cappatec

Core Engine

R×XML Specification

Version 5, February 2016

Applicable for Core Engine 1.5

Author:

cappatec OG, Salzburg/Austria

Table of Contents

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.

History

Changes in February 2016 version of this document:

- This document now applies for Cappatec Core Engine 1.5; for previous versions see the January 2014 specification
- `unrestricted` attribute added for Reference Sets
- `<Profile>` tag added for requests and responses
- Enhanced profile description for subsections
- R×XML Profiles added

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 3 main types of R×XML documents, requests, responses and containers, denoted by corresponding `<Request>`, `<Response>` and `<Container>` tags. Clients may send requests, while any documents returned by Core will be responses. Containers are a generalization of requests and responses and may be used to store, import or export data in R×XML format. 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. Additionally, the dot character . can be used, e.g. for an inverted domain namespace such as com.cappatec.client.1:

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

Containers do not include the attributes `type`, `sid` and `iid`. Note that containers are not suitable for data exchange with the Core Engine as they lack abilities to address sessions and instances.

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 no, one or more `<Set>` tags.

Set

Each `<Set>` may contain a `name` attribute and 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.

The name attribute is mandatory if the set is referenced from a Column of type reference.

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	Short (16 bit)
int	Integer (32 bit)
long	Long (64 bit)
float	Float (32 bit)
double	Double (64 bit)
string	String (max. length: 2^{31} - 1)
password	String (masking support)
multiline	String (line break support)
bool	Boolean
date	Date (Epoch)
time	Time (Epoch)
timestamp	Timestamp (Epoch)
reference	containing `<Reference>` tags
lob	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 contained `<PrimaryKey>` tags. Contains the name of the Process (requires referenceactivity, not combinable with referenceset)

viii) referenceactivity: sets the reference activity for contained `<PrimaryKey>` tags. Contains the name of the Activity (requires referenceprocess, not combinable with referenceset)

ix) referenceset: sets the reference set for contained `<PrimaryKey>` tags. Contains the name of the Set (not combinable with referenceprocess and referenceactivity)

x) unrestricted: if true, allows to set a value for this field that is not contained in the referenced set. Activities are recommended to add this value to the reference set as an option for future requests.

xi) min: sets the minimum number of selections for foreign `<PrimaryKey>` tags

xii) max: sets the maximum number of selections for foreign `<PrimaryKey>` tags (values >1 are changed 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 `<PrimaryKey>` tags. Composite keys are represented by multiple `<PrimaryKey>` tags within a single `<Reference>`.

Cappatec Specific

The use of composite keys is discouraged. As of version 1.5 Cappatec Client does not support composite keys in legacy mode.

End Cappatec Specific

Primary Key

If located within a `<Description>`, the value of the `<PrimaryKey>` 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 `<PrimaryKey>`. Unlike a `<Cell>`, a `<PrimaryKey>` may, of course, not contain a null attribute which is set to true.

Container

A container is a simple structure that contains one or more sets. Unlike other types (such as Logon, Logoff, Description, Data, Submit and Cancel) containers cannot be used directly to exchange data with the Core. Containers, however, provide an efficient way to import, export or store data.

```
<Container>
     <Data>
          <Set name="set1"> … </Set>
          …
     </Data>
</Container>
```

Logon

Due to the simple nature of the procedure, the plain code is provided below. Note that the Core’s success response contains the up to 512 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. An optional profile declaration references a profile for this request by fully qualified name (namespace and name) and by a line number (see section R×XML Profiles). 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>
          <Profile>
               <Name>…</Name>
               <Number>…</Number>
          </Profile>
          <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 `<PrimaryKey>` 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~~ Version 4, January 2014: Deprecated, this tag is not used anymore and will be removed permanently in future versions.

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
110	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>
```

R×XML Profiles

R×XML is a powerful scheme for representing data in a set oriented manner and provides a comprehensive protocol for data exchange, including meta-information about sessions, instances, processes and activities.

The protocol itself, however, is generic. Plain R×XML defines types of requests and responses, but it does not describe how to string together various types to get a consecutive communication between clients and the Core.

For this purpose, R×XML is being specialized to meet the requirements of a certain application by defining profiles. A profile describes how and in which order certain R×XML types are used to implement a concrete application scenario. For instance, requesting a simple set from the Core can be realized by sending a plain Data Request. An editing operation in a table, by contrast, will probably involve multiple requests of different types.

To define profiles the following notion is used:

```
Profile: [<namespace>.]<name>
{<number> →|← <type>[:"<subsection>"] [(<comment>)]}
```

A profile is given a unique name. The left-to-right arrow indicates a request, i.e. client to Core; the right-to-left arrow indicates the response, i.e. Core to client. In addition to the request type, a corresponding subsection may be indicated. Optionally, the comment describes concrete activities or arguments.

Example:

