolution (meaning which record "wins" when individual field values are conflicting). In source and target modes, the respective record's field values always win. In timestamps mode, the most up-to-date field value wins.

<merge/>	
Appearance	<merge> </merge>
Attributes	table tablename mode verbose no_exec flag_ignore transaction if any Custom Attribute
Contents	destination_record source_record
May occur in	<request>
Remarks	The information needed to perform a merge is: The type of record to be merged: Company or Person The merge mode: source, destination or timestamps A key identifying the source record (Record ID and External Keys are supported) A key identifying the destination* record After the merge operation, the source record is deleted. The "surviving" record is always identified as the destination record, independently of the merge mode.

In this example the source company record is identified via the external key (1234) and is merged into the destination record (extkey=5678). The remaining company contains the field values of the original source company. Existing child records of the source company are moved to the destination company.

```
<request>
        <merge tablename="Company" mode="source">
          <source_record extsystem="External" extkey="1234"/>
          <destination_record extsystem= "External" extkey="5678"/>
        </merge>
      </request>
      <response>
        <merge tablename="Company" mode="source"/>
      </response>
```

To obtain more information about conflicts during the merge process, use the attribute verbose = "true":

```
<request>
        <merge tablename="Company" mode="source" verbose="true">
          <source_record extsystem="External" extkey="1234"/>
          <destination_record extsystem= "External" extkey="5678"/>
        </merge>
      </request>
      <response>
        <merge tablename="Company" mode="source" verbose="true">
          <conflicts>
            <conflict>
              <link linkId="0" srcId="FI" srcName="Company" destId="FI"
destName="Company"/>
               <record source="4294979927" destination="4294979928" duplicate="0"/>

              <source_record>
                <Company table="FI" recId="x0000000100003157">
                  <CoGrp>1</CoGrp>
                  <CoNo>12631</CoNo>
                  <Company>Source</Company>
                  <!-- … -->
                </Company>
              </source_record>
              <destination_record>
                <Company table="FI" recId="x0000000100003158">
                  <CoGrp>1</CoGrp>
                  <CoNo>12632</CoNo>
                  <Company>Destination</Company>
                  <!-- … -->
                </Company>
              </destination_record>
            </conflict>
            <conflict>
              <link linkId="0" srcId="FI" srcName="Company" destId="PI"
destName="LeadStatus"/>
              <record source="4017" destination="4018" duplicate="0"/>
              <source_record>
                <LeadStatus table="PI" recId="x0000000000000fb1">
                  <Date>2009-09-15</Date>
                  <RepID>000100002</RepID>
                  <CoGrp>1</CoGrp>
                  <CoNo>12631</CoNo>
                  <!-- … -->
                </LeadStatus>
              </source_record>
              <destination_record>
                <LeadStatus table="PI" recId="x0000000000000fb2">
                  <Date>2009-09-15</Date>
                  <RepID>000100002</RepID>
                  <CoGrp>1</CoGrp>
                  <CoNo>12632</CoNo>
                  <!-- … -->
                </LeadStatus>
              </destination_record>
            </conflict>
          </conflicts>
          <results>
            <result>
              <FI typ="" src="4294979927" dst="4294979928" result="source"/>
            </result>
            <result>
              <XF typ="child" src="454" result="relink"/>
            </result>
            <result>
              <PI typ="child" src="4017" dst="4018" result="source"/>
            </result>
          </results>
        </merge>
      </response>
```

In this example the source person record is identified via the unique record ID (4294967297) and is merged into the destination record (ID=4294967285). The remaining person contains the field values of the original destination person. Existing child records of the source person are moved to the destination person.

```
<request>
        <merge tablename="Person" mode="destination">
          <source_record recId="4294967297"/>
          <destination_record recId="4294967285"/>
        </merge>
      </request>
      <response>
        <merge tablename="Person" mode="destination"/>
      </response>
```

<metainfo>

The `<metainfo>` command returns the Meta information of a table, such as the fields of a table and their attributes and the indices of the table.

The response nodes `<Flags name="FieldAttributes">` and `<Flags name="FieldFormat">` are primarily intended for the update support for error analysis.

<metainfo/>	
Appearance	<metainfo> </metainfo>
Attributes	table tablename flags include fields mode transaction if any Custom Attribute
Contents	(No child elements)
May occur in	<request>
Remarks	none

```
<request>
        <metainfo tablename="Company"/>
      </request>
      <response>
        <metainfo tablename="Company">
          <Flags name="FieldAttributes">
            <Flag name="InputHook" value="x00000001"/>
            [...]
          </Flags>
          <Flags name="FieldFormat">
            <Flag name="DigitGrouping" value="x00000001"/>
            [...]
          </Flags>
          <Company index="4" id="FI" name="Firma" customname="" basename=""
dbname="FI" count="168">
            <fields>
              <CoGrp fid="0" fnr="0" name="FiGr." dbname="ID" type="S" len="4"
ilen="2" cat="0" ucat="-1" attr="x20000000" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
              <CoNo fid="1" fnr="1" name="FiNr." dbname="ID" type="L" len="9"
ilen="4" cat="0" ucat="-1" attr="x40000020" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
```

```
              <Company fid="2" fnr="2" name="Firma" dbname="Firma" type="C"
len="120" ilen="240" cat="0" ucat="-1" attr="x1" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
              <Synonym fid="3" fnr="3" name="Synonym" dbname="Syno" type="C"
len="30" ilen="60" cat="0" ucat="-1" attr="x0" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
             <Division fid="4" fnr="4" name="Division" dbname="Division" type="C"
 len="40" ilen="80" cat="0" ucat="-1" attr="x0" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
              <Country fid="5" fnr="5" name="Land" dbname="Land" type="K" len="80"
 ilen="2" cat="2" ucat="-1" attr="x1" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
            </fields>
            [...]
            <indexes>
              <index id="1" type="p" autonumbered="1">
                <field fid="0" fnr="0" name="CoGrp" type="S" len="4" ilen="2"
cat="0" ucat="-1" attr="x20000000" ofmt="x0" field_class="x1000"
text_class="x2000000"/>
              <field fid="1" fnr="1" name="CoNo" type="L" len="9" ilen="4" cat="0"
 ucat="-1" attr="x40000020" ofmt="x0" field_class="x1000" text_class="x2000000"/>

              </index>
              [...]
            </indexes>
          </Company>
        </metainfo>
      </response>
```

Attribute	Meaning
fid	Unique ID of field in Aurea CRM
fnr	Zero based index of field in data base
name	Field name (output in current language)
dbname	Column name of database table
type	See Field types.
len	Maximum input / output length
ilen	Internal length in bytes, e.g. if type = L and ilen = 8 then it is a 64Bit integer, if ilen=4 then it is a 32 bit integer
cat	Catalog number
ucat	Parent catalog number, if catalog is a dependent catalog
attr	Field attributes, see list in appendix Field Attributes.
ofmt	Field output format attributes, see list in appendix Field output formats.
Text_class	Origin of language dependent fieldname, see list in appendix FieldTypes and Categories.

Attribute	Meaning
cid	IDs for tables and fields
ccat	IDs for catalog fields
core	IDs for core fields. If core=all, the values for all verticals are returned, if core='bb,bc', only the values for the BTB and OTC verticals are returned.

<mp>

The `<mp>` command facilitates generation of arbitrary XML content.

<mp/>	
Appearance	<mp> </mp>
Attributes	if output test transaction any Custom Attribute
Contents	(Any Content)
May occur in	<request>
Remarks	See Message Processing.

<nop>

The `<nop>` command allows testing if interface is "up and running" – in case of success the request does not return any data and therefore it is the easiest method of testing.

<nop/>	
Appearance	<nop> </nop>
Attributes	if transaction any Custom Attribute
Contents	(No child elements)

<nop/>	
May occur in	<request>
Remarks	none

```
<request>
        <nop/>
      </request>
      <response>
        <nop/>
      </response>
      In principle, no command is needed for the "up and running" test. However,
then the following error is returned:
      <request/>
      <response>
        <return type="error" func="C_Portal::ProcessXml">
          <ecode>-10049</ecode>
          <etext>No command found in document</etext>
        </return>
      </response>
```

<putdoc>

The `<putdoc>` command imports documents into the document tables D1 or D2.

<putdoc/>	
Appearance	<putdoc> </ putdoc >
Attributes	if transaction any Custom Attribute
Contents	links? rawdata? xmldata? filedata?
May occur in	<request>
Remarks	The document itself can either be specified as BASE64 encoded blob or read from a file (based on file name).

In this example a text file is imported into the D1 table.

The document itself is provided as a base64-encoded string within the `<rawdata>` element. The fields `<Name> and <Keyword>` are mandatory.

The response returns the document reference (A...for Documents from D1).

```
      <request>
        <putdoc>
          <Name>Testdatei.txt</Name>
          <Keyword>Putdoc_Test</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
      </request>
      <response>
        <!-- -->
```

```
        <putdoc>
          <docid>A1-333</docid>
        </putdoc>
      </response>
```

In order to import additional field values into the document record as well, you have to add an `<import>` command after the `<putdoc>` command to update the record created by the `<putdoc>` command.

```
      <request>
        <putdoc>
          <Name>Testdatei.txt</Name>
          <Keyword>Putdoc_Test</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
        <import>
          <fields>
            <Documents>
              <links>
                <link tablename="Documents" id="$lastRecId"/>
              </links>
              <Private>false</Private>
              <DocClass>Textfile</DocClass>
              <Owner>Own RepName</Owner>
              <FreeC1>Test</FreeC1>
              <!-- … -->
            </Documents>
          </fields>
        </import>
      </request>
```

In this example a text file is imported into the D1 table `(<putdoc>)`. The following `<import>` command creates a document link record (D3) linked to this D1 record and to the company declared in the `<links>` section (if the company record exists and is unique).

```
      <request>
        <putdoc>
          <Name>Testdatei.txt</Name>
          <Keyword>Putdoc_Test</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
        <import>
          <fields>
            <DocumentLink>
              <links>
                <Company>
                  <Company>My PutDoc Company</Company>
                </Company>
                <link tablename="Documents" id="$lastRecId"/>
              </links>
            </DocumentLink>
          </fields>
        </import>
      </request>

      <response>
        <putdoc>
          <docid>A1-335</docid>
        </putdoc>
        <import>
        <return table="D3" tablename="DocumentLink" id="4294967737" type="insert">

            <links>
              <link table="FI" id="4294979928" linkId="-1"/>
             <link table="D1" tablename="Documents" id="4294967631" linkId="-1"/>

            </links>
```

```
          </return>
        </import>
      </response>
```

XML documents need not be base64-encoded, they can be embedded as inline XML (element <xmldata>).

```
      <request>
        <putdoc>
          <Name>HelloWorld.xml</Name>
          <Keyword>Putdoc_Test</Keyword>
          <xmldata>
            <root>
              <hello attr="true">world</hello>
            </root>
          </xmldata>
        </putdoc>
      </request>
      <response>
        <putdoc>
          <docid>A1-336</docid>
        </putdoc>
      </response>
```

In order to import documents directly from file system, use <filedata>. In this case no other elements are needed in the request:

```
      <request>
        <putdoc>
          <filedata>C:\MyFiles\Textfile.txt</filedata>
        </putdoc>
      </request>
      <response>
        <putdoc>
          <docid>A1-338</docid>
        </putdoc>
      </response>
```

This request combines elements of the <query> and the <putdoc> commands:

With the definitions provided in the <tables> and <condition> elements a query for companies is executed. Afterwards the document is imported into the D2 table as a child record of the company record found by the query.

The document itself is provided as a base64-encoded string within the <rawdata> element. The fields <Name> and <Keyword> are mandatory.

The response returns the document reference (K...for CustomerDocuments)

```
      <request>
        <putdoc>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="My Putdoc
 Company"/>
          </condition>
          <Name>Testdatei.txt</Name>
          <Keyword>Putdoc_Test</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
      </request>
      <response>
        <putdoc>
          <docid>K1-131</docid>
        </putdoc>
      </response>
```

Example <putdoc> - Import documents into the D2 document table

Note: The result of the query has to be just one unique record.

If no record is found, this request is equivalent to the request from the example in Error! Reference source not found. and only the document is imported into the D1 table.

If more than one record is found, the following error is returned:

```
<response>
  <putdoc>
    <return type="error" func="C_Portal::XmlProcessPutDocument">
      <ecode>-10042</ecode>
      <etext>Link record not unique</etext>
      <table>FI</table>
    </return>
  </putdoc>
</response>
```

Example `<putdoc>` - Response, if link record is not unique

In addition to the example before, this request contains the `<doclinks>` element:

```
<request>
  <putdoc>
    <tables>
      <table tablename="Company"/>
    </tables>
    <condition>
      <cond tablename="Company" fieldname="Company" op="=" value="My Putdoc
 Company"/>
    </condition>
    <doclinks>
      <field tablename="Contact" fieldname="Document1" id="4294967631"/>
    </doclinks>
    <Name>Testdatei.txt</Name>
    <Keyword>Putdoc_Test</Keyword>
    <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
  </putdoc>
</request>
<response>
  <!-- … -->
  <putdoc>
    <docid>K1-133</docid>
    <doclinks>
      <doclink tablename="Contact" fieldname="Document1" id="4295044745"/>
    </doclinks>
  </putdoc>
</response>
```

A record from a declared table (@tablename) is referenced by its record ID (@id). The document reference of the imported document is written into the declared field (@fieldname) of this record.

In this example: The text file is imported to the D2 table as a child of the company "My Putdoc Company". The contact record with record ID 4294967631 is read and the document reference (K1-133) is written into the field Document1 of this contact.

Note: Normally the record ID is specified by a variable ($lastRecId). e.g.: from a preceding `<query>` command.

If the record declared in the `<doclinks>` element does not exist, the following error is returned:

```
<response>
  <!-- … -->
```

```
        <putdoc>
          <docid>K1-134</docid>
          <doclinks>
            <return table="MA" id="4295044746" type="error"
func="C_Portal::WriteRecord">
              <ecode>-10004</ecode>
              <etext>Cannot create cursor</etext>
              <cursorid>-30</cursorid>
              <recordid>4295044746</recordid>
              <type>write</type>
            </return>
          </doclinks>
        </putdoc>
      </response>
```

In order to update an existing D1 document via <putdoc> you explicitly have to read the document via <links>, since <putdoc> doesn't provide a matchup-mechanism.

```
      <request hello="world">
        <putdoc>
          <links>
            <Documents>
              <StatNo>10</StatNo>
              <SeqNo>19262</SeqNo>
            </Documents>
          </links>
          <Name>test3.txt</Name>
          <Keyword>D1_Update_12484</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
      </request>
```

In order to update an existing D2 document via <putdoc> you explicitly have to read the document via <links>, since <putdoc> doesn't provide a matchup-mechanism. Additionally you have to specify the company/person context, because without this context a new D1 document would be created instead updating the existing D2 document.

```
      <request>
        <putdoc>
          <tables>
            <table tablename="Person"/>
          </tables>
          <condition>
           <cond tablename="Person" fieldname="LastName" op="=" value="Hofmann"/>

          </condition>
          <links>
            <CustomerDocuments>
              <StatNo>1</StatNo>
              <SeqNo>128</SeqNo>
            </CustomerDocuments>
          </links>
          <Name>test3.txt</Name>
          <Keyword>D2_Update_12485</Keyword>
          <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
        </putdoc>
      </request>
```

<query>

The <query> command defines a read operation over one or more tables.

<query/>	
Appearance	<query> </query>
Attributes	labels maxrecords reverse select skiprecords lenient_filter chunked The following attributes are solely valid, if chunked = "true". back chunk_size clear hardcache navigable offset page qid any Command Attribute if any Custom Attribute
Contents	tables links? fields? condition? sortlist? custom_sortlist?
May occur in	<request>
Remarks	none

In this example all company records are returned that match the declared condition (field Company starts with "My Company"). The values of the fields declared are returned in the response.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields tablename="Company" fields="Company,Synonym"/>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="My
Company*"/>
          </condition>
        </query>
      </request>
      <response>
        <query>
          <Company tableshort="FI" id="4294967394">
            <Company>My Company 1</Company>
            <Synonym/>
          </Company>
          <Company tableshort="FI" id="4294967399">
            <Company>My Company 2</Company>
            <Synonym>my</Synonym>
          </Company>
          <!-- any other company records -->
        </query>
      </response>
```

In this example all company records are returned that match the declared condition for company table (Company starts with "My Company") and within each Company all person records that match the condition for person table (LastName starts with "M").

Note: Normally in the <tables> section the dependency of the tables is declared. Only one root table is possible.

It is not possible to mix conditions from different tables within one <condition> section. If desired for each table a separate condition section has to be declared.

