



XMPP

XEP-0204: Collaborative Data Objects

Dave Bryson
<mailto:dbryson@mitre.org>

Dan Winkowski
<mailto:winkowsk@mitre.org>

Michael Krutsch
<mailto:michael@mitre.org>

Chad Smith
<mailto:chadsm@mitre.org>

Jasen Jacobsen
<mailto:jasenj1@mitre.org>

Marshall Huss
<mailto:mhuss@mitre.org>

2007-01-17
Version 0.1

Status	Type	Short Name
Deferred	Standards Track	TO BE ISSUED

This document specifies an XMPP protocol extension that supports the exchange of structured data objects.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 - 2011 by the [XMPP Standards Foundation](#) (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <http://xmpp.org/about-xmpp/xsf/xsf-ipr-policy/> or obtained by writing to XMPP Standards Foundation, 1899 Wynkoop Street, Suite 600, Denver, CO 80202 USA).

Contents

1	Introduction	1
2	Requirements	1
3	Usage Model	2
3.1	Strict Synchronization mode	2
3.2	Lazy Synchronization mode	3
4	Description Language	3
5	Protocol	4
6	Use Cases	7
6.1	Query a client to determine if it supports a CDO message	7
6.2	Query the Server to determine available CDO types	8
6.3	Query the Server to determine the state of a specific CDO	9
6.4	Query an endpoint for a list of CDOs	9
6.5	Create a new CDO	10
6.6	Create a new Item on an existing CDO	12
6.7	Update an Item on an existing CDO	13
6.7.1	Update Styles	15
6.8	Delete an Item on an existing CDO	16
6.9	Retire an existing CDO	17
7	Errors	18
7.1	CDO specific conditions	22
7.2	Invalid Constraint condition types	23
7.3	Special Case: Outdated version error	23
8	Implementation Notes	24
8.1	Service	24
8.2	Client	24
9	Security Considerations	25
9.1	User Authentication and Authorization	25
9.2	End-to-End Encryption	25
10	IANA Considerations	25
11	XMPP Registrar Considerations	25
11.1	Protocol Namespace	25
12	XML Schema	26
12.1	Description Language	26

12.2 Data Synchronization Protocol	28
----------------------------------------------	----

1 Introduction

While the value of IM and Multi-user chat is obvious to anyone reading this JEP, the potential for ambiguity and miscommunication (particularly in a structured data environment) may not be. There are several domains where text communication is accompanied by a need to exchange structured data, e.g.: a help desk dealing with trouble tickets; a financial institution dealing with trades (buy and sell orders), an emergency scenario with first responders. The purpose of this JEP is to define a set of Collaborative Data Object (CDO) protocols that support the exchange of structured data objects. These data objects are explicitly described by a declarative language, instantiated as structured XML, and transported as extended XMPP stanzas. This JEP defines a protocol for exchanging these structured CDOs as part of a session conversation (either IM or Multi-User Chat) based around a strict synchronization model. In a strict synchronization model, participants will receive every change made to a CDO. An alternative synchronization model, a lazy synchronization model is also described to support users who either do not wish, or are not able because of infrastructure limitations, to receive the CDO stanzas in real time and wish to explicitly request them.

2 Requirements

This JEP describes a protocol that is designed to fulfill the following requirements:

1. Determine the ability for clients and servers to support and exchange collaborative data objects
2. Enable the exchange of structured data objects between clients
3. Define a protocol that supports the synchronization of structure data among participating clients. Currently the protocol is based on a strict synchronization model where users will receive every change made to a CDO. Future versions of the protocol may contain a lazy synchronization model that may relax the current approach.
4. Operate in both private and group chat

Future enhancements may provide the ability to:

- Query room for active CDOs
- Query room history - results in CDO events for playback (per CDO or for all CDOs)
- Register methods to a CDO
- Register synchronization preference with framework (strict, lazy) CDO
- Method invocation request to framework to execute and return response

3 Usage Model

Here is a high-level description of the flow between two clients exchanging information using the CDO protocol.

3.1 Strict Synchronization mode

Participants using strict synchronization will receive every change event to the CDOs in a room or private chat.

Imagine two chat clients, A and B, that wish to share structured data in a synchronized manner. Both clients want to ensure they have the most up-to-date information. The two endpoints can do so by collaborating on the data using the CDO protocol.

