cappatec

Core Engine

R×XML Specification

Version 2, February 2013

Applicable for Core Engine 1.0 P, Core Engine 1.0 A, Core Engine 1.1 A

Authors:

Fabian Knirsch, BSc
Oliver Langthaler, BSc

Table of Contents

Cappatec Core Engine XML Interface ... 4
- Introduction ... 4
- Purpose of This Document ... 4
- Notations ... 4

XML Header ... 6

Document Types ... 6

Representing Data ... 6
- Data ... 6
- Set ... 6
- Description / Row ... 6
- Group ... 7
- Column / Cell ... 7
- Reference ... 7
- Primary Key ... 8

Logon ... 9
- Request ... 9
- Response ... 9

Logoff ... 9
- Request ... 9
- Response ... 9

Data ... 10
- Request ... 10
- Response ... 10

Describe ... 11
- Request ... 11
- Response ... 11

Submit ... 12
- Request ... 12
- Response ... 12

Cancel ... 13
- Request ... 13
- Response ... 13

General Responses ... 14
- Success Response ... 14

Error Response ... 14

Sample communication ... 16

Recommendations ... 18

Enterprise-Specific Definitions ... 18

Implementation Recommendations for Client developers ... 18

Implementation Recommendations for activity developers ... 18

Disclaimer and Contact Information ... 19

Disclaimer ... 19

Contact Information ... 19

Copyright Note ... 19

Cappatec Core Engine XML Interface

Introduction

Internally, the Cappatec Core Engine handles all data in a relational, set-oriented manner. As a consequence, the interface as well as the entire protocol between the engine and its Clients is structured in a similar fashion. In order to provide a high level of interoperability with established technical and commercial applications, XML has been chosen as uniform interface language. This necessitates a new standard, allowing data to be arranged in a relational manner within XML. R×XML is the result of these considerations. Its main strength is the incorporation of the relational paradigms while introducing a certain amount of abstraction, which is necessary to become independent of any physical representation. Furthermore, it integrates a protocol which enables authentication and session handling.

Purpose of This Document

This document provides a detailed description of the Cappatec Core Engine XML Interface (R×XML). It is intended for developers of Cappatec Core Engine activities and Client applications. Specifically, this document provides:

i) A general overview of the Cappatec Core Engine, supported transmission protocols and application layer protocols.
ii) An introduction to the R×XML interface and data representation.
iii) Detailed descriptions of supported Requests and the appropriate Responses.
iv) Recommendations for implementation and enterprise specific extensions.

Readers should note that this document will not describe language specific implementation details, but implementation recommendations. This specification is intended to describe the XML-based Core Requests and Responses on an abstract and generalized level.

Notations

If not explicitly stated, any R×XML code shown in this document consists of fragments. Therefore, it is not valid as it is provided here. Omissions are usually, but not necessarily, represented as: …

Example:

```
<Request type="logon">
     …
</Request>
```

The mandatory data section is omitted here and might be explained elsewhere.

This specification may contain Cappatec specific descriptions. They are represented in the following format.

Example:

Cappatec Specific

Responses may contain a `<Pragma_ctTitle>` tag, which represents a title that will be displayed in Cappatec Client windows.

End Cappatec Specific

XML Header

All R×XML documents sent to the Core are required to follow the XML 1.0 specification and should be UTF-8 encoded, resulting in the following header:

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

Document Types

There are 2 main types of R×XML documents, requests and responses, denoted by corresponding `<Request>` and `<Response>` tags. Clients may send requests, while any documents returned by Core will be responses. The nature of the request or response needs to be specified in an attribute named type. For example, a Client who wishes to logon to the Core would have to send:

```
<Request type="logon">
```

Any further requests need to include attributes containing the session ID (SID) which has been returned by the Core in case of a successful logon, as well as an instance ID (IID), which can be chosen freely by the Client. Valid characters for the IID string include a…z, A…Z and 0…9:

```
<Request type="data" sid="…" iid="…">
```

Representing Data

Data

Any information sent in a request or response must be enclosed by a `<Data>` tag. With the exception of some basic system requests and responses (such as a logon request or a success response), which follow a simplified pattern, a `<Data>` tag consists of one or more `<Set>` tags.