```
      <request>
        <query>
          <tables>
            <table tablename="Company">
              <table tablename="Person"/>
            </table>
          </tables>
          <fields>
            <field tablename="Company" fieldname="CoGrp"/>
            <field tablename="Company" fieldname="CoNo"/>
            <field tablename="Company" fieldname="Company"/>
            <field tablename="Person" fieldname="LastName"/>
            <field tablename="Person" fieldname="FirstName"/>
          </fields>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="My
Company*"/>
          </condition>
          <condition>
            <cond tablename="Person" fieldname="LastName" op="=" value="M*"/>
          </condition>
        </query>
      </request>
      <response>
        <query>
          <Company tableshort="FI" id="4294975508">
            <CoGrp>1</CoGrp>
            <CoNo>8212</CoNo>
            <Company>My Company 1</Company>
          </Company>
          <Company tableshort="FI" id="4294979923">
            <CoGrp>1</CoGrp>
            <CoNo>12627</CoNo>
            <Company>My Company 2</Company>
            <Person tableshort="KP" id="4294977464">
              <LastName>Maier</LastName>
              <FirstName>Sabine</FirstName>
            </Person>
            <Person tableshort="KP" id="4294977464">
              <LastName>Mader</LastName>
              <FirstName>Georg</FirstName>
            </Person>
            <!– any other person records within this company -->
          </Company>
          <!– any other company records -->
        </query>
      </response>
```

In the example below we have a filter condition which causes a conversion error (<etext>Conversion error</etext>) because the condition compares a numeric field fieldname="FreeN1" with an alphanumeric value (value="update").

```
      <request>
        <query maxrecords="5">
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields tablename="Company" fields="1,2,3,4,5,6,7"/>
          <condition>
            <cond tablename="Company" fieldname="FreeN1" op="!=" value="update"/>

          </condition>
        </query>
```

```
      </request>
      <response>
        <query maxrecords="5">
          <return type="error" func="C_Portal::CheckField">
            <ecode>-10023</ecode>
            <etext>Conversion error</etext>
            <field table="FI" tablename="Company" fid="63" fnr="63" type="L"
fieldname="FreeN1"/>
            <description>parse error</description>
            <value/>
          </return>
        </query>
      </response>
```

In the example below we use `lenient_filter="true"` and therefore the incorrect condition `fieldname="FreeN1"` is ignored and only the condition on `fieldname="Company"` is executed.

```
      <request>
        <query maxrecords="5" lenient_filter="true">
          <tables>
            <table tablename="Company" />
          </tables>
          <fields tablename="Company" fields="2"/>
          <condition>
            <cond tablename="Company" fieldname="FreeN1" op="!=" value="update"/>

            <lop op="and"/>
           <cond tablename="Company" fieldname="Company" op="()" value="update"/>

          </condition>
        </query>
      </request>
```

Chunked read

Reading in chunks allows for more efficient retrieving of (possible) huge data sets and navigating in these data sets. First only the record-IDs are read (up to maxrecords). Subsequent queries return the next "page" (determined by chunk_size) of records by referencing the original query via a query-ID (qid).

In the example below the data two chunks of 10 records are read.

```
      <request>
        <query maxrecords="50" qid="query_chunked_FI.xml" chunked="true"
chunk_size="10">
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields tablename="Company" fields="CoGrp,CoNo,Company,Country"/>
        </query>
      </request>
      <request>
        <query qid="query_chunked_FI.xml"/>
      </request>
```

If you don’t specify a qid in the requests, interface generates a unique qid to be used in the following queries.

```
      <request>
        <query maxrecords="50" chunked="true" chunk_size="10">
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields tablename="Company" fields="CoGrp,CoNo,Company,Country"/>
        </query>
      </request>
      <response>
        <query maxrecords="50" chunked="true" chunk_size="10"
qid="QVRQQzI1ODJfdXBkYXRlLndlYi5zZXJ2aWNlcy50ZXN0LmV4ZV9fMjAxMi
0xMC0xNiAxMzowNjoxMS44NzNfNjkwMTg5MDMwX1A3Njg4X1Q4NzYwXykjPmFsVi5SSXE=" count="50">
```

```
      …
      </response>
```

In order to read the next page you have to do something like in the request shown below.

```
      <request>
        <query qid="QVRQQzI1ODJfdXBkYXRlLndlYi5zZXJ2aWNlcy50ZXN0LmV4ZV9fMjAxM
i0xMC0xNiAxMzowNjoxMS44NzNfNjkwMTg5MDMwX1A3Njg4X1Q4NzYwXykjPmFsVi5SSXE="/>
      </request>
```

`chunk_size` can be set on every query; if not set explicitly on the "following" queries, the last value of is `chunk_size` used.

If `navigable = true` is not specified the initial list of record-IDs is reduced by the records returned by each query. The last chunk of data is marked with `last_chunk="true"`. If all records have been read "Query already completed" is returned.

```
      <response>
        <query qid="query_chunked_FI.xml" chunk_size="10" count="50"
last_chunk="true">
          <query qid="query_chunked_FI.xml">
            <return type="error" func="C_Portal::XmlExecuteQueryChunked">
              <ecode>-10090</ecode>
              <etext>Chunked read error</etext>
              <description>Query already completed</description>
            </return>
          </query>
        </response>
```

If `navigable = true` the record-IDs are kept at the server until the timeout is reached (timeout = 60 minutes after last use); by default navigation direction is forward, unless back=true. The starting point of the navigation can be set via the offset or page attribute.

count returns the total number of records (maxrecords) from the initial query; if navigable = true the current offset is returned in the pos attribute.

<refresh>

The `<refresh>` command refreshes the Aurea CRM internal caches (e.g. format cache, catalogue cache, rep cache). Please note, that the refresh request requires SU user privileges.

<refresh/>	
Appearance	<refresh> </refresh>
Attributes	catalogs configuration formats protocols reps transaction if any Custom Attribute
Contents	(No child elements)

<refresh/>	
May occur in	<request>
Remarks	none

```
<request user="SU" pwd="__SUPWD">
  <refresh/>
</request>
<response user="SU">
  <refresh/>
</response>
```

<row_export>

The `<row_export>` command allows for exporting data through the use of an export format, its functionality is equivalent to the Aurea CRM export module. Currently, the exported data cannot be returned in the XML response (it can only be written to the export file).

<row_export/>	
Appearance	<row_export> </row_export>
Attributes	debug format_name output_file
Contents	(No child elements)
May occur in	<request>
Remarks	

<row_import>

The `<row_import>` command allows for importing data through the use of an import format, its functionality is equivalent to the Aurea CRM import module. Import data can be specified using a file, or as XML using a single `<data>` block or split into multiple "lines" using `<rows>` with `<row>` sub-nodes.

<row_import />	
Appearance	<row_import> </row_import>
Attributes	debug format_name input_file record_separator field_separator boundary_separator
Contents	rows/row data

<row_import />	
May occur in	<request>
Remarks	The response <r> of the <row_import> command includes information about what happened with the data record. <r ipos=[integer] irelpos=[integer] status=[integer] table=[string] id=[long integer]/> iposhierarchical position, 0 = root table, 1=child table first level, 2=child table second level ... irelpos hierarchical position of the table if imported multiple times status0 = not processed, e.g. if no data record for this table is included in the import data 1 = record is matched 2 = record is inserted 4 = record is changed 8 = record is deleted 16 = record is rejected due to an error 32 = similar records are found 64 = matchup logic has been aborted with no success (example: primary key is specified, and the record is not found) 128 = record went through manual matchup (not possible in non-interactive scenarios like u7.interface) 256 = match via external key The response <e> of the <row_import> command includes detailed information about import errors for each row, the concerned field is further detailed in the <field> element. <e rc=[integer]> where rc references the row in which the error occurred <field table=[string] tablename=[string] fid=[integer] fnr=[integer] type=[string] fieldname==[string]/> The response <msg> provides an error message.

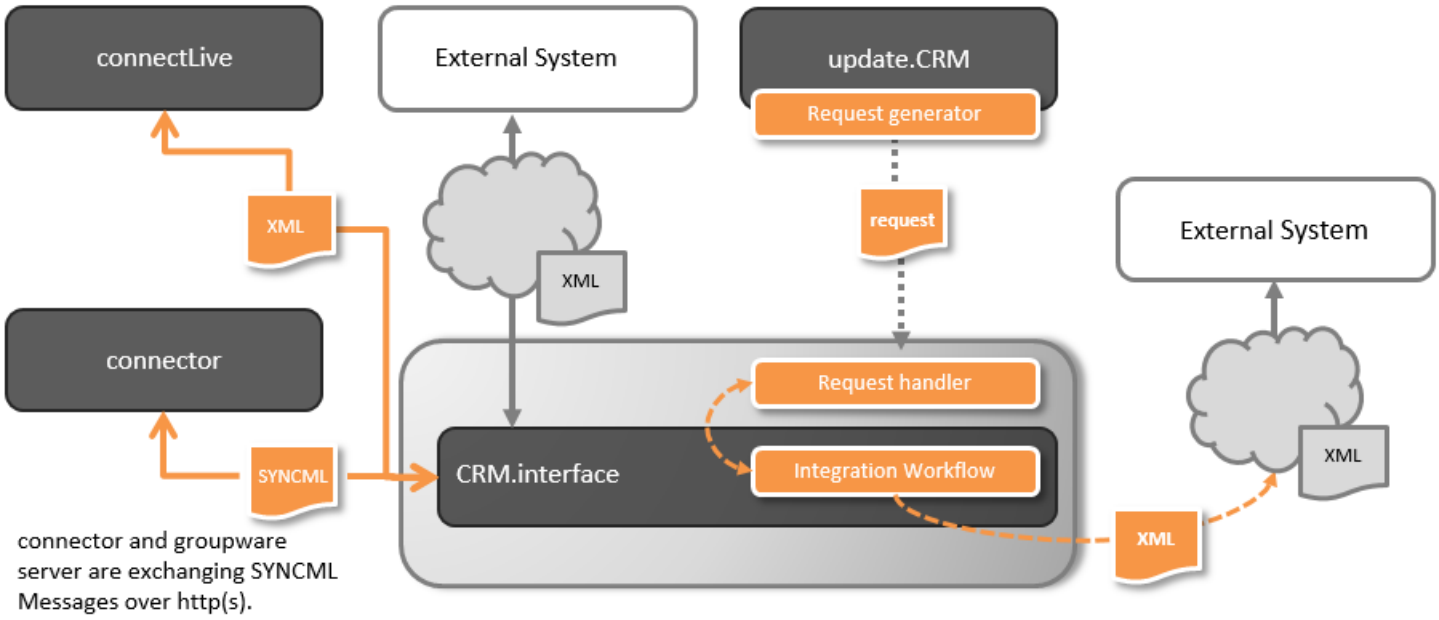
In the example below all attribute definitions (fields, separator, data ...) are taken from the Aurea CRM import format.

```
<request>
  <row_import format_name=" Import format Company"/>
</request>
```

In the example below the input file definition of the Aurea CRM import format is overwritten by the request. Path can be absolute or – as defined in this example via using the $ syntax – relative to the installation directory of CRM.interface.

```
      <request>
        <row_import format_name=" Import format Company"
input_file="$\Documents\import_companies.txt"/>
      </request>
```

The example below illustrates how to override the separators defined in the Aurea CRM import format via the `<row_import>` request. Please note that you have to escape the separators according to the XML language definition ("
" is equivalent to Lf). The separator settings are equivalent to the definition in the Aurea CRM format as depictured below.




```
      <request>
        <row_import format_name=" Importformat Firmen"
input_file="$\Documents\import_firmen.txt" record_separator="&#10;"
field_separator=";" boundary_separator=","/>
      </request>
```

The example below illustrates how data can be passed to the `<row_import>` request directly. In this case you have to overwrite the data source defined in the Aurea.CRM format via `input_file=""`.

If data is defined in a CDATA section you can send un-escaped data (e.g. you do not need to escape special characters with "&"). Otherwise you have to ensure that the data is in valid XML format.

```
      <request>
        <row_import format_name="Importformat CompanyPerson" input_file=""
record_separator="&#10;" field_separator=";">
          <!-- empty @input_file because we have one in the format-->
          <rows>
            <row>Test AG;1020;Wien;Hauptstrasse 68;782612;Joe;Doe</row>
            <row>Test AG;1020;Wien;Hauptstrasse 68;782612;Tom;Tester</row>
            <row><![CDATA[ Test & Co GmbH;2340;Mödling;Einbahnstrasse
77;32145]]></row>
          </rows>
        </row_import>
      </request>
```

The next example executes basically the same import request as in example for <row_import> - import data in request, but now all data is wrapped into one <data> element.

```
      <request>
        <row_import format_name=" Importformat CompanyPerson" input_file=""
record_separator=";" field_separator=",">
          <!-- empty @input_file because we have one in the format-->
          <data>Test AG,1020,Wien,Hauptstrasse 68,782612,Joe,Doe;Test
AG,1020,Wien,Hauptstrasse 68,782612,Tom,Tester;Test GmbH,2340,Mödling,Einbahnstrasse
 77,32145</data>
        </row_import>
      </request>
```

In the next example all data is wrapped into one CDATA section.

```
      <request>
        <row_import format_name="Importformat CompanyPerson" input_file=""
record_separator=";" field_separator=",">
          <!-- empty @input_file because we have one in the format-->
          <data><![CDATA[Test AG,1020,Wien,Hauptstrasse 68,782612,Joe,Doe;Test
AG,1020,Wien,Hauptstrasse 68,782612,Tom,Tester;Test & Co
GmbH,2340,Mödling,Einbahnstrasse 77,32145]]></data>
        </row_import>
      </request>
```

The ipos attribute means the hierarchical position of the table, where 0 is the root table, 1 is a child table, 2 is a grandchild table etc. irelpos is an index of the table if imported multiple times.

The status attribute says, if the record is updated or newly created (status = 2) etc. status 0 is returned for the last data set, because in this data set only a company (and no person) is specified. The id attribute represents the id of the record in the Aurea CRM data base. For further details see the remarks in the description of the <row_import> command.

```
      <response>
        <!-- … -->
        <row_import format_name="pvcs_67776_row_import" record_separator="&#xA;"
 field_separator=";">
          <r row="1" ipos="0" irelpos="0" status="261" table="FI"
id="296352743515"></r>
          <r row="2" ipos="0" irelpos="0" status="261" table="FI"
id="296352743515"></r>
          <r row="3" ipos="0" irelpos="0" status="16" table="FI" id="0">
            <e rc="3">
      <field table="FI" tablename="Company" fid="63" fnr="63" type="L"
fieldname="FreeN1"></field>
              <msg>negative value</msg>
            </e>
          </r>
          <r row="4" ipos="0" irelpos="0" status="16" table="FI" id="0">
            <e rc="4">
      <field table="FI" tablename="Company" fid="123" fnr="123" type="D"
fieldname="MatchupDate"></field>
              <msg>invalid month</msg>
            </e>
          </r>
          <r row="5" ipos="0" irelpos="0" status="261" table="FI"
id="296352743515"></r>
        </row_import>
      </response>
```

```
      <request>
        <row_export format_name="Importformat CompanyPerson"
output_file="$\Documents\ Export_Firmen_und_Personen.txt "/>
      </request>
      <response>
        <!-- … -->
        <row_export format_name=" Importformat CompanyPerson" output_file="$\
```

```
Documents\Export_Firmen_und_Personen.txt"/>
      </response>
```

The example above illustrates a <row_export> command, where the resulting file is written to the "\documents" subfolder of the CRM.interface installation directory.

Note: Currently there is no possibility to specify separators in the <row_export> directly, this may only be done in the Aurea CRM export format.

<sleep>

The <sleep> command causes interface to sleep for a given number of milliseconds. This request might be useful in testing scenarios.

<sleep>	
Appearance	<sleep> </sleep>
Attributes	msec transaction if any Custom Attribute
Contents	(No child elements)
May occur in	<request>
Remarks	Use with caution and consider that time-outs may occur on end-points.

```
<request>
  <sleep msec="1000"/>
</request >
<response>
  <sleep msec="1000"/>
</response>
```

<status>

The <status> command allows for testing, if interface is up and running – it returns various data (Operating System etc.) where interface is running and internal status data about CRM.interface.

<status/>	
Appearance	<status> </status>
Attributes	datamodel profiling_snapshot if transaction any Custom Attribute

<status/>	
Contents	(No child elements)
May occur in	<request>
Remarks	Content of the response may be changed without notice in further versions.

```
<request>
  <status/>
</request>
```

<test>

The `<test>` command is for internal purpose only. Do not use except when explicitly instructed to do so.

<transaction>

The `<transaction>` command is used to begin and end transactions explicitly.

<transaction/>	
Appearance	<transaction> </transaction>
Attributes	cmd if any Custom Attribute
Contents	(No child elements)
May occur in	<request>
Remarks	For further details see Working with transactions.

<update>

The `<update>` command allows updating multiple records from a single table (like in the CRMwin service module).

<update/>	
Appearance	<update> </update>
Attributes	any Command Attribute if any Custom Attribute
Contents	tables links? fields? condition? sortlist? custom_sortlist?
May occur in	<request>
Remarks	The <update> command is executed as a <query> and the resulting records are updated with the specified fields. Be sure that the set of records to be updated are properly limited by conditions or links, otherwise you might update too many or even all records in a table.