For example, let's assume the clients wish to coordinate a meeting over chat. The process may look like this:

1. Client A creates a new meeting that contains structured data such as the title of the meeting, participant e-mail list, start time, date, length, and location.
2. Client A sends the meeting information to client B over normal chat.
3. When client A adds, updates, or deletes an item on the meeting, the changes are reflected to Client B
4. Similarly, when client B adds, updates, or deletes an item on the meeting, the changes are reflected to Client A

The above example describes the main flow between two clients. However a more detailed description of the flow using the example above would look like the following below. In this example we also show the interaction with the server (server X). Server X is the IM server that both client A and B are connected to.

1. Client A wishes to send information about a new meeting to client B
2. Client A first sends an IQ packet to client B to determine if the client supports the CDO protocol.
3. Client B responds to the IQ packet from client A confirming that it does support the CDO protocol.
4. Client A creates a new meeting that contains structured data such as the title of the meeting and the location.
5. When client A sends the CDO to client B, the message is intercepted by server X.

6. Server X, examines the message from client A and determines that it is a CDO create message. The server increments the version of the meeting items and forwards the message to client B.
7. When client A adds, updates, or deletes an item on the meeting, the changes are first interrogated by server X and then forwarded to Client B
8. Similarly, When client B adds, updates, or deletes an item on the meeting, the changes are first interrogated by server X and then forwarded to Client A
9. After processing the message from Client A, Server X will return a receipt to Client A. The receipt contains the UUID for the new item along with confirming the event was valid.

3.2 Lazy Synchronization mode

Lazy synchronization is a proposed alternate synchronization scheme that is appropriate for entities who either do not wish, or are not able because of infrastructure limitations, to receive the CDO stanzas in real time and wish to explicitly request them. Under lazy synchronization, entities are only notified that CDOs have been created, retired or that the existing CDOs that they keep track of have been updated and are now outdated. The details behind these events are not transmitted. Explicit action is required to resynchronized to the current state for any specific CDO identifier or change to the strict synchronization scheme. Lazy synchronization is set per endpoint (groupchat room).

4 Description Language

In topic-focused collaboration, a group of participants come together to discuss particular categories of information. This use of a topic as a focal point is significant because it provides participants with meaningful, often unspoken, context of information. Semantic cues, manipulation capabilities, state transitions, and information presentation are all included in this context.

CDOs leverage topic-focused collaboration by requiring every CDO to be an instance of a type. A type is analogous to a topic because participants are given a common understanding of the information being discussed and its context. This relationship is also similar to the programming concept of a class and an object. Much like a class describes the capabilities of an object, a CDO type describes an instance.

CDO Description Language (CDO-DL) is the means by which a type is defined. It is meant to be a highly extensible framework through which multiple (but equivalent) definitions of type capabilities can exist. This extensibility allows users with different operating environments to collaborate consistently.

Some information is required for every CDO-DL to maintain consistent interpretation:

- **UUID:** A universally unique identifier to distinguish this type from all others. This value is meant for machine use, but it may also be structured for human use.
- **Label:** The primary identifier meant for human use. This should concisely describe the data and context of a type. Although uniqueness of this field is not required, it is highly recommended.
- **Version:** The version associated with the type. As a type matures, its UUID and version will change. This also provides a mechanism for determining deprecation of previous versions.
- **Description:** A more complete description of the data and context of a type. A description can exist for individual languages.
- **Schema:** The XML Schema definition of the data associated with a type. This can either be embedded in the CDO-DL or referenced in an external URL.

Beyond this core data, every CDO-DL can have multiple (but equivalent) standard-specific implementations for the remaining context of the type:

- **Layout:** The means by which data is presented to a user. Layouts are arranged in groups, each with a unique identifier and title. Each group is capable of containing multiple equivalent standard-specific presentations to support a diverse client base, but each group must contain at least an XHTML/XForm presentation. Individual presentations can be embedded or referenced in an external URL, and one layout group must be marked as the default layout through which users will initially interact with the data.
- **Method:** The operations which can be carried out on the data. Equivalent standard-specific operations can be grouped together as a single notional method with a single unique identifier and description.
- **State:** The finite states through which a data set can transition. States are individually defined along with the criteria for entering each. State transitions are also defined along with the actions they perform (such as turning on or off specific layouts and methods).