Set

Each `<Set>` is divided into two distinct sections:

i) A `<Description>`, which contains all necessary information about the nature of the transmitted data, such as the data type, the data grouping, the amount of valid values or the column names. It may be considered a heavily augmented table header in relational terms.
ii) One or many `<Row>` tags, containing the actual data. Each `<Row>` must exactly match the data pattern established in the `<Description>`, otherwise the data cannot be parsed properly.

Description / Row

Each `<Description>` (and, by extension, each `<Row>`) is required to contain at least one `<Group>`.

Group

In order to enable Clients to display information in a more structured manner to a user, data is arranged in groups. A `<Group>` within a `<Description>` contains one or more `<Column>` tags, while a `<Group>` within a `<Row>` contains one or more `<Cell>` tags.

Column / Cell

`<Column>` tags may exclusively exist within the `<Description>` section of a document and must each have a corresponding `<Cell>` within each following `<Row>`. Their value can be considered the column header in relational terms (hence the name column). They may contain the following attributes, of which all but the first are intended exclusively to provide a Client with necessary information:

i) type (mandatory): the type of data transmitted in the corresponding `<Cell>`

Type	Parsed as
short	Java Short (16 bit)
int	Java Integer (32 bit)
long	Java Long (64 bit)
float	Java Float (32 bit)
double	Java Double (64 bit)
string	Java String (max. length: 2^{31} - 1)
password	Java String (masking support)
multiline	Java String (line break support)
bool	Java Boolean
date	Java Date (Epoch)
time	Java Time (Epoch)
timestamp	Java Timestamp (Epoch)
reference	containing `<Reference>` tags
lob	Java String (base64)

ii) key: marks a column as primary key column (for select lists)
iii) display: sets the display name for a column
iv) mandatory: marks mandatory input
v) locked: marks a column as read-only
vi) length: sets the maximum input length of a column
vii) referenceprocess: sets the reference process for a “Browse…” button
viii) referenceactivity: sets the reference activity for a “Browse…” button
ix) min: sets the minimum number of selections for a “Browse…” button
x) max: sets the maximum number of selections for a “Browse…” button (values >1 change to inf)

`<Cell>` tags may only contain one single Boolean attribute, “null”, which must be set to true in order to transmit null values. Otherwise, their value is parsed according to its describing `<Column>`.

Furthermore, both `<Column>` and `<Cell>` tags may contain a `<Reference>` tag.

Reference

`<Reference>` tags are used to transmit `<Primary Key>` tags. Composite keys are represented by multiple `<Primary Key>` tags within a single `<Reference>`.

Primary Key

If located within a `<Description>`, the value of the `<Primary Key>` tag is used as the name of the primary key, and it must contain a “type” attribute, much like a `<Column>`. However, the type values “password” and “lob” are not applicable for primary keys.

If located within a `<Row>`, its value is parsed according to the type attribute of the describing `<Primary Key>`. Unlike a `<Cell>`, a `<Primary Key>` may, of course, not contain a null attribute which is set to true.

Logon

Due to the simple nature of the procedure, the plain code is provided below. Note that the Core’s success response contains the 500 bit hex coded session ID (SID), which may be increased in size in the future.

Request

```
<Request type="logon">
     <Data>
          <Username>randallmunroe</Username>
          <Password>correcthorsebatterystaple</Password>
     </Data>
</Request>
```

Response

```
<Response type="success">
     <SID> … </SID>
</Response>
```

Logoff

Due to the simple nature of the procedure, the plain code is provided below.