```
      <request>
        <update>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="Test
interface 1"/>
          </condition>
          <fields>
            <Company>
              <Synonym>update</Synonym>
              <FreeN1>10</FreeN1>
            </Company>
          </fields>
        </update>
      </request>
      <response>
        <update>
          <return table="FI" tablename="Company" id="4294979929" type="update"/>
          <return table="FI" tablename="Company" id="4294979937" type="update"/>
        </update>
      </response>
```

In the example above all company records that match the condition (Company starts with "update-test") is updated with the fields Synonym and FreeN1.

The response returns the record id of each updated record.

<xquery>

The `<xquery>` command executes a "Query" or "Read Engine" format.

<xquery />	
Appearance	<xquery> </xquery>
Attributes	name type any Command Attribute if any Custom Attribute
Contents	par*
May occur in	<request>
Remarks	Regarding par* see example below.

In the example the format with type "Query" (defined in CRMweb) and the declared name ("My Web Query") is read and executed. The response does not return the format definition but the records which are the result of executing the query.

```
<request>
  <xquery name="My Web Query" type="32"/>
</request>
<response>
  <xquery name="My Web Query" type="32">
    <Company tableshort="FI" id="4294967332">
      <Company>My Company 1</Company>
      <Synonym>internal</Synonym>
      <Country>Österreich</Country>
      <Person tableshort="KP" id="4294968027">
        <LastName>Maier</LastName>
        <FirstName>Sabine</FirstName>
      </Person>
      <Person tableshort="KP" id="4294968028">
        <LastName>Müller</LastName>
        <FirstName>Stefan</FirstName>
      </Person>
      <!-- … -->
    </Company>
    <Company tableshort="FI" id="4294975524">
      <!-- … -->
    </Company>
  </xquery>
</response>
```

In this case the format with type "Read Engine" is executed and the response also returns the resulting records. The structure of the response is similar to the example before.

```
<request>
  <xquery name="My Read Engine" type="34"/>
</request>
```

If the "Query" or "Read Engine" format contains parameters, they need to be specified in `<par>` elements. Each parameter is referenced with fieldname and its respective table prefix.

```
<request>
  <xquery name="Test Read Engine with parameter" type="34">
    <par prefix="FI" fieldname="Country">Österreich</par>
```

```
    </xquery>
  </request>
```

If parameters are unresolved, the following error is returned:

```
  <response>
    <!--... -->
    <xquery name="Test" type="34">
      <return type="error" func="C_CursorMD::ResolveFilterParameters">
        <ecode>-10081</ecode>
        <etext>Unresolved parameter</etext>
        <prefix>FI</prefix>
        <fieldname>Country</fieldname>
        <index>0</index>
      </return>
    </xquery>
  </response>
```

Common Elements

List of common elements XML syntax reference.

<table>

The `<table>` element is used to reference a single table. If a table is read multiple times in a query aliases have to be used.

<table/>	
Appearance	<table> </table>
Attributes	alias allfields flags flags2 index keys keysend linkId maxrecords readtype relindex setlen
Contents	<table>
May occur in	<request>
Remarks	none

```
  <request>
    <query>
      <tables>
        <table tablename="Company">
          <table tablename="Person">
            <table tablename="Contact"/>
          </table>
          <table tablename="Contact" alias="Contact2"/>
        </table>
      </tables>
      <condition>
        <cond tablename="Contact" alias="Contact2" fieldname="PeGrp" op="="
value="0"/>
        <cond tablename="Contact" alias="Contact2" fieldname="PeNo" op="="
value="0"/>
      </condition>
      <fields tablename="Company" fields="Company"/>
      <fields tablename="Person" fields="LastName"/>
      <fields tablename="Contact" fields="Contact,Date,Time"/>
```

```
          <fields tablename="Contact" alias="Contact2"
fields="Contact,Text,Subject"/>
        </query>
      </request>
```

In this example the contact table is read twice, first person-related and second company-related.

In the second case, without the condition, all contacts related to the company would be returned (also contacts related to any person from that company). The condition serves to return only the contacts directly related to the company.

Note: In the second case the alias name is used as tablename.

```
<Contact2 tableshort="MA" id=" 4294967394">
  <Contact>Letter</Contact>
  <Text>any text</Text>
  <Subject>any text</Subject>
</Contact2>
```

Because the combination of tablename and alias has to be unique, one additional reference to the same table can also use the tablename as alias in order to have the same "tablename" in the response.

<field>

The `<field>` element is used to reference a single field. This can be used if only a single field is needed or for fields that need to have differing flags set (like read as external key etc.).

<field/>	
Appearance	<field> </field>
Attributes	table tablename fid fieldname extkey (bool)
Contents	(no contents)
May occur in	< fields >
Remarks	None

In this case different attributes (e.g. extkey) can be set separately for each field.

```
<request>
<query>
  <tables>
    <table tablename="Company"/>
    <table tablename="Person"/>
  </tables>
  <fields>
    <field tablename="Company" fieldname="Company"/>
    <field tablename="Company" fieldname="Synonym"/>
    <field tablename="Company" fieldname="Country" extkey="true"/>
    <field tablename="Company" fieldname="LeadStatus" extkey=" false" />”
    <field tablename="Person" fieldname="LastName"/>
    <field tablename="Person" fieldname="FirstName"/>
  </fields>
```

```
</query>
</request>
```

<link>

The <link> element is used to reference a linked record (or "record link") by record id.

<link/>	
Appearance	<link> </link>
Attributes	table tablename recId linkId optional id
Contents	(no contents)
May occur in	<links>
Remarks	none

In this example the linked company record is referenced by its record id (attribute recId). The imported person record is linked to this company.

```
<request>
  <import>
    <fields>
      <Person>
        <links>
          <link tablename="Company" recId="4294967297"/>
        </links>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
      </Person>
    </fields>
  </import>
</request>

<request>
  <import>
    <fields>
      <Contact>
        <links>
          <link tablename="Company" recId="4294967297"/>
          <link tablename="Activity" recId="4294967565" optional="true"/>
        </links>
        <Contact>Brief</Contact>
        <Subject>Imported via CRM.interface</Subject>
      </Contact>
    </fields>
  </import>
</request>

<request>
  <import>
    <fields>
      <ProblemResolution>
        <links>
          <link table="FI" tablename="Company" recId="4294980423"/>
        <link table="FI" tablename="Company" recId="4294981501" linkId="1"/>

        <link table="FI" tablename="Company" recId="4294981502" linkId="2"/>
```

```
            <link table="FI" tablename="Company" recId="4294981503" linkId="3"/>

           </links>
           <No>4032010</No>
           <ProblemGroup>Software</ProblemGroup>
           <Problem>Bug</Problem>
         </ProblemResolution>
       </fields>
     </import>
   </request>
```

<links>

The <links> element is used to reference linked records (or "record links").

<links/>	
Appearance	<links> </links>
Attributes	(no attributes)
Contents	<link> (any record)
May occur in	<import> <query> <putdoc> <doclinks> (any record) <match>
Remarks	The record links can be specified via the <link> element (it by record id), or using by specifying a record.

```
<request>
  <import>
    <fields>
      <Person>
        <links>
          <Company>
            <Company>update software AG</Company>
            <Synonym>update</Synonym>
            <Country>Österreich</Country>
          </Company>
        </links>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
      </Person>
    </fields>
  </import>
</request>
```

In the example above the linked record (company) is specified by contents of certain fields. The person record is imported and linked to this company, if exactly one unique matching company record exists. Otherwise an error is returned.

Note: The link record has to be just one unique record.

In the next example more than one link record for the contact record is specified (company and activity). Usually the contact is only imported if unique company and activity record exist. In this case the activity link record is not mandatory (due to `optional="true"`).

```
<request>
  <import>
    <fields>
      <Contact>
        <links>
          <Company>
            <Company>update software AG</Company>
            <Synonym>update</Synonym>
            <Country>Österreich</Country>
          </Company>
          <Activity optional="true">
            <Activity>PR Event</Activity>
            <Level>1</Level>
          </Activity>
        </links>
        <Contact>Brief</Contact>
        <Subject>Imported via interface</Subject>
      </Contact>
    </fields>
  </import>
</request>
```

In the default data model a ticket record has more than one link to the company table (default link, end customer company, contact company and company billing address).

In the next example the ticket record is linked to 4 different companies, each specified by the relevant link id.

Note: The link ids can be checked up in the data model in the service module of CRMwin or with a suitable `<metainfo>` request.

```
<request>
  <import>
    <fields>
      <ProblemResolution>
        <links>
          <Company>
            <Company>update software AG</Company>
            <Synonym>default link</Synonym>
          </Company>
          <Company linkId="1">
            <Company>company for ticket end customer</Company>
            <Synonym>link no 1</Synonym>
          </Company>
          <Company linkId="2">
            <Company>company for ticket contact</Company>
            <Synonym>link no 2</Synonym>
          </Company>
          <Company linkId="3">
            <Company>company for billing address</Company>
            <Synonym>link no 3</Synonym>
          </Company>
        </links>
        <No>4032010</No>
        <ProblemGroup>Software</ProblemGroup>
        <Problem>Bug</Problem>
      </ProblemResolution>
    </fields>
```

```
        </import>
      </request>
      <response>
        <import>
          <return table="KM" tablename="ProblemResolution" id="4294967501"
type="insert">
            <links>
              <link table="FI" tablename="Company" id="4294980423" linkId="-1"/>
              <link table="FI" tablename="Company" id="4294981501" linkId="1"/>
              <link table="FI" tablename="Company" id="4294981502" linkId="2"/>
              <link table="FI" tablename="Company" id="4294981503" linkId="3"/>
            </links>
          </return>
        </import>
      </response>
```

<condition>

The <condition> element is used to describe a condition for one table.

<condition/>	
Appearance	<condition> </condition>
Attributes	(no attributes)
Contents	<cond> <lop>
May occur in	<query>
Remarks	none

<cond>

The <cond> element is used to describe a condition on a single field.

<cond/>	
Appearance	<cond> </cond>
Attributes	table tablename fid fieldname fieldname2 mnr name value extkey (bool) extkey2 mnr2 value2 op
Contents	(no contents)
May occur in	<condition> <lop>
Remarks	none

The example request below returns all companies located in Austria.

```
      <request>
        <query>
          <tables>
```

```
            <table tablename="Company"/>
          </tables>
        <fields tablename="Company" fields="Company, Synonym,Country,Street,Tel"/>

          <condition>
           <cond tablename="Company" fieldname="Country" op="=" value="Austria"/>

          </condition>
        </query>
      </request>
```

The request below returns all companies located in Austria and within these companies all person and contact records matching the condition (Person: LastName starts with "B" and Contact: contact type=Letter).

Note: A separate <condition> section has to be declared for each table.

```
      <request>
        <query>
          <tables>
            <table tablename="Company">
              <table tablename="Person"/>
              <table tablename="Contact"/>
            </table>
          </tables>
        <fields tablename="Company" fields="Company,S ynonym,Country,Street,Tel"/>

          <fields tablename="Person" fields="LastName, FirstName"/>
          <fields tablename="Contact" fields="Contact, Subject "/>
          <condition>
           <cond tablename="Company" fieldname="Country" op="=" value="Austria"/>

          </condition>
          <condition>
            <cond tablename="Person" fieldname="LastName" op="=" value="B*"/>
          </condition>
          <condition>
            <cond tablename="Contact" fieldname="Contact" op="=" value="Letter"/>

          </condition>
        </query>
      </request>
```

In the next example the condition is interpreted as follows: companies where LeadStatus=”Customer” AND Revenue>1000000 and Employees>250 AND Rep=current Rep AND (Country=Austria OR Germany).

Note: In order to combine conditions with the logical operator OR you have to include them into a `<lop value=”or”/>` element. By default `<cond>` elements on the top level are combined with the logical operator AND.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
         <fields tablename="Company" fields="Company,Synonym,Country,Street, el"/>

          <condition>
            <cond tablename="Company" fieldname="LeadStatus" op="="
value="Customer"/>
            <lop value="and">
           <cond tablename="Company" fieldname="Revenue" op=">" value="1000000"/>

             <cond tablename="Company" fieldname="Employees" op="=" value="250"/>
```

```
            </lop>
            <cond tablename="Company" fieldname="Rep" op="=" value="$curRep"/>
            <lop value="or">
           <cond tablename="Company" fieldname="Country" op="=" value="Austria"/>

           <cond tablename="Company" fieldname="Country" op="=" value="Germany"/>

            </lop>
          </condition>
        </query>
      </request>
```

In the example below all interest records are returned where `InterestGroup="sports" AND Interest="soccer"`. The child catalog value is specified by the attribute value, the parent catalog value by the attribute value2.

Note: For backwards compatibility, the catalog values can be combined into the value attribute by using ~ as separator. (e.g. value="soccer~sports"). But this is deprecated and should not be used because it might not be supported in upcoming versions of interface any more.

```
      <request>
        <query>
          <tables>
            <table tablename="Interests"/>
          </tables>
          <fields tablename="Interests" fields="InterestGrp,Interest"/>
          <condition>
           <cond tablename="Interests" fieldname="Interest" op="=" value="soccer"
value2="sports"/>
          </condition>
        </query>
      </request>
```

In the example below all companies are returned where the contents of the field FreeC1 and FreeC2 are equal (and not empty).

Note: In order to compare the contents of two fields in a condition, the first field has to be specified by the fieldname attribute, the second field by the fieldname2 attribute.

Note: It is not possible to compare fields with different field types.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields tablename="Company" fields="Company,FreeC1,FreeC2"/>
          <condition>
           <cond tablename="Company" fieldname="FreeC1" op="=" fieldname2="FreeC2"
/>
            <cond tablename="Company" fieldname="FreeC1" op="!=" value=""/>
            <cond tablename="Company" fieldname="FreeC2" op="!=" value=""/>
          </condition>
        </query>
      </request>
```

In example below all companies are returned where the contents of the field Rep equals the rep of the logged-in user.

```
<request>
  <query>
    <tables>
      <table tablename="Company"/>
    </tables>
    <fields tablename="Company" fields="Company,Rep"/>
    <condition>
      <cond tablename="Company" fieldname="Rep" op="=" value="$curRep" />
    </condition>
  </query>
</request>
```

In this example all companies are returned where the contents of the field `FreeD1` equals the current date plus 5 days.

Note: Further variables that can be used are described in the trigger section of the CRM.core Administrator Guide.

```
<request>
  <query>
    <tables>
      <table tablename="Company"/>
    </tables>
    <fields tablename="Company" fields="Company,FreeD1"/>
    <condition>
     <cond tablename="Company" fieldname="FreeD1" op="=" value="$curDay+5d"
/>
    </condition>
  </query>
</request>
```

CRM.interface supports conditions on the timestamps of field values. The example below illustrates how to find out when the value of a certain field was last modified (using the attribute lupd).