5 Protocol

The CDO protocol uses the namespace: <http://www.xmpp.org/extensions/xep-0204.html#ns-data-sync/> is the root element and MUST be contained within a message stanza `<message/>` element. When the `<data-sync>` element is present in the `<message>`, the message MUST not contain a `<body>` element.

Listing 1: Message with data-sync element

```
<message to='romeo@example.net/orchard'
```

```

        from='juliet@example.com/balcony'
        type='chat' xml:lang='en'>
<data-sync protocol="1.0" uuid="" type="cdo:Meeting"
            event="create" xmlns="http://www.xmpp.org/
            extensions/xep-0204.html#ns">
  <item type="field" uuid="" event="create" ref="/Meeting/Title"
        version="0">
    <value>This is a new meeting</value>
  </item>
</data-sync>
</message>

```

The <data-sync> element may contain <item> elements. The attributes of the <data-sync> element maintain high-level information about the enclosed <item> elements to help synchronize information exchanged between clients.

Attribute	Description	Mandatory?
protocol	The version of the CDO exchange protocol	Yes
uuid	A universal unique identification number	Yes
packetID	a unique number created by the client to identify the cdo request, referenced by the framework in responses to a client (errors, responses to events). The packetID of the receipt MUST match the original data-synch message from the sender.	Yes
type	the uuid of the CDO definition (CDO-DL)	mandatory on create event and info event, otherwise forbidden
event	cdo event type: create, update, retire, info	Yes

Attribute	Description	Mandatory?
uuid	assigned ID of the field instance at creation time	Yes
type	distinguishes the type of field (allowed children may vary)	No
ref	xpath reference to the element	mandatory on create event and info event, otherwise forbidden
event	identified action on the item: create, update, delete, and info with update as default	Yes
version	integer version number of item being changed	Yes
updateStyle	indicates inclusive or exclusive update style with exclusive as the default	No

Each <item> may contain:

- 0 or 1 (0..1) <value> elements with content corresponding to the change to the identified CDO instance element contents

Listing 2: Value elements

```
<item type="field" uuid="" event="create" ref="/Meeting/Title" version="0">
  <value>This is a new meeting</value>
</item>
```

- 0 or unbounded (0..*) <attribute> elements. The name attribute corresponds to the name of the attribute in the CDO-DL. Note: All attributes are sent as part of an item, even those not changed when using the updateStyle exclusive.

Listing 3: Attribute elements

```
<item type="field" uuid="" event="create" ref="/Meeting/Time/Start" version="0">
  <attribute name="date">28 May 2006</attribute>
</item>
```

6 Use Cases

6.1 Query a client to determine if it supports a CDO message

Client A should send an IQ packet to another client (client B) before exchanging CDO messages to determine if client B supports CDO processing. This is accomplished using Service Discovery as described in [Service Discovery](#)¹.

NOTE: This assumes we register our information in the Jabber registrar. Here is an example of the exchange between two clients: See Service Discovery for specific packet requirements

Listing 4: Client A construct an IQ packet and sends it to Client B

```
<iq type='get' from='joe@mitre.org/Desktop' to='bob.mitre.org/Laptop'>
  <query xmlns='http://jabber.org/protocol/disco#info' />
</iq>
```

Listing 5: Client B receives the packet and responds with the following

```
<iq type='result'
  from='bob.mitre.org/Laptop'
  to='joe@mitre.org/Desktop'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
      category='cdo'
      type='text'
      name='Collborative_Data_Objects' />
    <feature var='http://www.xmpp.org/extensions/xep-0204.html#ns' />
    <feature var='jabber:iq:register' />
    <feature var='jabber:iq:search' />
    <feature var='jabber:iq:time' />
    <feature var='jabber:iq:version' />
  </query>
</iq>
```

Listing 6: When client A receives the response above, it will look for the value of the var attribute in feature

```
<feature var='http://www.xmpp.org/extensions/xep-0204.html#ns' />
```

Listing 7: If client A finds the value

```
http://www.xmpp.org/extensions/xep-0204.html#ns
```

it can assume the Client B supports the CDO protocol.

¹XEP-0030: Service Discovery <<http://xmpp.org/extensions/xep-0030.html>>.