```
Profile: com.cappatec.client.listcustomer
1 → data: "2" (request data from CustomerManagement.ListCustomers)
2 ← data
3 → cancel
```

In this example, the client sends a data request for subsection “2” and expects a data response. After receiving a data response, the client cancels the activity.

Note that logon and logoff as well as error or success responses are usually not explicitly modeled in a profile. Profiles are very suitable for graphical representation such as sequence diagrams.

Profiles themselves are not strictly governed by the Core, even though they can have significant impact on the behavior of an activity. Profiles should therefore be seen as behavior patterns for clients when doing complex interactions with activities. Activities may therefore also support more than one pattern. A common example is the Describe Request that can be omitted when the structure of the response set is known.

Cappatec Core Engine 1.5 allows an optional `<Profile>` tag in requests to inform the Core about the expected behavior.

Default Profiles

For Cappatec Client a number of default profiles are predefined.

Cappatec Specific

~~In Cappatec Client profiles are referred to as being the *action type* of an activity.~~

Version 5, February 2016: Deprecated, action types are no longer defined along with activities for core management. Cappatec web client support for legacy-activities only.

End Cappatec Specific

The following is a list of predefined profiles in the com.cappatec.rxxml namespace. These profiles cover all the basic interaction scenarios.

```
Profile: com.cappatec.rxxml.insert
1 → describe
2 ← description
3 → submit
4 → cancel


Profile: com.cappatec.rxxml.update
1 → describe
2 ← description
3 → submit: "Selection"
4 ← description
5 → submit
6 → cancel


Profile: com.cappatec.rxxml.delete
1 → describe
2 ← description
3 → submit
4 → cancel


Profile: com.cappatec.rxxml.function
1 → describe
2 ← description
3 → submit
4 ← data
5 → cancel