```
<request>
  <query>
    <tables>
      <table tablename="Company"/>
    </tables>
    <fields tablename="Company" fields="Company,FreeD1"/>
    <condition>
     <cond tablename="Company" fieldname="Synonym" op=">=" lupd="$curDay-2"
/>
    </condition>
  </query>
</request>
```

In this example all companies are returned where the content of the field Synonym was last modified within the last 2 days.

Note: In order to use time conditions as well, you have to add the time to the value of lupd separated by a comma using the time format hhmmssttt. e.g. lupd="$curDay-2,144500000" ... last modified since 2:45 p.m. 2 days ago.

Note: If no time is specified the condition is evaluated for the date only.

<sortlist>

The `<sortlist>` element is used to specify the sort order for one table.

<sortlist/>	
Appearance	<sortlist> </sortlist>
Attributes	(no attributes)
Contents	<sort>
May occur in	<query>
Remarks	none

<custom_sortlist>

The `<custom_sortlist>` element is used to describe post-processing sort criteria.

<custom_sortlist/>	
Appearance	<custom_sortlist> </custom_sortlist>
Attributes	(no attributes)
Contents	<sort>
May occur in	<query>
Remarks	none

Attributes

Attributes are designed to contain data related to a specific element.

Request Attributes

List of attributes for request element.

ixslt

ixslt	
Syntax	[ixslt = string]
May occur in	<request>
Description	A semicolon-delimited list of style sheets that are applied to the XML request before processing of commands starts.

log

log	
Syntax	[log = string]
May occur in	<request>
Description	A filename where the XML response is logged.

logmode

logmode	
Syntax	[logmode = (append | create)]
May occur in	<request>
Description	Whether to append to or truncate the file specified in the log attribute.

noerrorlog

noerrorlog	
Syntax	[noerrorlog = Boolean : false]
May occur in	<request>
Description	An indicator of whether to create the error and error import logs.

oxslt

oxslt	
Syntax	[oxslt = string]
May occur in	<request>
Description	A semicolon-delimited list of style sheets that are applied to the XML response after processing of commands has finished.

Session attributes

List of attributes related to session.

auth

auth	
Syntax	[auth = string]
May occur in	<request>, any Command element
Description	An authentication token that contains encrypted credentials (username and password). If specified, the user and pwd attributes are ignored. This token can be generated using the `update.mmUtilities.Cryptography` class in conjunction with configured users managed by the `mmUserConfig.exe` utility. If Force login encryption is enabled, a login can only be authenticated through this token. A login through the user and pwd attributes fails.

domain

domain	
Syntax	[domain = string]
May occur in	<request>, any Command element
Description	The domain name for single sign-on (SSO) authentication. **Note:** In an non-interactive scenario the domain/user combination as specified in the XML request cannot be verified against the current "active user" and so has to come from a trusted source (HTTP authentication for example).

impersonate

impersonate	
Syntax	[impersonate = string]
May occur in	<request>, any Command element
Description	A reference to a login user that is to be impersonated. See Impersonation on page 144.

lang

lang	
Syntax	[lang = string]
May occur in	<request>, any Command element
Description	The ISO639 language code for the language to be used. Alternatively, the Aurea CRM language index can be specified for backwards compatibility. See the xml:lang attribute.

xml:lang

xml:lang	
Syntax	[xml:lang = string]
May occur in	<request>, any Command element

xml:lang	
Description	The ISO639 language code for the language to be used. The following table lists the languages supported by Aurea CRM. ISO639indexLanguage de0German en1English fr2French es3Spanish pt4Portuguese nl5Dutch da6Danish it7Italian cs8Czech hu9Hungarian sk10Slovak pl11Polish el12Greek uk13Ukrainian sl14Slovenian ru15Russian sv16Swedish fi17Finnish no18Norwegian tr19Turkish hr20Croatian sr21Serbian ro22Romanian ja23Japanese zh24Chinese ko25Korean bg26Bulgarian th27Thai

langNo

langNo	
Syntax	[langNo = unsignedShort]
May occur in	<request>, any Command element
Description	The language number as defined in the Language table (table 00).

module

module	
Syntax	[module = string]
May occur in	<request>, any Command element
Values	interface connectlive connector webservice
Description	The name of the module to be assumed.

pwd

pwd	
Syntax	[pwd = string]
May occur in	<request>, any Command element
Description	The login password (in plain text).

user

user	
Syntax	[user = string]
May occur in	<request>, any Command element
Description	The login username. The user and pwd attributes can be used to authenticate a login in plain text if no encryption is desired (e.g. in a trusted local network) or necessary (e.g. if the request is transmitted using HTTPS). If no pwd attribute, but a domain attribute is given, single sign-on (SSO) authentication is used (see the domain attribute).

Common Attributes

List of attributes for common elements.

extkey (boolean)

extkey (boolean)	
Syntax	[extkey = Boolean : false]
May occur in	<field> < fields > <cond> <getcat> any Import field
Description	An indicator of whether the referenced (field-) value specifies an external key.
Remarks	Currently, this attribute is also used to specify an external key in other places. This is fixed in an upcoming version.

extkey (value)

extkey (value	
Syntax	[extkey = string]
May occur in	<link> destination_record source_record
Description	The value of an external key.
Remarks	Currently, this attribute is also used to specify whether an external key is referenced in other places. This is fixed in an upcoming version.

extkey_as_attr

extkey_as_attr (boolean)	
Syntax	[extkey_as_attr = Boolean : false]
May occur in	<field> < fields > <cond>
Description	An indicator of whether the external key of a catalog value is returned as its own attribute extkey on field level. If true, the external key is returned (as value of attribute extkey) as well as the catalog text in the response.
Remarks	This attribute is only valid in conjunction with the attribute extkey.

The following examples of request and response with `extkey_as_attr` on field level.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <fields>
            <field tablename="Company" fieldname="Company"/>
            <field tablename="Company" extkey="true" extkey_as_attr="true"
fieldname="Country"/>
          </fields>
        </query>
      </request>

      <response>
        <query>
          <Company tableshort="FI" id="73014444233">
            <Company>Austrian Company</Company>
            <Country extkey="AUT">Österreich</Country>
          </Company>
        </query>
      </response>
```

extsystem

extsystem	
Syntax	[extsystem = string]
May occur in	< cond > < link > < merge > any valid Command element any valid Table or field element
Description	The value of an external system.
Remarks	Used in conjunction with an external key.

fid

fid	
Syntax	[fid = int]
May occur in	< field > (or any other field reference)
Description	The field id or the core field id (see cid attribute in <metainfo/>), unique within its table.
Remarks	The field index is usually specified in the fnr attribute.

fieldname

fieldname	
Syntax	[fieldname = string]
May occur in	< field > (or any other field reference)
Description	The XML name of a field, unique within its table.
Remarks	The language-dependent name is usually specified in the name attribute.

fields

fields	
Syntax	[fields = string]
May occur in	< field > (or any other field reference)
Description	A comma-separated list of fields.
Remarks	Wildcards and regular expressions can be used.

fnr

fnr	
Syntax	[field = int]
May occur in	< field > (or any other field reference)
Description	The field index, unique within its table.
Remarks	The field id is usually specified in the fid attribute.

if

if	
Syntax	[if = string]
May occur in	any Command element
Description	An XPath specifying a condition that is evaluated against the current MP-document. If successful, the element is processed, otherwise it is ignored.
Remarks	See Message Processing.

mnr

mnr	
Syntax	[mnr = unsignedShort]
May occur in	(any field)
Description	The tenant number.

name

name	
Syntax	[name = string]
May occur in	<cond> (used in various places in the response document)
Description	The name of a table, field, catalog, or other entity.
Remarks	Commonly used to reference a language-dependent entity in the data model.

optional

optional	
Syntax	[optional = Boolean : false]
May occur in	<link> (any import field)
Description	An indicator of whether a processing error of this element causes an error. If true, an error is ignored and the respective element is treated as if it is absent.
Remarks	Commonly used for optional links.

recId

recId	
Syntax	[recId = unsignedLongHex]
May occur in	<table> (any record)
Description	A 64-bit record id specified in hexadecimal notation and prefixed with an 'x' character.
Remarks	Usually represented in full length with left zero padding.

table

table	
Syntax	[table = string]
May occur in	< table > or any other table reference
Description	The table id. When used in an XML request, interface also tries to look up the table by its name when it is not found via its id.
Remarks	See also the tablename attribute.

tablename

tablename	
Syntax	[tablename = string]
May occur in	< table > or any other table reference
Description	The XML name of a table. When used in an XML request, interface also tries to look up the table by its id when it is not found via its name.
Remarks	See also the table attribute.

value

value	
Syntax	[value = string]
May occur in	<cond> (used in various places in the response document)
Description	The value of a condition or field.
Remarks	none

Command attributes

List of attributes that are parsed on every command element, although most of them affect the processing of only a subset of commands.

Below is the list of attributes:

__iflags

__iflags	
Syntax	[__iflags = mmFlags]
May occur in	(any Command element)
Description	Internal flags that control processing.
Remarks	Do not use (internal use only).

catbynum

catbynum	
Syntax	[catbynum = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether catalog values are specified by a numeric index (as opposed to its textual value). This affects both request (records specified in <import> commands for example) and response (records returned from <query> commands for example).
Remarks	The catbynum and catexkeys indicators cannot both be set

catexkeys

catexkeys	
Syntax	[catexkeys = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether catalog values are specified by external key (as opposed to its textual value). This affects both request (records specified in <import> commands for example) and response (records returned from <query> commands for example).
Remarks	The catbynum and catexkeys indicators cannot both be set

catformat

catformat	
Syntax	[catformat = string]
May occur in	(any valid Command element)
Description	Indicates a format how catalog values are specified in requests, if the values are a combination of external key and text. Values: 1 = [EXTKEY] Text 2 = Text [EXTKEY] 3 = (EXTKEY) Text 4 = Text (EXTKEY) 5 = EXTKEY-Text 6 = EXTKEY|Text 7 = EXTKEY - Text 8 = EXTKEY | Text 9 = EXTKEY Text
Remarks	This indicator takes precedence over the catbynum and catexkeys attributes.

dateformatin

dateformatin	
Syntax	[dateformatin = string]
May occur in	(any valid Command element)
Description	The order of the date components if not unique. Possible values are "dmy", "mdy", "ymd", "ydm".
Remarks	none

dateformatout

dateformatout	
Syntax	[dateformatout = string]
May occur in	(any valid Command element)
Description	The formatting for date values in the response.
Remarks	See Formatting Date and Time Values.

fixcatint

fixcatint	
Syntax	[fixcatint = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether values of static catalogs are specified as numeric index (as opposed to textual value).
Remarks	none

flags

flags	
Syntax	[flags = mmFlags]
May occur in	(any valid Command element)
Description	Flags that control processing.
Remarks	See List of flags that control processing on page 159.

flags2

flags2	
Syntax	[flags2 = mmFlags]
May occur in	(any valid Command element)
Description	Flags that control processing.
Remarks	See List of flags that control processing on page 159.

force_internal_matchup

force_internal_matchup	
Syntax	[force_internal_matchup = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether the Aurea CRM internal logic is to be used for a matchup (as opposed to the external matchup configured in the configuration table).
Remarks	See Matchup on page 147.

internalfields

internalfields	
Syntax	[internalfields = string]
May occur in	(any valid Command element)
Description	An indicator of whether certain fields should be generated in the response. These include the primary key and last modified timestamp for records, and the update timestamps for fields.
Remarks	none

lazy_catfilter

lazy_catfilter	
Syntax	[lazy_catfilter = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether non-existing catalog values used in a query do not result in an error. If true, a condition that always evaluates to false is generated instead of the failing comparison.
Remarks	none

lazy_filter

lazy_filter	
Syntax	[lazy_filter = Boolean : false]
May occur in	(any valid Command element)
Description	Combines the lazy_catfiler and lazy_repfilter attributes.
Remarks	none

lazy_repfilter

lazy_repfilter	
Syntax	[lazy_repfilter = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether non-existing rep values used in a query do not result in an error. If true, a condition that always evaluates to false is generated instead of the failing comparison.
Remarks	none

matchup

matchup	
Syntax	[matchup = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether the matchup is to be done for records in <import> commands.
Remarks	The force_internal_matchup and use_configured_matchup attributes are only considered when matchup is true. See Matchup on page 147.

nodefaults

nodefaults	
Syntax	[nodefaults = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether default values should be set when creating new records.
Remarks	This flag is primarily for testing purposes – the setting "nodefaults" can cause the record not to be created due to missing default values or violate your business rules.

noerror

noerror	
Syntax	[noerror = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether field-check errors during the processing of < import > commands are to be ignored (and processing continues).
Remarks	For example, if you have no permission to update a field, this field is left unchanged (and the record is updated with all fields you have permission for) instead of an error generated and no update taking place.

nomustcheck

nomustcheck	
Syntax	[nomustcheck = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether must-fields are to be checked.
Remarks	This flag is primarily for testing purposes – setting "nomustcheck" can violate your business rules due to missing mandatory fields.

notrigger

notrigger	
Syntax	[notrigger = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether triggers are run when processing write operations on records.
Remarks	This flag is primarily for testing purposes – setting "notrigger" can violate your business rules.

readback

readback	
Syntax	[readback = string]
May occur in	(any valid Command element)
Description	Return record data instead of success/failure information for write operations on records.
Remarks	See Returning record data.

return

return	
Syntax	[return = string]
May occur in	(any valid Command element)
Description	Deprecated, only retained for backwards compatibility.
Remarks	Use readback='true' instead of return='2', and readback='false' instead of return='0'.

timeformatout

timeformatout	
Syntax	[timeformatout = string]
May occur in	(any valid Command element)
Description	The formatting for time values in the response.
Remarks	See Formatting Date and Time Values.

threads

type	
Syntax	[threads = unsignedByte]
May occur in	(any valid Command element)
Description	The number of concurrent threads to be used when importing records.
Remarks	See Working with "Threads" on page 148.

transaction

transaction	
Syntax	[transaction = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether a transaction context is established or not. Requests within transactions are rolled back if one of the requests fails.
Remarks	Although it is possible to set the transaction on every command element it doesn't affect the behavior in all cases since this attribute is ignored by the data base layer. E.g. during pure read only operations there is no transaction context established at all. **Note:** if catnew = true, catalog values are generated anyway no matter if the requests of the transaction are executed successfully or rolled back. For further details and examples see Working with transactions below.

truncate

truncate	
Syntax	[truncate = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether field values for text fields in < import > commands should be truncated if they are too long.
Remarks	none

updtimestamps

updtimestamps	
Syntax	[updtimestamps = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether the update timestamps for fields should be generated in <query> results.
Remarks	These timestamps (and more) are also generated when setting the internalfields attribute.

use_configured_matchup

use_configured_matchup	
Syntax	[use_configured_matchup = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether the matchup configured in the configuration table is to used.
Remarks	See Matchup.

xsdt

xsdt	
Syntax	[xsdt = Boolean : false]
May occur in	(any valid Command element)
Description	An indicator of whether XML Schema data types are to be used in the response. Affects the following field types: numbers, dates and times.
Remarks	The use of XML Schema data types is highly recommended for data exchange.

<cond> attributes

List of attributes for <cond> element.

extkey2

extkey2	
Syntax	[extkey2 = Boolean : false]
May occur in	<cond> (any valid field)
Description	An indicator of whether the parent catalog is specified as external key (as opposed to textual value).
Remarks	See also the mnr2 attribute.

In this example all interest records are returned where the external key of InterestGroup="ext_parent" AND the external key of Interest="ext_child". Both catalogs are specified as external keys, the child catalog due to attribute extkey="true", the parent catalog due to extkey2="true".

```
      <request>
        <query>
          <tables>
            <table tablename="Interests"/>
          </tables>
          <fields tablename="Interests" fields="InterestGrp,Interest"/>
          <condition>
            <cond tablename="Interests" fieldname="Interest" op="=" extkey="true"
value="ext_child" extkey2="true" value2="ext_parent"/>
          </condition>
        </query>
      </request>
```

mnr2

mnr2	
Syntax	[mnr2 = unsignedShort]
May occur in	<cond> (any valid field)
Description	The tenant number for a parent catalog.
Remarks	See also the extkey2 attribute.

op

op	
Syntax	[op = string]
May occur in	<cond>
Values	=, != (also <>), <, >, <=, >=, (),)(**Note:** These operators must be given in escaped form, e.g. "<" instead of "<" and "&lgt;" instead of ">".
Description	The comparison operator for a single condition.
Remarks	The () and)(operators mean "containing" and "not containing".

value2

value2	
Syntax	[value2 = string]
May occur in	<cond> (any valid field)
Description	The value of the parent catalog.
Remarks	none

lupd

lupd	
Syntax	[lupd = string]
May occur in	<cond>
Description	The value of the last-modified timestamp of a specified field.
Remarks	See query with conditions with condition on timestamp.

<dictionary> attributes

List of attributes for <dictionary> command.

complete

complete	
Syntax	[complete = Boolean : false]
May occur in	<dictionary>
Description	An indicator of whether all or only user-defined fields should be considered when generating the dictionary.
Remarks	none

<getcat> attributes

List of attributes for <getcat> command.

include_locked_values

include_locked_values	
Syntax	[include_locked_values = Boolean : false]
May occur in	<getcat>
Description	An indicator of whether locked catalogs should be included in the response.
Remarks	none

id

id	
Syntax	[id = unsignedInt]
May occur in	<getcat>
Description	An indicator of whether a catalog is request by its id
Remarks	none

<getdoc> attributes

List of attributes for <getdoc> command.

decipher

decipher	
Syntax	[decipher = Boolean : false]
May occur in	<getdoc>
Description	
Remarks	See Document encryption on page 159.

decrypt

decrypt	
Syntax	[decrypt = Boolean : false]
May occur in	<getdoc>
Description	
Remarks	See Document encryption on page 159n.

forceBase64

forceBase64	
Syntax	[forceBase64 = Boolean : false]
May occur in	<getdoc>
Description	An indicator of whether XML documents are generated as base64-encoded blob (instead of inline XML).
Remarks	Usually used when uniform processing of documents is desired.

verify

verify	
Syntax	[verify = Boolean : false]
May occur in	<getdoc>
Description	
Remarks	See Document encryption.

<import> attributes

List of attributes for <import> command.

allow_deleted

allow_deleted	
Syntax	[allow_deleted = Boolean : false]
May occur in	<import>
Description	An indicator of whether records marked as deleted (via the DelCd field) can be updated.
Remarks	This flag must be set when a record marked as deleted should be undeleted.

catnew

catnew	
Syntax	[catnew = Boolean : false]
May occur in	<import>
Description	An indicator of whether unknown catalog values should be automatically created.