6.2 Query the Server to determine available CDO types

A CDO client can determine what types of CDOs are available by querying the server

Listing 8: Client A constructs an IQ get and sends it to Server X

```
<iq type='get' from='joe@mitre.org/Desktop' id='cdo_list_1'>
  <<query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-types'
    />
</iq>
```

Listing 9: Server X responds with an IQ result listing the available CDO types

```
<iq type='result' to='joe@mitre.org/Desktop' id='cdo_list_1'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-types'>
    <item id='1023'>
      <name>Meeting CDO</name>
      <description>Describes a meeting</description>
    </item>
    <item id='2001'>
      <name>Trouble ticket</name>
      <description>Describes a Trouble shooting ticket</description>
    </item>
  </query>
</iq>
```

Listing 10: Client A selects the CDO type it would like to download by the item id

```
<iq type='get' from='joe@mitre.org/Desktop' id='cdo_list_2'>
  <<query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-types'
    >
    <<<<item_id='1023' />
  </query>
</iq>
```

Listing 11: Server X responds with the selected CDO type. Returning the CDO description language needed by the client

```
<iq type='result' to='joe@mitre.org/Desktop' id='cdo_list_2'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-types'>
    <cdo-dl>
      ...Actual CDO-DL definition
    </cdo-dl>
  </query>
</iq>
```

6.3 Query the Server to determine the state of a specific CDO

A CDO client can query the server to determine the specific state of a particular CDO:

Listing 12: Client A constructs an IQ get and sends it to Server X wishing to obtain the current state of an existing CDO

```
<iq type='get' from='joe@mitre.org/Desktop' id='cdo_state_1'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-state' />
  <cdo_uuid='ly8qoxl6r0rk42faell48a' />
</query>
</iq>
```

Listing 13: Server X responds with an IQ result showing the current state of the selected CDO.

```
<iq type='result' to='joe@mitre.org/Desktop' id='cdo_state_1'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-state'>
    <data-sync protocol="1.0"
      uuid="ly8qoxl6r0rk42faell48a"
      type="cdo:Meeting"
      packetID="0001"
      event="info"
      xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
      <item type="field"
        uuid="kej3n4kd"
        event="info"
        ref="/Meeting/Attendees"
        version="2">
        <value>Bob, Jim, Mike, Added Dave</value>
      </item>
    </data-sync>
  </query>
</iq>
```

Note an event type of info. This requires all attributes for the data-sync element and items be present

6.4 Query an endpoint for a list of CDOs

In some cases a client may be interested in the state of all CDOs belonging to a specific endpoint - for example a chat room.

Listing 14: Client A constructs an IQ get and sends it to Server X wishing to obtain the current state of all CDOs belonging to the CDO users chatroom

```
<iq type='get' from='joe@mitre.org/Desktop' to='cdo_users@mitre.org'
  id='cdo_state_2'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-state'>
```

```

<query cdo_uid='*' />
</query>
</iq>

```

Listing 15: Server X responds with an IQ result listing all current CDOs belonging to the CDO users chatroom

```

<iq type='result' to='joe@mitre.org/Desktop' id='cdo_state_2'>
  <query xmlns='http://www.xmpp.org/extensions/xep-0204.html#ns-state'>
    <data-sync protocol="1.0"
      uuid="ly8qoxl6r0rk42faell48a"
      type="cdo:Meeting"
      event="info"
      xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns"/>
    <data-sync protocol="1.0"
      uuid="9sdfs454jh5"
      type="cdo:Location"
      event="info"
      xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns"/>
    ...
  </query>
</iq>

```

Note the value of the uuid for the cdo tag above uses an asterisk (*) to indicate all CDOs if desired, Client A can then query the Server for the details on the state of a specific CDO using the protocol described earlier.

6.5 Create a new CDO

When two endpoints (client A and client B) wish to create a new CDO. Each client and server MUST follow this algorithm:

Listing 16: Client A creates a new CDO and sends it to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid=""
    type="cdo:Meeting"
    packetID="0001"
    event="create"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field" uuid="" event="create" ref="/Meeting/
      Title" version="0">

```

```

        <value>Technical Exchange Meeting</value>
      </item>
    </data-sync>
  </message>

```

Here client A, has created a packetID and set event=create in both <data-sync> and <item>. The version MUST be set to zero Next, the message is intercepted by Server X. Server X is the IM server that both clients are connected to. When the server receives the message, it MUST increment the version number and assign a uuid for both the <data-sync> and <item>. Once processed Server X MUST send a copy of the message back to client A as a receipt that the message has been processed by Server X and forward the message to client B

Listing 17: Send receipt from Server X to client A