Profile: com.cappatec.rxxml.menu
1 → data
2 ← data
```

```
3 → cancel


Profile: com.cappatec.rxxml.select
1 → data
2 ← data (include description)
3 → cancel
```

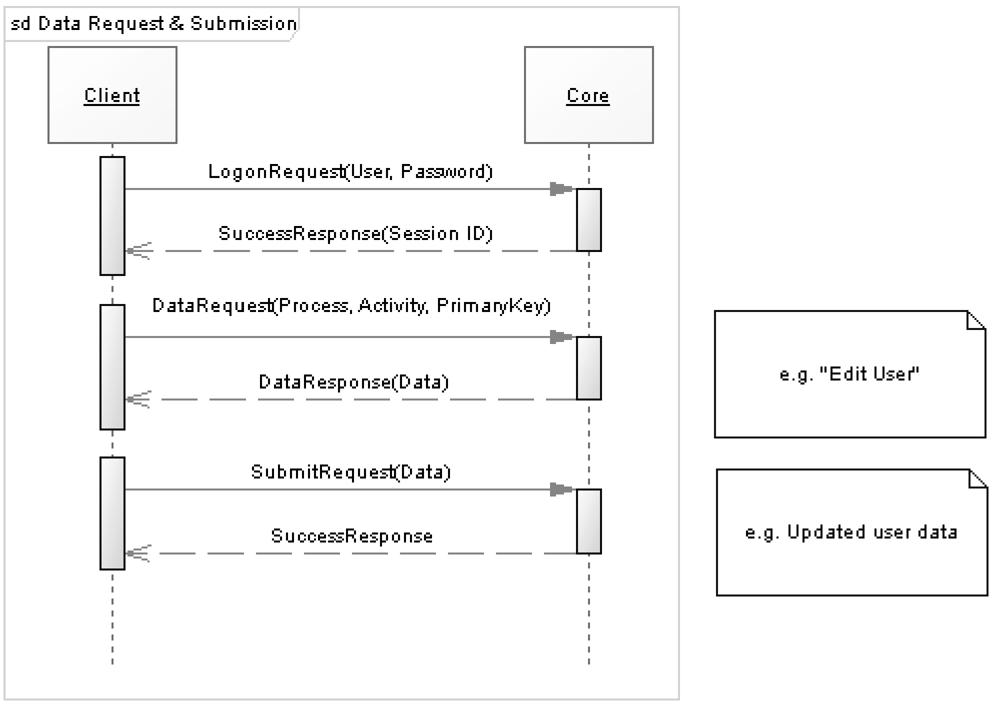
Extensions

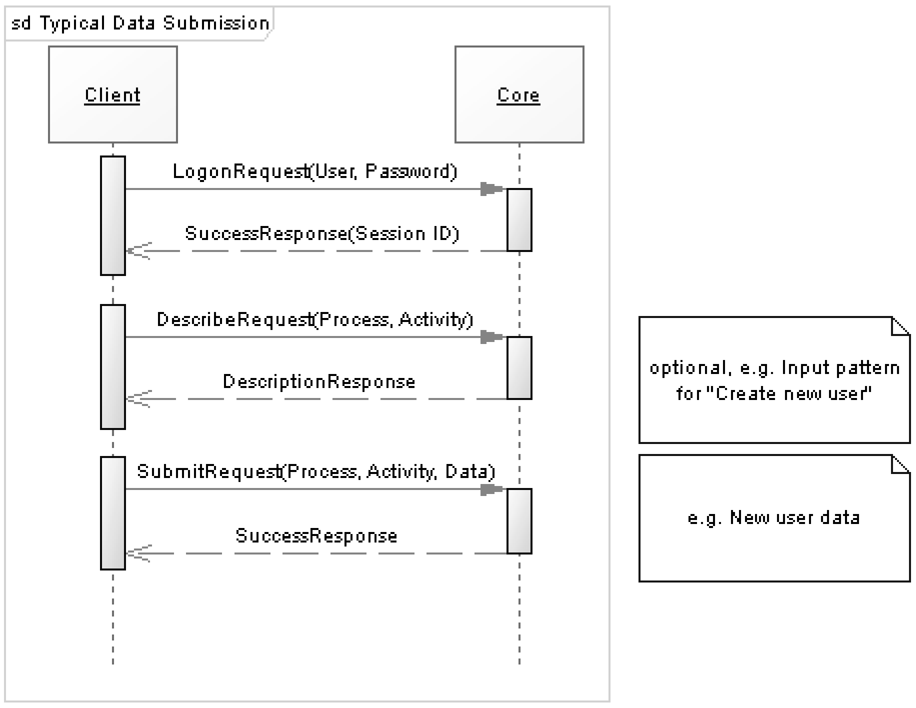
R×XML profiles can easily be extended to fully align with specific project requirements. For that purpose, simply define a profile that is familiar to both the client application and to the Core activities this application addresses.

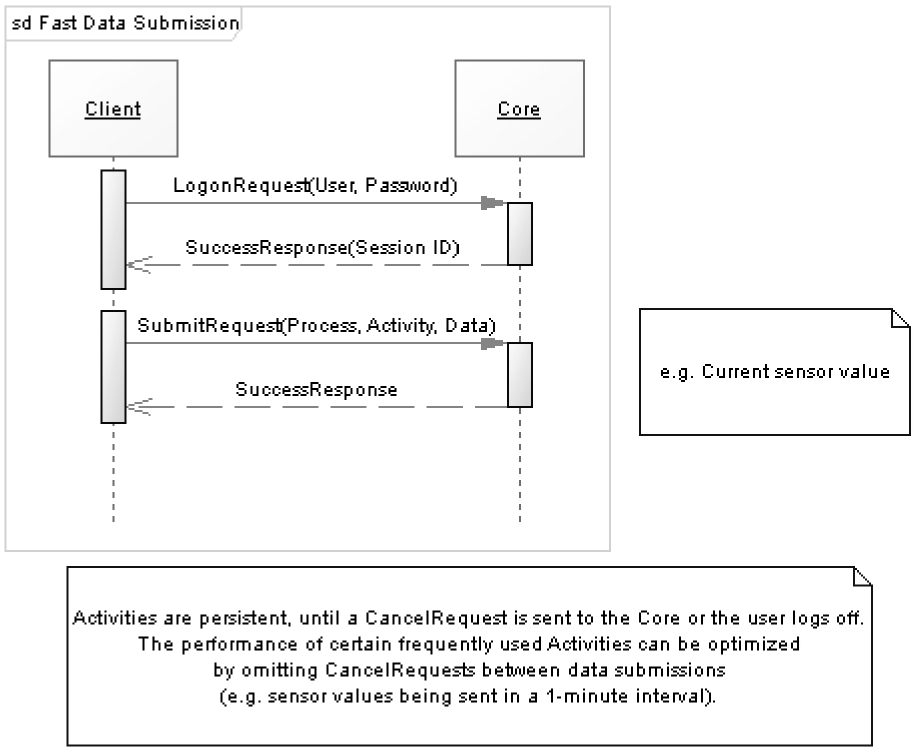
It is recommended to adhere to default profiles as far as practicable.

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 using the default profiles.


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. If namespaces should be used the inverted domain namespace notion is 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 explicitly declared as stateless will not be persistent and are automatically unloaded upon sending a response.
- Persistent 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

Contact Information

cappatec OG

Schlossallee 7/1

A-5412 Puch/Salzburg

www.cappatec.com

info@cappatec.com

Copyright Note