Remarks	It is recommended that new catalog values is not created using this flag due to risk of unwanted duplicates. It works only in the catalog base language.
	It is also possible to create new catalog entries based on an external key.
	`<request>`
	`  <import catnew="1">`
	`  <fields>`
	`  <Company>`
	`  <Company>update Test AG</Company>`
	`  <FreeK1>Australia</FreeK1>`
	`  <FreeK2 extkey="1">AT</FreeK2>`
	`  </Company>`
	`  </fields>`
	`  </import>`
	`</request>`
	In the sample above for FreeK2 a matchup is performed via ExtKey. If the value AT is not found the catalog is created– contrary to versions before 7.0.6.203 - (in this sample with Text = AT and ExtKey = AT) because of catnew="1". If FreeK1 doesn't exist yet (matchup is performed via Text) a catalog value with Text = "Australia" is created.
	No additional properties (with the exception of ExtKey) can be specified when creating catalog values this way.

force_update

force_update	
Syntax	[force_update = Boolean : false]
May occur in	<import>
Description	An indicator of whether the record contents in <import> commands should be checked against the current values in the database. If true, no check is performed.
Remarks	none

mode

mode	
Syntax	[mode = string]
May occur in	<import>
Values	normal, plausi, noexec, trigger
Description	A mode indicator for write operations.
Remarks	normal = Regular insert/update is performed. plausi = Plausibility check is performed.

no_insert

no_insert	
Syntax	[no_insert = Boolean : false]
May occur in	<import>
Description	An indicator of whether the creation of a new record is allowed. If true, only an update is allowed (after matchup has been performed).
Remarks	none

write_cursor_flags

write_cursor_flags	
Syntax	[write_cursor_flags = mmFlags]
May occur in	<import>
Description	Flags that control processing of the <import> command.
Remarks	Currently not used

allow_locked_catalogs

allow_locked_catalogs	
Syntax	[allow_locked_catalogs = Boolean : false]
May occur in	<import>
Description	An indicator of whether the import of locked catalog values is allowed as well. If true, locked catalog values are imported instead of returning an error.
Remarks	none

In case the catalog value "Locked Country" is locked an error is generated, see example below.

```
      <request>
        <import allow_locked_catalogs="false">
          <fields>
            <Company>
              <Company>My Company</Company>
              <Country>Locked Country</Country>
            </Company>
          </fields>
        </import>
      </request>
      <response>
        <import>
          <return type="error" func="C_Portal::CheckField">
            <ecode>-10017</ecode>
            <etext>Catalog entry not found</etext>
            <code>0</code>
            <field table="FI" tablename="Company" fid="5" fnr="5" type="K"
fieldname="Country"/>
            <value>Locked Country</value>
          </return>
        </import>
      </response>
```

In order to allow importing of records with locked catalog values you may set allow_locked_catalogs="true".

```
      <request>
        <import allow_locked_catalogs="true">
          <fields>
            <Company>
              <Company>My Company</Company>
              <Country>LockedCountry</Country>
            </Company>
          </fields>
        </import>
      </request>
      <response>
        <import allow_locked_catalogs="true">
        <return table="FI" tablename="Company" id="42953967942464" type="insert"/>

        </import>
      </response>
```

<insert> attributes

List of attributes for <cond> command .

See topic <import> attributes on page 107.

<link> attributes

List of attributes for <link> element.

optional

optional	
Syntax	[optional = Boolean : false]
May occur in	<link>
Description	An indicator of whether a link that cannot be successfully resolved results in an error.
Remarks	none

import

import	
Syntax	[import = Boolean : false]
May occur in	<link>
Description	none
Remarks	Currently not used.

matchup

matchup	
Syntax	[matchup = Boolean : false]
May occur in	<link>
Description	none
Remarks	Currently not used.

<metainfo> attributes

List of attributes for <metainfo> command.

flags

flags	
Syntax	[flags = mmFlags]
May occur in	<metainfo>
Description	The "include" flags (as optionally specified in the include attribute) in hexadecimal form.
Remarks	See the include attribute.

include

include	
Syntax	[include = string]
May occur in	<metainfo>
Values	Indexes= 0x00000001, Rels= 0x00000002, Links= 0x00000004, ReverseLinks= 0x00000008, AttributeList= 0x00000010, FormatList= 0x00000020, CatalogNames= 0x00000040, Relations= 0x00000080, AllTableFlags= 0x0000003f,all of the above BaseFields= 0x00001000, CoreFields= 0x00002000, UniqueFields= 0x00004000, VirtualFields= 0x00008000, GdmUpdateFields= 0x00010000, GdmCustomerFields= 0x00020000, GdmPartnerFields= 0x00040000, SqlCoreFields= 0x00080000, SqlVarFields= 0x00100000, CustomFields= 0x00200000, GeneratedFields= 0x00400000, GdmFields=0x00070000,GdmUpdateFields+GdmCustomerFields + GdmPartnerFields AllFieldFlags= 0x007ff000,all of the above CoreTexts= 0x01000000,language-dll DefaultTexts= 0x02000000,built-in dictionary CustomTexts= 0x04000000,custom dictionary MissingTexts= 0x08000000, AllTextFlags= 0x0f000000, all of the above AllXmlFlags= 0xffffffff, all possible flags

include	
Description	A comma-separated list of flags that indicate what is to be returned in the response.
Remarks	For further information see Field types in appendix.

mode

mode	
Syntax	[mode = string]
May occur in	<metainfo>
Values	Flat, List, Attribute, AttributeXml, ElementXml
Description	The format of the output.
Remarks	none

<merge> attributes

List of attributes for <merge> command.

flag_ignore

flag_ignore	
Syntax	[flag_ignore = Boolean : false]
May occur in	<merge>
Description	This flag is currently not used.
Remarks	none

mode

mode	
Syntax	[mode = string]
May occur in	<merge>
Values	source, destination, timestamps, fields
Description	The mode of how the merge operation is to be performed.
Remarks	source = The source record always wins. destination = The destination record always wins. timestamps = The record with a more recent update timestamp wins. fields = The records are merged on a field-by-field basis, fields with a more recent update timestamp win.

no_exec

no_exec	
Syntax	[no_exec = Boolean : false]
May occur in	<merge>
Description	An indicator of whether the merge is to be executed. If true, only the conflicts (if any) are returned, but the actual merge is not performed.
Remarks	none

verbose

verbose	
Syntax	[verbose = Boolean : false]
May occur in	<merge>
Description	An indicator of whether to return the conflicts (if any).
Remarks	none

<putdoc> attributes

List of attributes for <putdoc> commands.

encrypt

encrypt	
Syntax	[encrypt = Boolean : false]
May occur in	<putdoc>
Description	An indicator of whether the document contents are to be encrypted.
Remarks	See Document encryption on page 159.

sign

sign	
Syntax	[sign = Boolean : false]
May occur in	<putdoc>
Description	An indicator of whether the document contents are to be digitally signed.
Remarks	See Document encryption on page 159.

<query> attributes

List of attributes for <query> command.

back

back	
Syntax	[back = Boolean : false]
May occur in	<query>
Description	back=true delivers the "previous page" of records (in other words: navigates one page [determined by chunk_size] back from the current offset).
Remarks	Attribute is solely valid, if chunked = true and navigable = true. See Chunked read for further details.

chunked

chunked	
Syntax	[chunked = Boolean : false]
May occur in	<query>
Description	An indicator of whether data should be read in chunks.
Remarks	See Chunked read for further details.

chunk_size

chunk_size	
Syntax	[chunk_size = unsignedInt]
May occur in	<query>
Description	Defines the maximum number of records to be read in one chunk.
Remarks	Attribute is solely valid if chunked = true. See Chunked read for further details.

clear

clear	
Syntax	[clear = Boolean : false]
May occur in	<query>
Description	An indicator whether the chunked query context should be deleted. <query qid="myQuery" clear="true"/> deletes the context of myQuery and subsequent calls returns a "Chunked read error". <query qid="myQuery"> <returntype="error"func="C_Portal::XmlExecuteQueryChunked"> <ecode>-10090</ecode> <etext>Chunked read error</etext> <description>Query not found</description> </return> </query>
Remarks	Attribute is solely valid if chunked = true. See Chunked read for further details.

hardcache

type	
Syntax	[hardcache = Boolean : false]
May occur in	<query>
Description	An indicator whether records (including fields) are cached once read. Might reduce the number of reads on the data base, but uses more memory. Changes on records after reading them into the cache are not reflected.
Remarks	Attribute is solely valid if chunked = true and navigable = true. See Chunked read for further details.

labels

labels	
Syntax	[labels = Boolean : false]
May occur in	<query>
Description	An indicator of whether the language-dependent field names should be generated in the response.
Remarks	None

lenient_filter

lenient_filter	
Syntax	[lenient_filter = Boolean : false]
May occur in	<query>
Description	If this attribute is set, conditions which cause a parsing error (e.g. due to a conversion error because an integer value is compared with an alphanumeric value) are ignored and solely the remaining conditions are evaluated.
Remarks	If all conditions are ignored no data a returned. This attribute is intended to support "generic searches", where conditions on fields are generated dynamically. In most scenarios it makes more sense to overcome conversion errors in conditions by changing the conditions rather than using this attribute. For an example see reading with lenient filters in the section <query>.

maxrecords

maxrecords	
Syntax	[maxrecords = unsignedInt]
May occur in	<query>
Description	The maximum number of records to be read in a query.
Remarks	The default value is 9999. This can be changed by setting the corresponding value in the registry.

navigable

navigable	
Syntax	[navigable = Boolean : false]
May occur in	<query>
Description	An indicator whether the record IDs of a "chunked read" are cached allowing for random access.
Remarks	Attribute is solely valid if chunked = true. See Chunked read for further details.

offset

offset	
Syntax	[offset = unsignedInt]
May occur in	<query>
Description	The absolute offset (starting point) for the next "chunked read".
Remarks	Attribute is solely valid if chunked = true. See Chunked read for further details.

page

page	
Syntax	[page = unsignedInt]
May occur in	<query>
Description	The relative offset (page * chunk_size) for the next "chunked read".
Remarks	Attribute is solely valid if chunked = true. See Chunked read for further details.

qid

qid	
Syntax	[qid = String]
May occur in	<query>
Description	A unique ID for a "chunked read" operation.
Remarks	Attribute is solely valid if chunked = true. The qid (query id) can be used to read subsequent chunks of data. See Chunked read for further details.

reverse

reverse	
Syntax	[reverse = Boolean : false]
May occur in	<query>
Description	An indicator of whether the records are read in descending order.
Remarks	This sets the ordering for all tables used in the query.

select

select	
Syntax	[select = string]
May occur in	<query>
Description	An XPath filtering the query response. Only matching records are generated in the response.
Remarks	none

skiprecords

skiprecords	
Syntax	[skiprecords = unsignedInt]
May occur in	<query>
Description	The number of records to be skipped from the beginning.
Remarks	Can be used in conjunction with the maxrecords attribute to implement page-wise reading.

<refresh> attributes

List of attributes for <refresh> command.

catalogs

catalogs	
Syntax	[catalogs = Boolean : false]
May occur in	<refresh>
Description	An indicator of whether the catalog cache is to be refreshed.
Remarks	none

configuration

configuration	
Syntax	[configuration = Boolean : false]
May occur in	<refresh>
Description	An indicator of whether the configuration cache (i.e. the contents of the configuration table MC) are to be refreshed.
Remarks	none

formats

formats	
Syntax	[format = Boolean : false]
May occur in	<refresh>
Description	An indicator of whether the format cache (i.e. the contents of the format table FT) are to be refreshed.
Remarks	none

protocols

protocols	
Syntax	[protocols = Boolean : false]
May occur in	<refresh>
Description	An indicator of whether the write-protocol cache is flushed to the database.
Remarks	none

reps

reps	
Syntax	[reps = Boolean : false]
May occur in	<refresh>
Description	An indicator of whether the rep cache is to be refreshed.
Remarks	none

<row_export> and <row_import> attributes

List of attributes for <row_export> and <row_import> command.

Note: Not all attributes are supported by both of the commands. See details in attribute description below.

boundary_separator

boundary_separator	
Syntax	[boundary_separator = string]
May occur in	<row_import>
Description	Overwrites the boundary separator of the import format.
Remarks	none

debug

debug	
Syntax	[debug = Boolean : false]
May occur in	<row_export> <row_import>
Description	Activates the logging of import/export-related traces, these are written using log level "info".
Remarks	none

field_separator

field_separator	
Syntax	[field_separator = string]
May occur in	<row_import>
Description	Overwrites the field separator of the import format.
Remarks	none

format_name

format_name	
Syntax	[format_name = string]
May occur in	<row_export> <row_import>
Description	The name of the import/export format to be used.
Remarks	none

input_file

input_file	
Syntax	[input_file = string]
May occur in	<row_import>
Description	Overwrites the input file from the import format. If import data is specified as XML and an input file is specified in the format, it must be set to an empty string.
Remarks	none

output_file

output_file	
Syntax	[output_file = string]
May occur in	<row_export>
Description	Overwrites the output file of the export format.
Remarks	none

record_separator

record_separator	
Syntax	[record_separator = string]
May occur in	<row_import>
Description	Overwrites the record separator of the import format. If import data is specified as rows/row, it must be set to the linefeed character "
".
Remarks	

<sort> attributes

List of attributes for <sort> command.

reverse

reverse	
Syntax	[reverse = Boolean : false]
May occur in	<sort>
Description	An indicator of whether sorting is to be performed in descending order.
Remarks	In contrast to the reverse attribute specified on the <query> command, the <sort> element is used to specify sort order per field per table.

<sleep> attributes

List of attributes for <sleep> command.

msec

msec	
Syntax	[msec = integer]
May occur in	<sleep>
Description	Number of milliseconds to sleep.
Remarks	none

<status> attributes

List of attributes for <status> command.

datamodel

datamodel	
Syntax	[datamodel = Boolean : false]
May occur in	<status>
Description	An indicator of whether the complete datamodel (as generated by CRMwin using the -writexmldef parameter) is included.
Remarks	This is retained for backwards compatibility only. Use the <metainfo> command instead.

profiling_snapshot

profiling_snapshot	
Syntax	[profiling_snapshot = Boolean : false]
May occur in	<status>
Description	An indicator of whether a profiling snapshot is included.
Remarks	This includes call count and timings of selected functions.

<table> attributes

List of attributes for <table> command.

alias

alias	
Syntax	[alias = string]
May occur in	<table>
Description	An optional alias to be used to reference the table in the request, and to be used instead of the tablename in the response. In the request both the tablename and alias attributes must be used whenever referencing this table since only the combination of the attributes tablename and alias is unique in the request.
Remarks	If a table is read multiple times, all but one must have an alias defined (to distinguish references to them in the request).

allfields

allfields	
Syntax	[allfields = Boolean : false]
May occur in	<table>
Description	An indicator of whether all fields should be read and returned in the response.
Remarks	none

flags

flags	
Syntax	[flags = mmFlags]
May occur in	<table>
Description	Flags influencing the read operation.
Remarks	See Cursor Flags on page 160.

flags2

flags2	
Syntax	[flags2 = mmFlags]
May occur in	<table>
Description	Flags influencing the read operation.
Remarks	See Cursor Flags on page 160.

index

index	
Syntax	[index = unsignedByte]
May occur in	<table>
Description	The index to be used when reading this table.
Remarks	Available indexes can be queried using the <metainfo> command (or looked up in the CRMwin service module).

keys

keys	
Syntax	[keys = string]
May occur in	<table>
Description	An optional key value used in conjunction with the index, setlen and keysend attributes.
Remarks	Deprecated, use a condition instead.

keysend

keysend	
Syntax	[keysend = string]
May occur in	<table>
Description	An optional key value for range keys, used in conjunction with the index, setlen and keys attributes.
Remarks	Deprecated, use a condition instead.

linkId

linkId	
Syntax	[linkId = short]
May occur in	<table>
Description	The id that specifies the relation between two tables.
Remarks	See samples using linkId in this document.

maxrecords

maxrecords	
Syntax	[maxrecords =] [maxrecords = unsignedInt]
May occur in	<table>
Description	The maximum number of records to be read for this table.
Remarks	none

readtype

readtype	
Syntax	[readtype = string]
May occur in	<table>
Description	A mode how to read the table.
Remarks	Internal use only.

relindex

relindex	
Syntax	[relindex = short]
May occur in	<table>
Description	Deprecated, only retained for backwards compatibility. Use the linkId attribute instead.
Remarks	none

setlen

type	
Syntax	[setlen = unsignedByte]
May occur in	<table>
Description	An optional length for keys, used in conjunction with the index, keys and keysend attributes.
Remarks	Deprecated, use a condition instead.

<transaction> attributes

List of attributes for <transaction> command.

cmd

cmd	
Syntax	[cmd = string]
May occur in	<transaction>
Values	begin, end, commit, rollback
Description	A value that controls the transaction command.
Remarks	none

Other attributes

List of miscellaneous attributes.

context

context	
Syntax	[context = integer]
May occur in	< link > <mp:value-of>
Description	An optional index that specifies the context node (if not unique).
Remarks	none

id

id	
Syntax	[id = unsignedLong]
May occur in	<table> Any record
Description	A 64-bit record id.
Remarks	Deprecated, only retained for backwards compatibility. See the recId attribute and "Recommended settings".

no_check

no_check	
Syntax	[no_check = Boolean : false]
May occur in	<cond>
Description	An indicator of whether the contents of the value attribute are validated.
Remarks	None

In case the catalog value "my test" doesn't exist or is locked an error is generated, see example below.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="FreeK1" op="=" value="my test"/>

          </condition>
          <fields>
            <field tablename="Company" fieldname="Company"/>
          </fields>
        </query>
      </request>
      <response>
        <query>
          <return type="error" func="C_Portal::CheckField">
            <ecode>-10017</ecode>
            <etext>Catalog entry not found</etext>
            <code>0</code>
            <field table="FI" tablename="Company" fid="57" fnr="57" type="K"
fieldname="FreeK1"/>
            <value>my test</value>
          </return>
        </query>
      </response>
```

In order to override this behavior you may set no_check="true" – then the check for existence of the catalog value is omitted.