```

<message
  to='joe@mitre.org/Desktop'
  from='joe@mitre.org/Desktop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0001"
    event="create"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="xdF10939"
      event="create"
      ref="/Meeting/Title"
      version="1">
      <value>Technical Exchange Meeting</value>
    </item>
  </data-sync>
</message>

```

Listing 18: Forward the processed message from Server X to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0001"
    event="create"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"

```

```

        uuid="xdF10939"
        event="create"
        ref="/Meeting/Title"
        version="1">
        <value>Technical Exchange Meeting</value>
    </item>
</data-sync>
</message>

```

Once this is completed, both clients have a synchronized CDO message with a uuid and a version number assigned by the server.

6.6 Create a new Item on an existing CDO

When client A wishes to create a new Item on an existing CDO, he sends the following information to client B

Listing 19: Client A creates a new item on an existing CDO and sends it to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0002"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field" uuid="" event="create" ref="/Meeting/
      Title" version="0">
      <value>Meeting</value>
    </item>
  </data-sync>
</message>

```

Here client A, has set event=update on the <data-sync> and event=create on the new <item>. The version MUST be set to zero on the new item. The message is intercepted by Server X. Server X is the IM server that both clients are connected to. When the server receives the message, it MUST increment the version number and assign a uuid for the new <item>. Next, Server X MUST send a copy of the message back to client A as a receipt that the message has been processed by Server X and forward the message to client B

Listing 20: Send receipt from Server X to client A

```

<message
  to='joe@mitre.org/Desktop'
  from='joe@mitre.org/Desktop'

```

```

    type=' chat '
    xml:lang=' en '>
<data-sync protocol="1.0"
            uuid="ly8qoxl6r0rk42faell48a"
            type="cdo:Meeting"
            packetID="0002"
            event="update"
            xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  <item type="field" uuid="kej3n4kd" event="create" ref="/
    Meeting/Title/" version="1">
    <value>Meeting</value>
  </item>
</data-sync>
</message>

```

Listing 21: Forward the message from Server X to client B

```

<message
  to=' joe@mitre.org/Desktop '
  from=' bob@mitre.org/Laptop '
  type=' chat '
  xml:lang=' en '>
<data-sync protocol="1.0"
            uuid="ly8qoxl6r0rk42faell48a"
            type="cdo:Meeting"
            packetID="0002"
            event="update"
            xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  <item type="field" uuid="kej3n4kd" event="create" ref="/Meeting/
    Title/" version="1">
    <value>Meeting</value>
  </item>
</data-sync>
</message>

```

Once this is completed, both clients have a synchronized CDO message with a new item. The new item has been assigned a uuid and a version number by the server.

6.7 Update an Item on an existing CDO

When client A wishes to update an item on an existing CDO, he sends the following information to client B

Listing 22: Client A updates an item on an existing CDO and sends it to client B

```

<message
  to=' joe@mitre.org/Desktop '
  from=' bob@mitre.org/Laptop '
  type=' chat '
  xml:lang=' en '>

```

```

<data-sync protocol="1.0"
  uuid="ly8qoxl6r0rk42faell48a"
  type="cdo:Meeting"
  packetID="0003"
  event="update"
  xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  <item type="field"
    uuid="kej3n4kd"
    event="update"
    version="1">
    <value>2006-07-24T14:55:00</value>
  </item>
</data-sync>
</message>

```

Here client A, has set event=update on the <data-sync> and event=update on the changed <item>. Note the version is set to the current version being edited, in this case 1, and both the <item> and <data-sync> have an existing uuid.

The message is intercepted by Server X. Server X is the IM server that both clients are connected to. When the server receives the message, it MUST increment the version number and assign a uuid for the new <item>

Next, Server X MUST send a copy of the message back to client A as a receipt that the message has been processed by Server X and forward the message to client B

Listing 23: Send receipt from Server X to client A

```

<message
  to='joe@mitre.org/Desktop'
  from='joe@mitre.org/Desktop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0003"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="kej3n4kd"
      event="update"
      version="2">
      <value>2006-07-24T14:55:00</value>
    </item>
  </data-sync>
</message>

```

Listing 24: Forward the message from Server X to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0003"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="kej3n4kd"
      event="update"
      version="2">
      <value>2006-07-24T14:55:00</value>
    </item>
  </data-sync>
</message>

```

Once this is completed, both clients have a synchronized CDO message with an updated item. The updated item has the version number incremented.