Request

```
<Request type="logoff" sid="…">
</Request>
```

Response

```
<Response type="success">
</Response>
```

Data

Request

As the name implies, data requests are used to query the Core for data. Besides the mandatory process and activity information, a `<Subsection>` tag may be added, enabling the Core to distinguish between different data requests for the same activity. If a `<Description>` is to be included in the Core's response, an `<IncludeDescription>` tag containing "true" as value may be added. Information about the requested data is to be sent in form of a `<Set>`.

```
<Request type="data" sid="…" iid="…">
     <Data>
          <Process>…</Process>
          <Activity>…</Activity>
          <Subsection>…</Subsection>
          <IncludeDescription>true</IncludeDescription>
          <Set>
               …
          </Set>
     </Data>
</Request>
```

Response

The following data response may be the result of the request shown above. Since `<IncludeDescription>` has been set to true, a `<Description>` has been included:

```
<Response type="data">
   <Data>
      <Set>
         <Description>
            <Group>
               <Column type="string" length="100" …>…</Column>
               …
            </Group>
         </Description>
         <Row>
            <Group>
               <Cell>…</Cell>
               …
            </Group>
         </Row>
      </Set>
      …
   </Data>
</Response>
```

Describe

Request

In order to be able to display user input forms, Clients may query the server through a describe request, returning all the information required to display the form and apply input restrictions. Besides the session and instance IDs, only the name of the process and activity which are to be described are required:

```
<Request type="describe" sid="…" iid="…">
     <Data>
          <Process>…</Process>
          <Activity>…</Activity>
     </Data>
</Request>
```

Response

The resulting response is a data response which includes a detailed description, utilizing the attributes the `<Column>` tag offers. Furthermore, a `<Row>` may be included to transmit default values. If no default values need to be set beyond a certain point, the `<Group>` and `<Row>` tags may be closed prematurely (before a `<Cell>` has been added for every `<Column>`), resulting in the only valid case of a `<Description>` / `<Row>` mismatch.

```
<Response type="data">
   <Data>
      <Set>
         <Description>
            <Group>
               <Column type="string" length="100" …>…</Column>
               …
            </Group>
         </Description>
         <Row>
            <Group>
               <Cell>…</Cell>
               …
            </Group>
         </Row>
      </Set>
   </Data>
</Response>
```

Submit

Request

Clients may send submit requests in order to transfer data to the Core. The pattern is somewhat similar to a data response from the Core. However, the `<Description>` will only be parsed for the type attributes within `<Column>` and `<Primary Key>` tags in order to determine the data types of the received information.

```
<Request type="submit" sid="…" iid="…">
   <Data>
      <Process>…</Process>
      <Activity>…</Activity>
      <Set>
         <Description>
            <Group>
               <Column type="string">…</Column>
               …
            </Group>
         </Description>
         <Row>
            <Group>
               <Cell null="false">a</Cell>
               …
            </Group>
         </Row>
      </Set>
   </Data>
</Request>
```

Response

After the submit request has been successfully processed by the Core, it will return a success response (see “General Responses”).

Cancel

Request

Activities, which a Client triggers on the Core (by sending a request) remain persistent until a cancel request is received or the user is logged off. In order to release associated Core and Database resources such as memory and locks, activities should always be terminated by a cancel request:

```
<Request type="cancel" sid="…" iid="…">
     <Data>
          <Process>…</Process>
          <Activity>…</Activity>
     </Data>
</Request>
```

Response

Upon successful resource de-allocation, the Core will return a success response (see “General Responses”).

General Responses

Cappatec Specific

Responses may incorporate the following tags above the <Data> tag:

Tag	Description
<Pragma_ctTitle>	contains a title which is displayed in Cappatec Client windows
<Pragma_ctSelect>	Boolean, informs the Cappatec Client that the following data is a select list in which checkboxes need to be displayed

End Cappatec Specific

Success Response

The success response is the most basic of all responses. If a request which requires no further feedback (such as a submit or logoff request) is sent to the Core, its successful completion will be signaled by the following response:

```
<Response type="success">
</Response>
```

In case of a logon request, the success response will contain the session ID (SID):

```
<Response type="success">
     <SID> … </SID>
</Response>
```

Error Response

Core Engine errors, warnings and exceptions are represented as three digit error IDs (EIDs), which are contained in an <EID> tag within the <Data> section of the response.