```
      <request>
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond no_check="true" tablename="Company" fieldname="FreeK1" op="="
value="my test"/>
          </condition>
          <fields>
            <field tablename="Company" fieldname="Company"/>
          </fields>
        </query>
      </request>
      <response>
        <query>
          <Company tableshort="FI" id="429496732461">
            <Company>Test Company 88</Company>
          </Company>
        </query>
      </response>
```

target

target	
Syntax	[target = string]
May occur in	(any field)
Description	Specifies that this field is the target for a command-specific operation.
Remarks	Currently only used in the <putdoc> command, to specify this field as write target for the document link (to be used as document field or "link" in CRMwin etc.).

<mp> attributes

List of attributes for <mp> command.

output

output	
Syntax	[output = string]
May occur in	<node-set>
Values	text, xml
Description	Specifies how the result of the respective operation is to be encoded.
Remarks	See Message Processing.

test

test	
Syntax	[test = string]
May occur in	<if>
Values	text, xml
Description	An XPath specifying a condition.
Remarks	See Message Processing.

resolve

resolve	
Syntax	[resolve = Boolean : false]
May occur in	<condition>, <cond>
Description	An indicator of whether the contents of the value attribute are resolved or not interpreted.
Remarks	See Message Processing.

<xquery> attributes

List of attributes for <xquery> command.

name

name	
Syntax	[name = string]
May occur in	<xquery>
Description	The name of the format.
Remarks	none

type

type	
Syntax	[type = string]
May occur in	<xquery>
Description	The type of the format.
Remarks	The type can be specified by format type code (32 or 34, optionally prefixed by a hash '#' sign), or by name (Query or Read Engine).

Custom attributes

Custom attributes are attributes which are not part of the CRM.interface XML language.

Such attributes are copied verbatim to the response. This might be useful to transport custom information over requests and responses, e.g. you could think of using this feature to implement a kind of session handling.

Custom attributes are supported on <request> element, each command element (e.g. <update>) and on <field> and <fields> child elements of the <query> command.

Additional custom attributes on "Info Area" level for <match>, <insert>, <import>, <update> ... are supported as well.

See the sample below, where MyCompanyAttribute="MyCompanyAttributeValue" is set on the <Company> tag of the <import /> command and returned in the response XML.

Note: Some attributes are "internal" or "protected" respectively. Internal attributes are not copied; the usage of protected attributes causes an error.

Note: Internal attributes: tablename, table, tid, fieldname, field, fields, fid, fnr and name.

Note: Protected attributes: text, code, value, label, upd, null and locked.

```
<?xml version="1.0"?>
<request MyAttribute="MyAttributeValue">
  <import MyImportAttribute="MyImportAttributeValue">
    <fields>
      <Company MyCompanyAttribute="MyCompanyAttributeValue">
        <Company matchup="true"> CRM.interface test</Company>
        <Synonym/>
```

```
            </Company>
          </fields>
        </import>
        <query catexkeys="false" MyQueryAttribute="MyQueryAttributeValue">
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="
CRM.interface test"/>
          </condition>
          <fields tablename="Company" fields="Company,FreeN3"
MyFieldsAttribute="MyFieldsAttributeValue"/>
          <fields>
            <field tablename="Company" fieldname="FreeC1"
MyFieldAttribute="MyFieldsAttributeValue"/>
          </fields>
        </query>
      </request>
      <?xml version="1.0"?>
      <response MyAttribute="MyAttributeValue">
        <import MyImportAttribute="MyImportAttributeValue">
         <return table="FI" tablename="Company" id="429496732500" type="update"/>

        </import>
        <query catexkeys="false" MyQueryAttribute="MyQueryAttributeValue">
          <Company MyCompanyAttribute="MyCompanyAttributeValue"  tableshort="FI"
id="429496732500">
            <Company MyFieldsAttribute="MyFieldsAttributeValue"> CRM.interface
test</Company>
            <FreeC1 MyFieldAttribute="MyFieldsAttributeValue"/>
            <FreeN3 MyFieldsAttribute="MyFieldsAttributeValue"/>
          </Company>
        </query>
      </response>
```

Boolean attributes

Attributes of type Boolean should be specified as true or false.

They can be specified as 0/1 or on/off for backwards compatibility (however, this usage is deprecated and not recommended). If XML data types are used, then usage of true/false is mandatory.

Field Attributes

Learn about the field attributes.

The field attributes used in the meta information.

Attribute	Meaning
InputHook	The field has an Aurea CRM input hook.
OutputHook	The field has an Aurea CRM output hook.
SearchHook	The field has an Aurea CRM search condition hook.
Invisible	The field is internal.

Attribute	Meaning
Readonly	The field is read-only.
NoListSum	The field cannot be used as a summed field.
DecodeFI	The field needs its corresponding company record for decoding.
DecodeKP	The field needs its corresponding person record for decoding.
AutoLoad	The field has a computed default value.
Currency	The field is a currency field.
DefaultList	The field is a. Aurea CRM list default field.
NoList	The field is no list field.
Private	The field is the special private boolean flag.
NoEdit	The field is read-only (even for the superuser).
Bool	The field is of type boolean.
NoComm	The field is not communicated (via the Aurea CRM communications module).
NoTimestamp	The field is not time-stamped when modified.
Document	The field contains a reference to a document (stored in tables D1 or D2).
Required	The field is a required field (and must be filled on creation).
LockSelection	The field is the special lock selection flag.
LockActivity	The field is the special lock activity flag.
Email	The field contains an email address.
Obsolete	The field is from a previous version and not used anymore.

Attribute	Meaning
Tenant	The field contains a tenant number.
Free	The field has no business logic attached (and never have) and can be freely used.
Rep	The field contains a rep.
Group	The field contains a rep group.
Resource	The field contains a resource.
Hyperlink	The field contains an internet address.
StatNo	The field contains the station number of the primary key.
SeqNo	The field contains the sequence number of the primary key.
HierarchyCode	The field contains a rep hierarchy code.

Catservice Attribute

Catservice attribute is used to import catalog values using CRM.Interface.

catnew	
Syntax	[catservice = Boolean : false]
May occur in	<import>
Description	An indicator of whether the import command should run in the catalog service (maintainence) mode.

If this attribute is set to "true" and the import concerns the KA table, then the "normal" import is intercepted by the catalog maintainence functionality.

Note: Text must be provided in the base language.

Note: Deletion of catalog values is not supported by the interface catalog service functionality, but it is possible using service module.

Example importing a new catalog value (catservice = true)

This example illustrates the import of a new catalog value in both the base language (100, German) and an additional language (200, English). The response contains the catalog maintainence Ids and the particular mode (insert, update).

```
<Request>
        <import catservice = "true" dateformatin = "ymd">
                <fields>
                        <Catalog>
                                <Index>61</Index>
                                <ExtKey>FI-FREE3-2</ExtKey>
                                <Text>FI free3 2</ExtKey>
                                <Lang>100</Lang>
                         </Catalog>
                         <Catalog>
                                <Index>61</Index>
                                <ExtKey>FI-FREE3-2</ExtKey>
                                <Text>FI free3 English</ExtKey>
                                <Lang>200</Lang>
                         </Catalog>
                 </fields>
</Request>
```

Miscellaneous topics

This topic has some useful miscellaneous information.

Shadow User

CRM.interface uses a technical user – by default the WWW user – to access Aurea CRM if no login context is provided.

Use `update.Users.exe` – which is installed in the installation of interface - to create or modify the `users.xml` file. Because `users.xml` contains the usernames and passwords of a CRM user, it is highly recommend to encrypt the contents of the file via the option "Use Xml Encryption".

Authentication

Authentication can be done in two ways: via user and password (in clear text) or via an encrypted token.

Naturally, clear text authentication is recommended to be used only in a secure environment (e.g. when the XML requests are transported over HTTPS).

Plain text authentication (login)

Specified using the user and pwd attributes. The pwd attribute is optional and can be omitted when the login does not require a password.

Plain text authentication (single sign-in)

Specified using the domain and user attributes. This is mostly done when using HTTP authorization, where the successful HTTP authorization is carried over into the XML request as single sign-in.

Encrypted authentication

Specified using the auth attribute. A suitable token can be generated programmatically using the `update.lib.dll` C# assembly using InterfaceAuthenticationTicket.CreateTicket7(String username, String password). The tokens are specific to the CRM.interface they are generated for (i.e. they cannot be shared between the different numbers of CRM.interface), and are intended for one-time only (they have a very short expiration time by default).

Impersonation

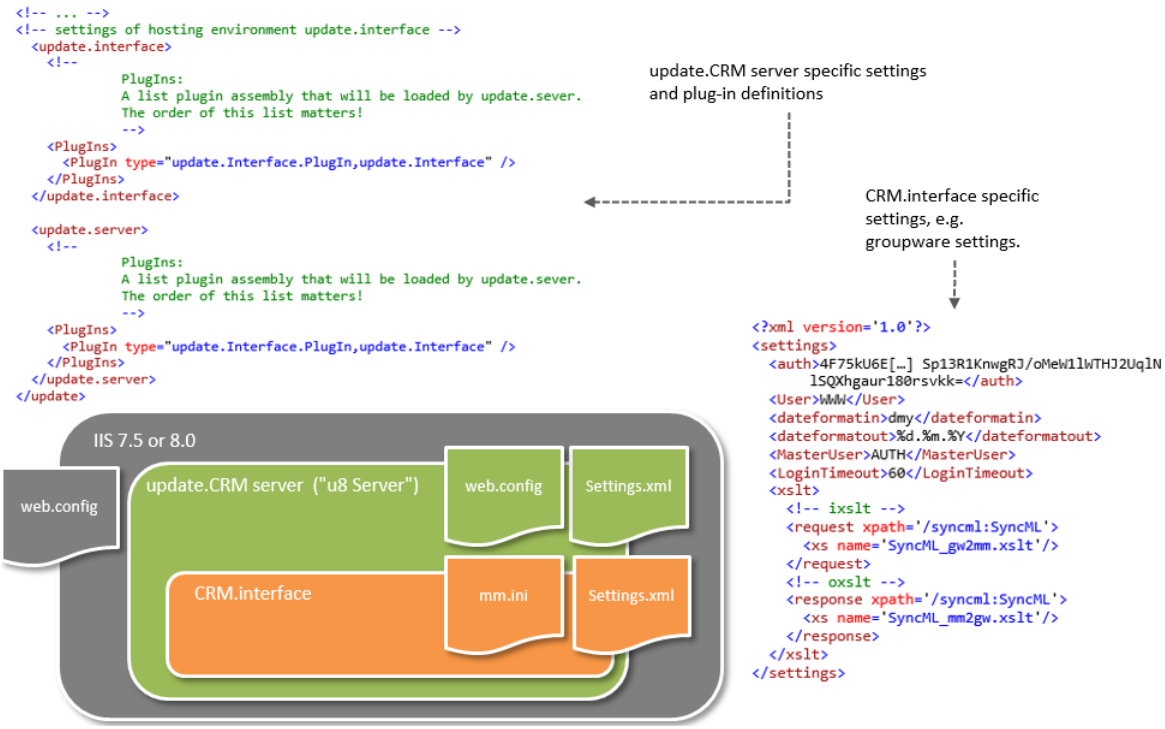
The user that has been granted impersonation rights has the same rights as the one whom it is impersonating.

Specified using the impersonate attribute.

Impersonation requires, that

- An impersonation user ("master user") is configured in CRM.interface (see figure below)
- the login matches the specified credentials
- and the impersonate attribute is specified

If all the above listed conditions are fulfilled a login without password check using the credentials given in the impersonate attribute is performed.



The impersonate attribute consists of at least one name/value pair (separated by semicolon), each referencing a login. The first successful login is then used for the respective request or command. The following list describes the possible values for the name part:

- **user:** the login name (as specified in the user attribute in Authentication)
- **rep:** the name of the associated rep (table RepUser/ID, field Name/3)
- **email:** the email address of the associated rep (table RepUser/ID, field E-mail1/27)

Example: (the value of the auth attribute is omitted for brevity)

```
<requestauth="…"impersonate="user=USERNAME;rep=RepUserName;email=email@mail.com">

  <!-- command -->
</request>
```

Referencing a list of fields

Learn how to reference a list of fields.

If the list starts with "regex:", the remainder is treated as a regular expression that is evaluated against all fields of the referenced table. Otherwise, it is treated as a comma-separated list of fieldnames, with two possible wildcards:

- "*": all fields of that table, excluding virtual fields
- "**": all fields of that table, including virtual fields

For a list of field types, see FieldTypes and Categories on page 165.

Example: Reading all "Free" fields of a table (fields which name start with Free)

```
<request>
  <query>
    <tables>
      <table tablename="Company"/>
    </tables>
    <fields tablename="Company" fields="regex:Free.+"/>
    <condition>
      <cond tablename="Company" fieldname="Company" op="=" value="Test"/>
    </condition>
  </query>
</request>
```

Formatting Date and Time Values

Comprehensive date and time formatting are supported in CRM.interface.

The supported formatting codes for dateformatout and timeformatout are listed in the table below.

Note:

`dateformatout`

solely supports date formatting codes and

`timeformatout`

solely supports time formatting codes. Means that for example you cannot use %a for formatting a time value via

`timeformatout`

Attributes	Meaning
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%j	Day of year as decimal number (001 – 366)
%m	Month as decimal number (01 – 12)
%M	Minute as decimal number (00 – 59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00 – 59)
%U	Week of year as decimal number, with Sunday as first day of week (00 – 53)
%w	Weekday as decimal number (0 – 6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00 – 53)
%x	Date representation for current locale

Attributes	Meaning
%X	Time representation for current locale
%y	Year without century, as decimal number (00 – 99)
%Y	Year with century, as decimal number
%z, %Z	Either the time-zone name or time zone abbreviation, depending on registry settings; no characters if time zone is unknown
%%	Percent sign

Examples:

%Y-%m-%d	1975-08-15
%d.%m.%Y	15.08.1975
%m/%d/%y	08/17/75

Note: The year has to be first or last, and the three components can optionally be separated by a single character (both separators must be the same). Day and month are always expressed in two digits.

Formatting Time Values

See Formatting Time Values.

Matchup

A record matchup occurs when processing <import> or <matchup> commands.

The `<import>` command uses the matchup logic to decide whether the record should be inserted or updated, whereas the `<matchup>`command returns possible matches for the given record data. The `<matchup>` command is only defined for the tables Company/FI and Person/KP.

custom (per field) matchup (only used with <matchup>)

Fields that have the attribute matchup='true' set are used as a filter to query the respective table. When no records are found, the matchup continues with internal/external matchup. When one record is found, the matchup stops, otherwise it is an error.

internal/external matchup

External matchup is defined as the C# assembly configured in the configuration table. Internal matchup is defined as the internal import matchup logic as documented in the business logic manual. If the external matchup is to be used with the `<import>` command, the use_configured_matchup='true' attribute has to be specified (as the `<matchup>`command is intended to be used for exactly this purpose, it is implied there and cannot be set/unset). The internal matchup can be forced with both commands by using the force_internal_matchup='true' attribute.

The internal/external matchup can be disabled by setting matchup='false' on the `<import>` command.

Note: CRM.interface is capable of three different matchup types.

- CRM.interface internal matchup, referred as internal matchup.
- Aurea CRM core matchup (which is also used in the import module)
- in the truest sense of the word external matchup

Working with "Threads"

For performance reasons, processing of bulk data can be split among a maximum of 16 worker threads.

If multiple `<fields>` blocks are used, they are assigned sequentially to the worker threads, with each worker thread that has finished processing his `<fields>` block obtaining the next until no blocks remain. If only one `<fields>` block is used, its records are split evenly amongst the threads, with the last thread also processing the remainder if not evenly divisible.

It is necessary to group dependent records together in blocks to ensure that parent records (or in general, linked records) are processed before their children (or linked-to records).

The following could occur in a 3rd party interface where records are always generated individually:

```
<import threads="5">
  <fields>
    <!-- ... -->
    <Company>
      <!-- ... -->
    </Company>
    <Person>
      <link>
        <!-- to preceding Company -->
      </link>
      <!-- ... -->
    </Person>
    <!-- ... -->
  </fields>
</import>
```

If this was the only `<fields>` block, the split between threads could occur between the two records (unless the request is carefully tuned), resulting in the possibility of the Person record to be processed before the Company record. Either more than one `<fields>` block has to be used to ensure proper ordering, or the records itself have to generated dependent of each other:

```
<import threads="5">
  <fields>
    <!-- ... -->
    <Company>
      <!-- ... -->
    </Company>
    <Person>
      <link>
        <!-- to preceding Company -->
      </link>
      <!-- ... -->
    </Person>
    <!-- ... -->
  </fields>
  <fields>
    <!-- ... -->
  </fields>
  <!-- ... -->
</import>
<import threads="5">
  <fields>
    <!-- ... -->
    <Company>
      <!-- ... -->
      <Person>
        <!-- implicit link to Company, no explicit <link> needed -->
        <!-- ... -->
      </Person>
    </Company>
    <!-- ... -->
  </fields>
</import>
```

Note: Side note: Theoretically, the very first example could also be imported twice, after which both records should have been imported successfully. However, this cannot be whole-heartedly recommended because of processing overhead, possible indeterminate record state due to triggers, other 3rd party interfaces etc.

Working with transactions

With a transaction you can ensure that either all database modifications of a command are executed or rolled back in case of an error (so that the command has no database modifying affect whatsoever).

It is possible to put requests into a transaction context.