6.7.1 Update Styles

There are two types of behaviors associated with an item update: inclusive and exclusive. For an inclusive update, the content of the item element in the CDO data synchronization packet is considered to be fully representative of the value that item will have at the end of the update operation. This behavior results in previously set values for an item being destroyed if they are not repeated for each item update. For an exclusive update, the content of the item element in the CDO data synchronization packet is considered to contain only the values to be modified as a result of the update operation. This behavior results in previously set values for an item being carried over if they are not explicitly contradicted in each subsequent item update.

Assume that client A has created a CDO `cdo_1`. This CDO has an item `item_1` with the attribute named `attr_1` set. If A wants to update `item_1` with a value for the attribute named `attr_2` without destroying the previously set value for `attr_1`, the following data synchronization packets are equivalent:

```

<data-sync protocol="1.0" uuid="cdo_1" packetID="0003" event="update"
xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  <item type="field"
    uuid="item_1" event="update" version="1" updateStyle="inclusive">
    <attribute name="attr_1">old value</attribute>
    <attribute name="attr_2">new value</attribute>
  </item>
</data-sync>
<data-sync protocol="1.0" uuid="cdo_1" packetID="0003" event="update"
xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  <item type="field"
    uuid="item_1" event="update" version="1" updateStyle="exclusive">
    <attribute name="attr_2">new value</attribute>
  </item>
</data-sync>

```

6.8 Delete an Item on an existing CDO

When client A wishes to delete an item on an existing CDO, he sends the following information to client B

Listing 25: Client A deletes an item on an existing CDO and sends it to client B

```
<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0004"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="kej3n4kd"
      event="delete"
      version="2">
    </item>
  </data-sync>
</message>
```

Here client A, has set event=update on the <data-sync> and event=delete on the changed <item>. Note the version is set to the current version being deleted, in this case 2, and both the <item> and <data-sync> have an existing uuid.

The message is intercepted by Server X. Server X is the IM server that both clients are connected to. Server X MUST send a copy of the message back to client A as a receipt that the message has been processed by Server X and forward the message to client B

Listing 26: Send receipt from Server X to client A

```
<message
  to='joe@mitre.org/Desktop'
  from='joe@mitre.org/Desktop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0004"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="kej3n4kd"
      event="delete"
      version="2">
  </data-sync>
</message>
```

```

    </item>
  </data-sync>
</message>

```

Listing 27: Forward the message from Server X to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0004"
    event="update"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
    <item type="field"
      uuid="kej3n4kd"
      event="delete"
      version="2">
    </item>
  </data-sync>
</message>

```

Once this is completed, both clients have a synchronized CDO message with a deleted item.

6.9 Retire an existing CDO

Retired objects are different than deleted items in that retired objects can still be referenced and reviewed but no changes can be made.

When client A wishes to retire an entire existing CDO, he sends the following information to client B

Listing 28: Client A deletes an existing CDO and sends it to client B

```

<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0005"
    event="retire"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  </data-sync>
</message>

```

Here client A, has set event=retire on the <data-sync>.

The message is intercepted by Server X. Server X is the IM server that both clients are connected to. Server X MUST send a copy of the message back to client A as a receipt that the message has been processed by Server X and forward the message to client B:

Listing 29: Send receipt from Server X to client A

```
<message
  to='joe@mitre.org/Desktop'
  from='joe@mitre.org/Desktop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0005"
    event="retire"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  </data-sync>
</message>
```

Listing 30: Forward the message from Server X to client B