EID	Description	Comment
101	Bad credentials	Username or password incorrect
102	Passwords not matching	May occur during password changes
103	User inactive	Use the Core management to re-activate the user
104	User locked	Use the Core management to unlock the user
105	Bad session ID	New login required
201	Database connection error	A required DB connection could not be established
202	Database query error	A DB query produced an error
203	Database error	General DB error
210	Activity loading error	An error occurred while loading the activity
211	Activity instantiating error	An error occurred while instantiating the activity
212	Activity unloading error	An error occurred while unloading the activity
213	Activity not found	Make sure a corresponding class file exists
214	Activity error	General activity error
301	Unknown request type	The specified request type is not known to the Core
302	Request type not implemented	The specified request type has not been implemented
303	Request parsing error	An error occurred while parsing the request

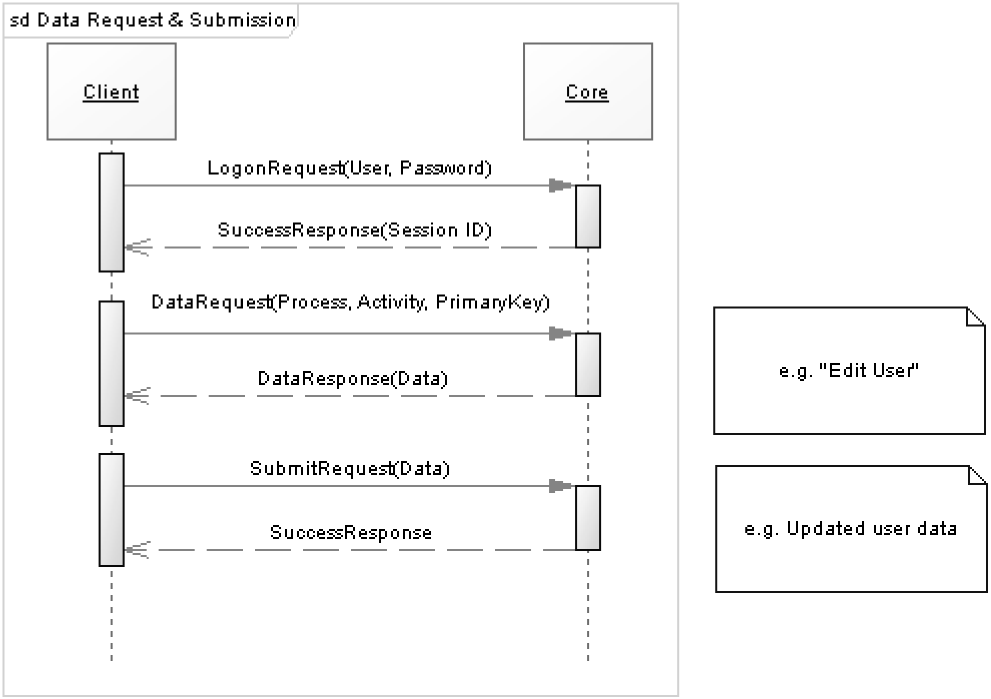
401	Activity not allowed	No permission to access the specified activity
501	Duplicate user	Choose a different user name
502	Duplicate group	Choose a different group name
503	Duplicate process	Choose a different process name
504	Duplicate activity	Choose a different activity name
601	Read only lock	Write request cannot be completed due to a lock
602	Full lock	Request cannot be completed due to a lock
603	Locked	Request cannot be completed due to a lock
999	Unknown error	All is lost