```
<request>
  <import transaction="true">
    <fields>
      <Company>
        <Company>update software AG</Company>
        <Person>
            <Wrong>wrong fieldname</ Wrong >
            <FirstName>John</FirstName>
            <LastName>Doe</LastName>
          </Person>
```

```
          </Company>
        </fields>
      </import>
    </request>
```

In this sample the company record is NOT inserted due to an error occurring in the following person record (“wrong fieldname”). In order to ensure a rollback of the transaction, the attribute transaction has to be set on the `<import>` element.

The response returns the error and information about the transaction rollback:

```
    <response>
      <import transaction="true">
        <return table="FI" tablename="Company" id="4294981504" type="insert"/>
       <return table="KP" id="0" type="error" func="C_Portal::ProcessImportNode">

          <ecode>-10027</ecode>
          <etext>Dictionary: Field not found</etext>
          <table table="KP" tablename="Person"/>
          <field>Wrong</field>
        </return>
        <return type="error" func="C_Portal::XmlProcessCommand">
          <ecode>-10089</ecode>
          <etext>Database transaction: Rollback</etext>
        </return>
      </import>
    </response>
```

It is possible to put the whole request into a transaction context and/or to control the transaction individually.

If the transaction attribute is set on the request element the behavior is the same as described above for command elements. All commands are executed in one transaction.

```
    <request transaction="true">
      <putdoc>
        <Name>Testdatei.txt</Name>
        <Keyword>Putdoc_Test</Keyword>
        <rawdata xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="bin.base64">dGVzdA==</rawdata>
      </putdoc>
      <import>
        <fields>
          <Documents>
            <links>
              <link tablename="Documents" id="$lastRecId"/>
            </links>
            <Private>false</Private>
            <DocClass>Textfile</DocClass>
            <Owner>Own RepName</Owner>
            <FreeC1>Test</FreeC1>
            <!-- … -->
          </Documents>
        </fields>
      </import>
    </request>
```

In this example, if the import fails, the document is not created as well.

Transactions can be started and active transactions can be committed or rolled back explicitly using the `<transaction>` command. Transactions are started with `cmd="begin"`, and ended with `cmd="end"` (rollback on error, otherwise commit), `cmd="commit"` (explicit commit), or `cmd="rollback"` (explicit rollback).

```
    <request>
      <!-- ... -->
```

```
    <transaction cmd="begin"/>
    <import>
      <fields>
        <Company>
          <Company>update software AG</Company>
          <Person>
            <Wrong>
              wrong fieldname</ Wrong >
              <FirstName>John</FirstName>
              <LastName>Doe</LastName>
            </Person>
        </Company>
      </fields>
    </import>
    <!-- ... -->
    <transaction cmd="end"/><!—end|commit|rollback -->
    <!-- ... -->
  </request>
```

Message Processing

The mp (Message Processing) execution engine provides access to the current request and response (up to the previous command) using standard XPath queries in a pseudo document with top-level element <root>, henceforth called the mp document.

Whenever an expression (e.g. an XPath from a test or select attribute) is evaluated, it is done so against the mp document. When processing starts, the mp document provides access to the XML request. After processing a command, its response is available for all successive commands.

CRM.interface includes elements for flow control and expression evaluation that are loosely based on XSLT, but naturally are implemented separately. All mp commands reside in their own namespace.

<mp:choose>

Provides multiple condition testing in conjunction with the `<mp:when>` and `<mp:otherwise>` elements.

<mp:choose />	
Attributes	(no attributes)
Contents	<mp:when> + <mp:otherwise>?
May occur in	<request> (any table)
Remarks	none

<mp:for-each>

`<mp:for-each>` allows repeated execution of a command.

<mp:for-each>	
Attributes	select
Contents	(any command)
May occur in	<request>
Remarks	none

<mp:if>

Allows conditional XML fragments.

<mp:if />	
Attributes	test
Contents	(any command) (any field)
May occur in	<request> (any table)
Remarks	none

<mp:otherwise>

Provides multiple condition testing in conjunction with the `<mp:choose>` and `<mp:when>` elements.

<mp:otherwise />	
Attributes	(no attributes)
Contents	(any command) (any field)
May occur in	<mp:choose>
Remarks	none

<mp:value-of>

Provides for expression evaluation, variable resolution, and XPath selection.

<mp:value-of />	
Attributes	eval resolve select
Contents	(no content)
May occur in	(any field) (any mp content)
Remarks	none

<mp:when>

Provides multiple condition testing in conjunction with the `<mp:choose>`and `<mp:otherwise>` elements.

<mp:when />	
Attributes	test
Contents	(any command) (any field)
May occur in	<mp:when>
Remarks	none

Setting context

Learn how to set context.

Description is provided in an upcoming revision of this document.

Built-in variables

Built-in variables are used in places where a record id is needed.

The following variables can be used:

- **$firstRecId**: the id of the first record of that table that is processed in a query or import (regardless of occurrence)
- **$lastRecId**: the id of the last record of that table that is processed in a query or import (regardless of occurrence)
- **$prevRecId**: the id of the first record of that table that was processed in the previous query or import command

Example: using variables to update a record that was previously read.

```
      <request>
        <query xsdt="true">
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="update
software Ger*"/>
          </condition>
          <fields tablename="Company" fields="Company"/>
        </query>
        <query xsdt="true">
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
           <cond tablename="Company" fieldname="Company" op="=" value="update*"/>

          </condition>
          <fields tablename="Company" fields="Company"/>
        </query>
        <import>
          <fields>
            <Company recId="$XXX">
              <!-- fields to be updated -->
            </Company>
          </fields>
        </import>
      </request>
```

When $XXX are updated:

- $firstRecId: update software Germany Gmbh
- $lastRecId: update software (Switzerland) GmbH
- $prevRecId: update software AG

```
      <response>
        <query xsdt="true">
          <Company table="FI" id="296352743973" recId="x0000004500000225">
            <Company>update software Germany GmbH</Company>
          </Company>
        </query>
        <query xsdt="true">
          <Company table="FI" id="296352743952" recId="x0000004500000210">
            <Company>update software AG</Company>
          </Company>
          <Company table="FI" id="296352743973" recId="x0000004500000225">
            <Company>update software Germany GmbH</Company>
          </Company>
          <Company table="FI" id="296352743974" recId="x0000004500000226">
            <Company>update software (Switzerland) GmbH</Company>
          </Company>
        </query>
        <import>
          <!-- ... -->
        </import>
      </response>
```

Flow control - Conditional execution

The expression in the test attribute of the `<mp:if>` or `<mp:when>` element is evaluated and the result converted to a boolean value. If the result is true, the content is processed else it is left unprocessed.

When alternative(s) are needed, the `<mp:when>` elements describe one or more alternatives to be chosen by the `<mp:choose>` element, the (optional) default alternative is described by the `<mp:otherwise>` element. For simple conditional testing with no alternative(s), use the `<mp:if>` element.

Example: depending on the query result, different processing can be performed.

```
<request xmlns:mp="http://www.update.com/xml/core/mp">
  <query>
    <tables>
      <table tablename="Company"/>
    </tables>
    <condition>
     <cond tablename="Company" fieldname="Company" op="=" value="update*"/>

    </condition>
  </query>
  <mp:choose>
    <mp:when test="count(/root/response/query[1]/Company)>1">
      <!-- multiple choice -->
    </mp:when>
    <mp:when test="count(/root/response/query[1]/Company)=1">
      <!-- unique match  -->
    </mp:when>
    <mp:otherwise>
      <!-- not found -->
    </mp:otherwise>
  </mp:choose>
</request>
```

Flow control – Loops using <mp:for-each>

The select attribute of the `<mp:for-each>` command is evaluated and the contents processed with the result of the select passed as context.

This allows for more complex operations. For example, a "simple" update of the records could also be done using the <update> command) on multiple records in a single request.

Expression evaluation using <mp:value-of>

The `<mp:value-of>` element is used to reference into the mp document.

Three attributes control the behavior:

select

The expression is evaluated and the results are converted to a string (as by a call to the string() XSLT function). A node-set is converted to a string by inserting the string value of the first node in the set.

Example: copy field contents from one field to another (here, the contents of FreeC1 are copied to FreeC2).

```
      <request xmlns:mp="http://www.update.com/xml/core/mp">
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition>
            <cond tablename="Company" fieldname="Company" op="=" value="update
software AG"/>
          </condition>
          <fields tablename="Company" fields="Company,FreeC1"/>
        </query>
        <import>
          <fields>
            <Company recId="$lastRecId">
              <FreeC2>
                <mp:value-of select="/root/response/query[1]/Company/FreeC1"/>
              </FreeC2>
            </Company>
          </fields>
        </import>
      </request>
```

eval

Due to the specific implementation, expressions that do not select nodes (as per select above), have to be treated differently. An example is calling XSLT functions.

Example: Negating a Boolean value and incrementing a number (this assumes that the record already exists)

```
      <request xmlns:mp="http://www.update.com/xml/core/mp">
        <query xsdt="true">
          <tables>
            <table tablename="ItemMaster"/>
          </tables>
          <condition>
            <cond tablename="ItemMaster" fieldname="ItemNo" op="="
value="mp_value-of_eval"/>
          </condition>
          <fields tablename="ItemMaster" fields="ItemNo,Available,FreeN1"/>
        </query>
        <mp:for-each select="/root/response/query[1]/ItemMaster">
          <import>
            <fields>
              <ItemMaster recId="@recId">
                <Available>
                  <mp:value-of eval="not(Available='true')"/>
                </Available>
                <FreeN1>
                  <mp:value-of eval="number(FreeN1)+1"/>
                </FreeN1>
              </ItemMaster>
            </fields>
          </import>
        </mp:for-each>
        <query xsdt="true">
          <tables>
            <table tablename="ItemMaster"/>
          </tables>
          <condition>
            <cond tablename="ItemMaster" fieldname="ItemNo" op="="
value="mp_value-of_eval"/>
          </condition>
          <fields tablename="ItemMaster" fields="ItemNo,Available,FreeN1"/>
        </query>
      </request>
      <response xmlns:mp="http://www.update.com/xml/core/mp">
```

```
        <query xsdt="true">
          <ItemMaster table="AR" id="12" recId="x000000000000000c">
            <ItemNo>mp_value-of_eval</ItemNo>
            <Available>false</Available>
            <FreeN1>20</FreeN1>
          </ItemMaster>
        </query>
        <import>
          <return table="AR" tablename="ItemMaster" id="12"
recId="x000000000000000c" type="update"/>
        </import>
        <query xsdt="true">
          <ItemMaster table="AR" id="12" recId="x000000000000000c">
            <ItemNo>mp_value-of_eval</ItemNo>
            <Available>true</Available>
            <FreeN1>21</FreeN1>
          </ItemMaster>
        </query>
      </response>
```

Additionally you can call a script in <mp:value-of eval>.

Note: The script call must be prefixed by implements-prefix in which the script is implemented, e.g. `<mp:value-ofeval='js:add(number(/root/response/query[1]/Company/FreeN1),666)'/>`.

Note: the script must be located in the `<msxsl:script>` section of the style sheet (`<msxsl:script language='JScript' implements-prefix='js'></msxsl:script>`)

The example below illustrates how to call a script (js:add) in `<mp:value-of eval>` and shows a simple implementation of this script in the style sheet.

```
      <?xml version='1.0'?>
      <request xmlns:mp='http://www.update.com/xml/core/mp'
log='log/log_mp_script.xml'>
        <import>
          <fields>
            <Company>
              <Company matchup='true'>mp_script</Company>
              <FreeN1>69</FreeN1>
            </Company>
          </fields>
        </import>
        <query>
          <tables>
            <table tablename='Company'/>
          </tables>
          <fields tablename='Company'
fields='CoGrp,CoNo,Company,FreeN1,FreeN2,FreeN3,Text'/>
          <condition>
            <cond table='FI' fieldname='Company' op='=' value='mp_script'/>
          </condition>
        </query>
        <import>
          <fields>
            <Company>
              <Company matchup='true'>mp_script</Company>
              <FreeN2>
                <mp:value-of
eval='js:add(number(/root/response/query[1]/Company/FreeN1),666)'/>
              </FreeN2>
              <FreeN3>
                <mp:value-of
resolve='{eval:js:add(number(/root/response/query[1]/Company/FreeN1),666)}'/>
              </FreeN3>
              <Text>
```

```
                <mp:value-of
resolve='js={eval:js:add(number(/root/response/query[1]/Company/FreeN1),666)}'/>
              </Text>
            </Company>
          </fields>
        </import>
        <query>
          <tables>
            <table tablename='Company'/>
          </tables>
          <fields tablename='Company'
fields='CoGrp,CoNo,Company,FreeN1,FreeN2,FreeN3,Text'/>
          <condition>
            <cond table='FI' fieldname='Company' op='=' value='mp_script'/>
          </condition>
        </query>
      </request>

      <?xml version='1.0'?>
    <xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'

      xmlns:msxsl='urn:schemas-microsoft-com:xslt' xmlns:js='urn:js'
      exclude-result-prefixes='msxsl js'>
      <xsl:output method='text'/>
      <xsl:param name='node_name'>root</xsl:param>
      <xsl:template match='test'>
      <xsl:value-of select='js:add(number(.),1)'/>
      </xsl:template>
      <msxsl:script language='JScript' implements-prefix='js'>
      <![CDATA[
      function add(n1,n2)
      {
      return n1+n2;
      }
      ]]>
      </msxsl:script>
      </xsl:stylesheet>
```

resolve

The expression is parsed and scanned for parts that are enclosed in curly braces. Currently, these parts can start with a dollar sign (in which case it is a variable), or be an identifier followed by a colon and an expression (in which case it is a selector).

Built-in variables:

- $userId: the internal id of the current request
- $userName: the name of the current login user
- $repId: the id of the current login user/rep
- $repName: the name of the current login rep
- $groupId: the group id of the current login rep
- $groupName: the group name of the current login rep
- $date: the current date
- $time: the current time
- $random: a random number in the range from 0-32767

Built-in selectors:

- select: see evaluation of select above
- eval: see evaluation of eval above

- node-set: select a complete XML tree (expression is evaluated as in select, but does not select only the text content, but the whole XML tree)
- value-set: same as select

Example: usage in conditions – the value attribute is resolved as described above on all elements where resolve=”true” on itself or any parent condition element.

```
      <request xmlns:mp="http://www.update.com/xml/core/mp">
        <!-- read a company -->
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <!-- condition or link -->
          <fields tablename="Company" fields="Company,Synonym,Country"/>
        </query>
        <!-- read all companies with the same country and synonym -->
        <query>
          <tables>
            <table tablename="Company"/>
          </tables>
          <condition resolve="true">
            <lop value="and">
              <cond tablename="Company" fieldname="Country" op="="
value="{select:/root/response/query[1]/Company/Country}"/>
              <cond tablename="Company" fieldname="Synonym" op="="
value="{select:/root/response/query[1]/Company/Synonym}"/>
            </lop>
          </condition>
          <fields tablename="Company" fields="Company,Synonym,Country"/>
        </query>
      </request>
```

List of flags that control processing

Learn about the list of flags that control processing.

Full description is provided with an upcoming revision of this document.

Returning record data

When data is imported, it is sometimes desirable not to return the status (insert, update, match) but the record itself. This can be done manually with a `<query>` following the `<import>` or using the "readback" feature.

The "readback" attribute on the `<import>` element specifies a reference to the `fields.xml` in the xml/ folder. This file is not present by default. The syntax is the same as the <form> elements in the `forms.xml` used by the groupware Server part. The value of the "readback" attribute of an `<import>` element corresponds to the "type" attribute of the `<form>` element in the `fields.xml` file. The record is then returned with the fields specified under the respective `<table>` element, just like a normal `<query>`.

Document encryption

Learn about document encryption.

Full description is provided with an upcoming revision of this document.

Recommended settings

You need to perform the settings recommended in this topic.

If the XML response is used for data exchange, the xsdt='true' attribute should be set on the `<query>` element. This enables the use of XML Schema data types that are culture-independent. If the response is used for presentation purposes, it can be useful to also use xsdt if for example dates or numbers are formatted in an XSLT stylesheets. If the response is only transformed to HTML with no custom formatting/calculation, then the culture-variant output (with no xsdt attribute, or `xsdt='false'`) might be preferable.

Cursor Flags

Here you can learn about the cursor flags.

Flags to be used with the flags attribute:

Flag	Flag (dec)	Meaning
0x00000020	32	do not check rights for records of this table
0x00000040	64	No longer supported. See 0x00004000 below!
0x00000080	128	return records of this table as "summed" (only applicable for 1:N relations)
0x00000100	256	WITH: parent record is only returned when at least one record exists
0x00000200	512	EXISTS: records of this table are not returned (used in combination with WITH)
0x00000400	1024	NOT EXISTS: like above

Flag	Flag (dec)	Meaning
0x00000800	2048	OPTIONAL
0x00004000	16384	also read records marked as deleted. Please note: for Aurea CRM interface Service Pack 6 and older this flag (allowing for reading also records marked as deleted) is 0x00000040 (64)! CRM.interface0x00004000 Aurea CRM interface SP7+ 0x00004000 Aurea CRM interface SP6 0x00000040

Example: companies and persons are read, companies without persons are not returned

```
<request>
  <query>
    <tables>
      <table tablename="Company">
        <table tablename="Person" flags="256"/>
      </table>
    </tables>
    <fields tablename="Company" fields="Company"/>
    <fields tablename="Person" fieldname="LastName,FirstName"/>
    <condition>
     <cond tablename="Company" fieldname="Company" op="=" value="update*"/>

    </condition>
  </query>
</request>
```

Example: return the sum of a field (e.g. costs) for all contact records for one company

```
<request>
  <query>
    <tables>
      <table tablename="Company">
        <table tablename="Contact" flags="128"/>
      </table>
    </tables>
    <fields tablename="Company" fields="Company"/>
    <fields tablename="Contact" fields="Costs"/>
    <condition>
      <cond tablename="Company" fieldname="Company" op="=" value="update
software AG"/>
    </condition>
  </query>
</request>
```

Note: when sum fields are used, only one contact record is returned (and only fields that can be summed are considered).

mmFlags XML Schema data type

The mmFlags data type represents a 32-bit unsigned integer used to store up to 32 Boolean flags.

It can be specified as hexadecimal notation if prefixed with an 'x' character, otherwise it is interpreted in decimal notation.

5

Other Functions and features

Learn about the other interesting features and functions of CRM.interface.

How to remove namespaces

Learn how to remove namespaces.

CRM.interface requests have to be namespace-neutral. Requests containing namespaces are therefore ignored:

```
<Biztalk:request xmlns:Biztalk="urn:COMPANYMAME/CRM/U7COMPANYIN">
  <status/>
</Biztalk:request>
<Biztalk:request xmlns:Biztalk="urn:COMPANYMAME/CRM/U7COMPANYIN">
  <status/>
</Biztalk:request>
```

In case a client uses namespaces in its requests you may remove these namespaces via adoption of the stylesheets in your command lists.

- modify your `in.xslt` (search for "Biztalk" to find differences to the out-of-the-box stylesheet) as sketched in the sample below.
- add an additional style sheet (in this sample `namespace_remover.xslt`) to your command list. This new style sheet might look like the sketch below.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:var="urn:var"
xmlns:user="urn:user"
xmlns:syncml="SYNCML:SYNCML1.1"
xmlns:Biztalk='urn: COMPANYMAME/CRM/U7COMPANYIN'
exclude-result-prefixes="msxsl var user syncml Biztalk"
version="1.0">
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/request|/syncml:SyncML">
    <root>
      <com obj='$preload' func='XMLProcess'>
        <par val='$xmldom'/>
      </com>
    </root>
  </xsl:template>
  <xsl:template match="/Biztalk:request">
    <root>
      <xslt>namespace_remover.xslt</xslt>
      <com obj='$preload' func='XMLProcess'>
        <par val='$xmldom'/>
      </com>
    </root>
  </xsl:template>
  <!-- ... -->
</xsl:stylesheet>
```

Field output formats

Learn about the field output format flags.

The field output format flags used in the meta information.

Attribute	Meaning
DigitGrouping	The field is formatted with digit grouping symbols (eg. 1,234.56 instead of 1234.56).
OneDecimalPlaces	The field uses one decimal place for precision.
NoDecimalPlace	The field uses no decimal place for precision.
Percent	The field contains a percentage.
Signed	The field is signed.
DisplayEmpty	The field is numeric and should be formatted as “0” instead of an empty string.
ThreeDecimalPlaces	The field uses three decimal places for precision.
LeadingZeroes	The field is formatted with leading zeroes up to the maximum length.
Amount	The field contains an amount and is formatted likewise (eg. 69,- instead of 69).
FourDecimalPlaces	The field uses four decimal places for precision.
FiveDecimalPlaces	The field uses five decimal places for precision.
Selectable	The field has a clickable selection attached.
SevenDecimalPlaces	The field uses seven decimal places for precision.
DDMMYY	The field is a date with day, month and two-digit year.
DDMM	The field is a date with day and month.

Attribute	Meaning
MMYYYY	The field is a date with month and four-digit year.
MMYY	The field is a date with month and two-digit year.
RightAligned	The field is right-aligned.
Centered	The field is centered.
LeftAligned	The field is left-aligned.
SixDecimalPlaces	The field uses six decimal places for precision.
OnlyDigits	The field is a character field, but contains only digits.
TimeWithSeconds	The field is a time field with second precision.
TimeWithMilliseconds	The field is a time field with millisecond precision.
XmlSchema	The field is formatting using XML schema data types.

FieldTypes and Categories

Fields are classified into type and categories that are used in various places.

Below are the types and categories:

Field types

Type character	Meaning
C	Character (String)
K	Catalog (also called dynamic catalog)
X	Fixed catalog (also called static catalog)
L	Integer (32-bit or 64-bit)
S	Integer (16-bit)
D	Date
T	Time
F	Float (32-bit or 64-bit)
B	Boolean
Z	Virtual

Field categories

The field category is determined from the field id.

Category	Start	Finish	Remarks
BaseFields	0	999	
CoreFields	4000	4199	
UniqueFields	4200	4300	
VirtualFields			currently not used
GdmUpdateFields	5000	6999	
GdmCustomer-Fields	7000	9999	

Category	Start	Fin-ish	Remarks
GdmPartnerFields	10000	17999	
SqlCoreFields	18000	18999	
SqlVarFields	19000	22999	
CustomFields	25000		
GeneratedFields	1000000		
GdmFields			GdmUpdateFields+GdmCustomerFields+Gdm-PartnerFields

Text categories

The text category determines the origin of the fieldname.

Category	Remarks
CoreTexts	language-dll (eg. `bb_ger.dll`, `bc_eng.dll`, `fs_nld.dll`)
DefaultTexts	built-in dictionary (contained in the datamodel-dll)
CustomTexts	custom dictionary (dictionary.xml in the xml/ folder)
MissingTexts	
AllTextFlags	

Error codes

Use the list of error codes for troubleshooting purpose.

fCode	Define	Description
-1	ALREADY_LOGGED_ON	User is already logged on.
-2	READ_REGISTRY	Registry not properly configured. Register component!

fCode	Define	Description
-3	INIT_SESSION	Init of session failed.
-4	MMINIT1	Database unavailable! Check all DLL's present and ODBC config.
-5	MMINIT2	Default ('WWW') user not added to DB!
-6	DBINIT	Check DSN configuration and security settings for Windows User!
-7	INITDEFAULT	Init default values failed
-8	OPENFILES	Could not open file.
-9	INITUSER	Initialization of user failed (check formats and rights).
-10	NOTINITIALIZED	Session not initialized (no GlobalLogon())
-11	SESSIONPOOL	No session pool available.
-12	USERPOOL	No session from user pool available.
-13	LOGIN	Invalid Login. User/Password unknown!
-14	CONNECTION_LOCKED	Connection is locked!
-15	CONNECTION_INIT	Connection cannot be initialized!
-16	ALL_CONNECTIONS_USED	No free connections!
-17	PASSWORD	Incorrect password!
-18	MODULE_RIGHT	Module locked. User not allowed access to this module.
-19	PASSWORDEXPIRED	User has to change password!

fCode	Define	Description
-20	UNDEFINED_CURSOR_ID	Invalid cursorID! cursor already closed?
-21	WRONG_CURSOR_SETLEN	Invalid set length (key)
-22	BUFFER_TO_SMALL	Buffer too small.
-23	READPACKSTART	The requested object/information is not found.
-24	NOTFOUND	The requested object/information is not found.
-25	ACCESSDENIED	General "access denied" error-code
-26	MIGRATE_FIRST	Database has to be migrated.
-30	UPDREC_NOT_FOUND	Record to be updated could not be found
-31	UPD_RIGHT	No rights to update record!
-32	FIELDRIGHT	No permission to change this field!
-33	INVALID_KEY_FIELD	Invalid key value! Catalog value deleted?
-35	LOAD_RIGHT_PROFILE	Failed to load rights-format.
-36	LOAD_MRIGHT_PROFILE	Failed to load tenant's right-format.
-40	BAD_BLOB	Not a blob or decompression-failure.
-50	BAD_PARAM	Invalid parameter.
-51	INIT_CURSOR	Cursor not initialized.
-60	SYSTEM_LOCKED_MANUAL	Super user locked the system

fCode	Define	Description
-61	SYSTEM_LOCKED_DATE	System is locked because of current date
-62	SYSTEM_LOCKED_DAILY	Daily system lock is active
-63	USER_BLOCKED	The user is blocked by blocking mechanism
-64	USER_NOCONCURACCOUNT	No right for concurrent logon of user.
-65	ALL_LOGGEDON	Maximum number of users for logon reached.
-66	EXCLUSIVE_LOGON	Only one login per module allowed (cockpit).
-70	XML_FMTIO_ERROR	I/O-Error accessing XML format.
-71	XML_FMTIO_NOTFOUND	Format record is not found.
-72	XML_FMTIO_INVALIDTYPE	Unsupported XML format-type.
-73	XML_READ_ERROR	Error reading XML-format. Most likely the format is invalid!
-74	XML_INIT_ERROR	XMLlibrary(`mmxmlmssi/or8i.dll`) could not be loaded.
-75	XML_DIFFERENT_TABLE	A field referenced an unexpected in-fo area. The contents of the XML are invalid.
-76	XML_INVALID	XML to read is not valid.
-77	XML_EMPTY_FORMAT	Format for default values is empty and therefore invalid.
-78	XML_NORIGHT	No right to import format (usually data model format).

fCode	Define	Description
-80	RRC_NUMERIC_ERROR	Numeric range error.
-81	RRC_VALUE_ERROR	Invalid value.
-82	RRC_RANGE_ERROR	Range error.
-99	EXCEPTION	Unknown exception occurred
-100	MEMORY	Memory error: system is low on memory.
-101	TABLE	Invalid table. Usually causes by XML import of formats
-102	INVALID_FIELD	Invalid field. Usually causes by XML import of formats
-103	INVALID_CATALOG	Invalid catalog. Usually causes by XML import of formats
-104	INVALID_LANGUAGE	Invalid language. Usually causes by XML import of formats
-105	INVALID_REP	Invalid rep. Usually causes by XML import of formats
-106	PASSWORD_TOOSHORT	New password is to short (password rules)
-107	PASSWORD_MISSINGDIGITS	Password must contain digits (password rules)
-108	PASSWORD_MISSINGALPHA	Password must contain alphabetic characters (password rules)
-109	PASSWORD_CONTAINSUSERNAME	Password must not contain user name (password rules)
-110	PASSWORD_INHISTORY	The password is contained in the password history (password rules).

fCode	Define	Description
-111	PASSWORD_MISSINGUPPERLOW-ER	The password must contain upper- and lowercase characters (password rules).
-112	PASSWORD_INBLACKLIST	The password is in blacklist (pass-word rules).
-113	PASSWORD_NOCYCLE	The password must not share x characters with previous password (password rules).
-120	COPYRECORD_ORDER_AL-READY_EXISTS	Copy record: copy offer to order and destination record exists already (BTB only).
-121	COPYRECORD_INSTALLED-BASE_ALREADY_EXISTS	Copy record: copy order to installed base and destination record exists already (BTB only).
-122	COPYRECORD_TARGET_AL-READY_EXISTS	Copy record: destination record exist already (BTB only).
-123	COPYRECORD_OF-FER_IS_IN_STATE_ORDER	Copy record: copy offer to order and offer state equals 'order' (BTB only).
-124	COPYRECORD_MISS-ING_MUST_FIELD	Copy record: missing must field (BTB only).
-127	TICKETREPS_CONFIG_NAME	Dispatching dashboard: could not read configuration entry.
-128	TICKETREPS_READ_CONFIG_DATA	Dispatching dashboard: could not read configuration format.
-130	TICKETREPS_INVALID_FORMAT	Dispatchingdashboard:configuration format is invalid.
-132	TICKETREPS_DUP_SUB	Dispatching dashboard: could not copy conditions.

fCode	Define	Description
-133	TICKETREPS_DB_READ	Dispatching dashboard: error starting read engine.
-150	CONFIG_INVALID	Configuration table: invalid section/option combination.
-151	CONFIG_INVALIDFORMAT	Configuration table: invalid format name.
-158	CRYPTO_EXCEPTION	Error encrypting document.
-800	NUL_SQLERROR	Generic SQL error.
-801	NUL_CURSOR	No cursor available.
-810	NUL_CONN_EXIT	An attempt to establish a database connection failed and the calling application should exit.
-811	NUL_CONN_ERROR	An attempt to establish a database connection failed.
-812	NUL_CONN_REOPEN	The database connection could not be reopened.
-813	NUL_CONN_BAD	An invalid database connection has been specified.
-815	NUL_CURS_ERROR	An invalid cursor has been specified.
-816	NUL_CURS_ALREADY_LOCK	A cursor has already been locked.
-900	NUL_READ_DELETED	The read is successful, but the record is marked as deleted.
-901	NUL_READ_ABORT	The read operation has been aborted.
-902	NUL_READ_GETTABLE_FAILED	GetTable failed on reading a record.

fCode	Define	Description
-903	NUL_READ_GETMUTEX_FAILED	GetMutex failed on reading a record.
-904	NUL_READ_OUTOFMEMORY	GetMemory failed on reading a record.
-1000	NUL	Record modification error.
-1001	NUL_NOTFOUND	Record not found.
-1002	NUL_DUPLICATE	Record already exists.
-1004	NUL_ART	Unknown read type.
-1008	NUL_RECCHANGED	The record has been modified.
-1012	NUL_OPEN	File cannot be opened!
-1013	NUL_CONTINUE_EDIT	Unable to complete write. Rights/must field problem.
-1016	NUL_INCOMPLETE	Record invalid! Record may have been deleted.
-1017	NUL_NOMOREIDS	The number cycler is full.
-1018	NUL_PERSONS	A company could not be deleted because related persons still exist.
-1019	NUL_EXCEEDS_CONTINGENT	OTC Serial entry: exceeds contingent.
-1020	NUL_RIGHT_DENY	Access to the info area has been denied.
-1021	NUL_RIGHT_CONDITIONAL	Access to the info area has been denied by a conditional right.
-1022	NUL_RIGHT_TOPDOWN	Access to the info area has been denied by a top-down right.

fCode	Define	Description
-1023	NUL_RIGHT_BOTTOMUP	Access to the info area has been denied by a bottom-up right.
-1024	NUL_RIGHT_MUST	Access to the info area has been denied because a mandatory field contained no data.
-1025	NUL_INV_TIMESPAN	Record could not be saved because date constraints are not fulfilled.
-1160	NUL_WH_NO_TERMIN_POSSIBLE	Recurrence record: no contact with current settings possible (daily, weekly)
-1161	NUL_WH_NO_DATE	Recurrencerecord:RecurrenceCount or EndDate missing
-1162	NUL_WH_STARTENDDATE	Recurrence record: EndDate before start date of parent contact
-1163	NUL_WH_NO_PERIOD	Recurrence record: period missing (daily:DailyRecurrence, weekly:WeeklyRecurrence, monthly:MonthlyRecurrence, yearly:YearlyRecurrence)
-1164	NUL_WH_NO_WEEKDAY	Recurrence record: weekday missing (weekly[16-22]:Monday to Sunday)
-1165	NUL_WH_NO_MONTH	Recurrence record: Month missing (yearly[28])
-1166	NUL_WH_INVALID_WEEKDAY	Recurrence record: invalid weekday or Saturday or Sunday selected and Weekend = false (monthly[26]:MonthlyRecurrenceWeekday , yearly[31]:YearlyRecurrenceWeekday)
-1167	NUL_WH_NO_MONTHDAY	Recurrence record: day of month is missing (monthly[24]:MonthlyRecurrenceMonthday,yearly[29]:YearlyRecurrenceWeekday)

fCode	Define	Description
-1168	NUL_WH_WRONG_STNO	Recurrence record: Record must not be changed on a station different from where it is created
-1169	NUL_WH_NO_CONTACTDATE	Recurrence record: Appointment record does not contain a date
-1100	NOT_SUPPORTED	Feature not supported!
-2001	NUL_US_ID	Update US record: rep type must be rep
-2002	NUL_US_USER	Update US record: invalid user name
-2003	NUL_US_USER_ALIAS	Update US record: invalid alias name
-2004	NUL_US_PW	Update US record: invalid password
-2005	NUL_US_USER_CHG	Update US record: user name may not be changed
-2011	NUL_ES_USER_UID	Update ES record: rep type must be rep
-2012	NUL_ES_PWGLOBAL	Update ES record: pwGlobal only if snoMaster != 0
-2013	NUL_ES_ONLY_MASTER	Update ES record: field can only be changed by snoMaster
-2014	NUL_ES_USER	Update ES record: invalid user name
-2015	NUL_ES_ALIAS	Update ES record: invalid alias name
-2016	NUL_ES_PW	Update ES record: invalid password
-10000	RRC_TABLE_RIGHT	No access to table because of rights (code is 1000 + table ID)