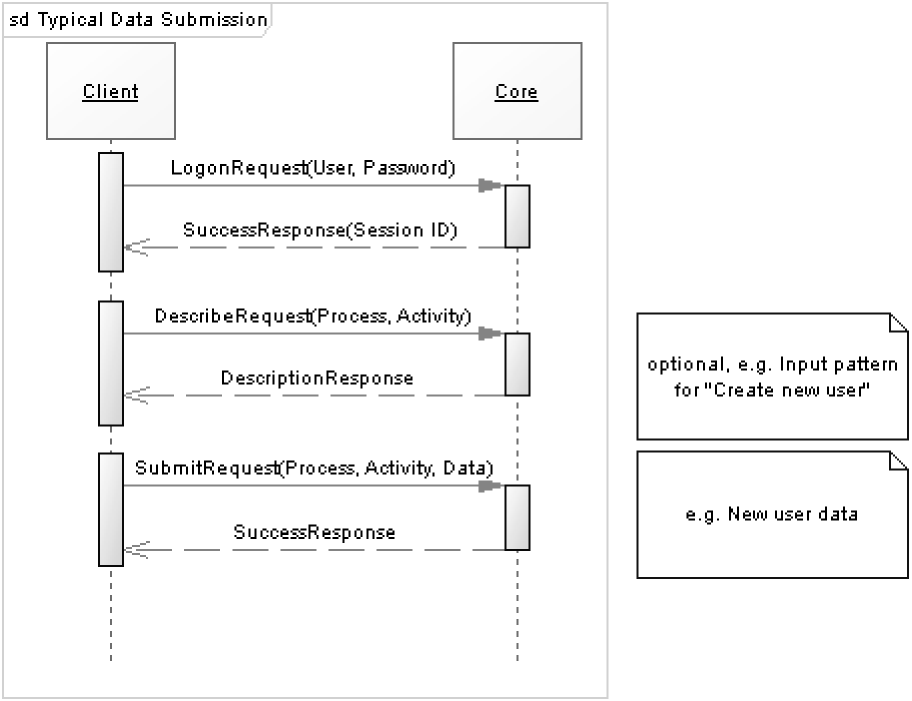
Here is an example of a typical error response:

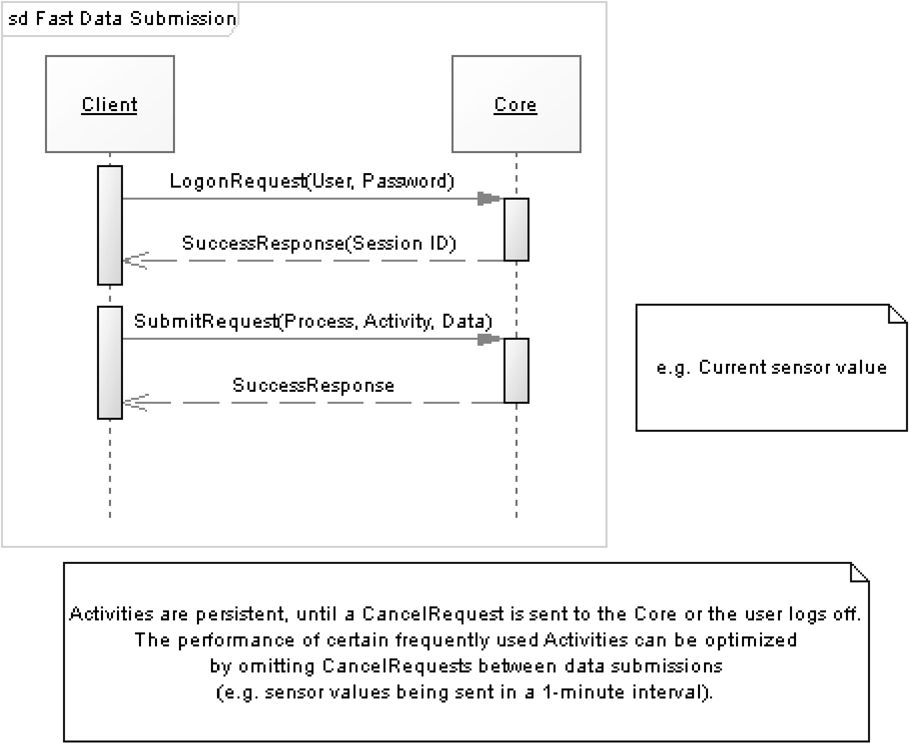
```
<Response type="error">
     <Pragma_ctTitle>Locked</Pragma_ctTitle>
     <Data>
          <EID>603</EID>
     </Data>
</Response>
```

Sample communication

The following sequence diagrams are intended to illustrate the order in which requests and responses are exchanged between the Core and a Client.


sd Data Request & Submission
Client
Core
LogonRequest(User, Password)
SuccessResponse(Session ID)
DataRequest(Process, Activity, PrimaryKey)
DataResponse(Data)
SubmitRequest(Data)
SuccessResponse
e.g. "Edit User"
e.g. Updated user data


sd Typical Data Submission
Client
Core
LogonRequest(User, Password)
SuccessResponse(Session ID)
DescribeRequest(Process, Activity)
DescriptionResponse
SubmitRequest(Process, Activity, Data)
SuccessResponse
optional, e.g. Input pattern for "Create new user"
e.g. New user data


sd Fast Data Submission
Client
Core
LogonRequest(User, Password)
SuccessResponse(Session ID)
SubmitRequest(Process, Activity, Data)
SuccessResponse
e.g. Current sensor value

Activities are persistent, until a CancelRequest is sent to the Core or the user logs off.
The performance of certain frequently used Activities can be optimized
by omitting CancelRequests between data submissions
(e.g. sensor values being sent in a 1-minute interval).

Recommendations

Enterprise-Specific Definitions

Clients may define their own pragma flags if necessary. In a request, these flags should be located above the `<Data>` tag. Server-side parsing of these tags is currently not supported by Core libraries and needs to be done manually.

Naming convention: Pragma_*<tag name>*

Implementation Recommendations for Client developers

General

- Clients may call the activity "Home" to receive a complete list of accessible activities and activity types.
- Clients may omit describe requests if they are familiar with the data structure. If no Client will ever call a describe request for a specific activity, it does not even have to be implemented within the activity.
- In order to minimize server load, Clients should do as many input sanity checks as possible.
- Receiving error code 999 should not to be taken lightly, a detailed analysis is imperative.
- Clients should not keep obsolete connections and send a logoff request whenever feasible.

Persistency

- Activities are instantiated upon first request based on SID, IID, process and activity.
- Clients may use the IID to maintain multiple instances of one or more activities.
- The IID may be chosen freely by clients and may contain any un-escaped, valid XML character. However, ascending numbers starting at 1 are recommended.
- Activities are persistent within the Core and may be accessed by Clients as long as their requests refer to the same SID, IID, process and activity.
- Activities are automatically unloaded upon user logoff or timeout.
- Clients should always send a cancel request to unload an activity upon completion.

Implementation Recommendations for activity developers

Columns/cells with a locked attribute may be used to send data to Clients that needs to be returned in Client responses (similar to hidden form fields in HTML).

Disclaimer and Contact Information

Disclaimer

The information herein is solely provided for informational and development purposes. No license to any intellectual property right is granted by this document or in connection with the usage of Cappatec products.

Cappatec assumes no liability whatsoever and disclaims any warranty relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or non-infringement. Under no circumstance shall Cappatec be liable for any direct, indirect, consequential, punitive, special or incidental damages. This includes, without limitation, damages for business interruption and damages for loss of information or profits arising out of the use or inability to use this document, even if Cappatec has been advised of the possibility of such an event.

Cappatec makes no representations or warranties concerning the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Cappatec does not guarantee any updates to the information contained within this document. Cappatec products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life unless specifically stated otherwise.

Contact Information

The Cappatec Core Engine developer team:

Fabian Knirsch, BSc	fabian.knirsch@cappatec.com
Oliver Langthaler, BSc	oliver.langthaler@cappatec.com

www.cappatec.com

Copyright Note