```
<message
  to='joe@mitre.org/Desktop'
  from='bob@mitre.org/Laptop'
  type='chat'
  xml:lang='en'>
  <data-sync protocol="1.0"
    uuid="ly8qoxl6r0rk42faell48a"
    type="cdo:Meeting"
    packetID="0005"
    event="retire"
    xmlns="http://www.xmpp.org/extensions/xep-0204.html#ns">
  </data-sync>
</message>
```

Once this is completed, both clients have retired (deleted) the CDO.

7 Errors

The error reporting mechanism for this specification follows the recommendations as set forth in the XMPP Core RFC. As such, when an error condition is detected, a stanza-related error must be generated and returned to sending entity. The error condition should be described using an XMPP general condition and CDO specific condition. Further amplification to the error condition should be provided by including the applicable subset of the data-sync message which caused the error.

Error types other than continue (cancel, modify, wait) disrupt the normal processing of CDO data-sync messages and only the associate error message will be issued by a supporting server.

Addition of only the applicable subset of the data-sync message in an error stanza is meant to assist the sending entity in isolating what directive in the message caused the error. Most CDO specific conditions include sufficient metadata to enable the sender to isolate the cause, but it is possible for the sender to generate messages in which this metadata is not present. In this case, it may not be possible for the sender to determine the message subset causing the error unless that subset is included in the error.

Listing 31: Assume the following message is sent:

```
<message from="miles@example.com" to="aral@example.com">
  <cdo:data-sync
    protocol="1.0"
    packetID="miles@example.com:packet-1"
    event="update"
    uuid="miles@example.com:instance-1">
    <item type="field" event="create" ref="/Meeting/Title"
      >
      <value>Winterfair preparation</value>
    </item>
    <item type="field" event="update"
      uuid="miles@example.com:field-1" version="2">
      <value>Imperial waltz</value>
    </item>
  </cdo:data-sync>
</message>
```

The first potential cause for an error in a data synchronization packet is the root data-sync element. An error at this level is independent of the content of any element descendants of the data-sync element. When this occurs, a server error sent to the recipient MUST include the data-sync element to provide context; a bandwidth-constrained connection MAY omit the element descendants.

```
<message to="aral@example.com" type="error">
  <cdo:data-sync
    protocol="1.0"
    packetID="miles@example.com:packet-1"
    event="update"
    uuid="miles@example.com:instance-1"/>
  <error type="cancel">
    <item-not-found/>
    <cdo:no-such-instance/>
  </error>
</message>
```

The second potential cause for an error is a child item element. For an item of event type create, the sender is not required to nominate a UUID for the item. If multiple items are used in this fashion, it is not possible for the item related to the error to be identified by reference. The only way to indicate to the sender the item related to the error is by isolating that item in the error stanza. As a result, the subset for this case is the data-sync element with all child item elements removed except the item related to the error.

```
<message to="aral@example.com" type="error">
  <cdo:data-sync
    protocol="1.0"
    packetID="miles@example.com:packet-1"
    event="update"
    uuid="miles@example.com:instance-1">
    <item type="field" event="update"
      uuid="miles@example.com:field-1" version="2">
      <value>Imperial waltz</value>
    </item>
  </cdo:data-sync>
  <error type="cancel">
    <item-not-found/>
    <cdo:no-such-item/>
  </error>
</message>
```


7.1 CDO specific conditions

Error Type	General Condition	Specific Condition	Description
cancel	<xmpp:feature-not-implemented/>	<cdo:unkown-protocol-version/>	The protocol version for the data sync packet is unknown to the framework and cannot be processed.
continue	<xmpp:undefined-condition/>	<cdo:deprecated-protocol-version framework-protocol-version="P"/>	The protocol version for the data sync packet is older than that used by the framework, but the packet can still be processed.
cancel	<xmpp:feature-not-implemented/>	<cdo:deprecated-protocol-version framework-protocol-version="P"/>	The protocol version for the data sync packet is older than that used by the framework, and the framework cannot process the packet.
continue	<xmpp:undefined-condition/>	<cdo:instance-identifier-conflict new-identifier="I"/>	The framework cannot create a new instance with the specified identifier because one already exists with that identifier. As a result, the framework has created a new identifier for the instance to be created.
cancel	<xmpp:item-not-found/>	<cdo:no-such-instance/>	The framework has no record of an instance with the specified identifier.
cancel	<xmpp:not-allowed/>	<cdo:instance-retired/>	The event cannot be executed on the instance because that instance has been retired.
wait	<xmpp:undefined-condition/>	<cdo:instance-is-active/>	The retire event cannot be executed on the instance because the instance is actively being updated.
cancel	<xmpp:item-not-found/>	<cdo:no-such-type/>	The framework has no record of a type with the specified identifier.
continue	<xmpp:undefined-condition/>	<cdo:deprecated-type latest-type="type-identifier"/>	The framework will create a new instance with the specified type, but one or more types have been registered as superseding this type.
continue	<xmpp:undefined-condition/>	<cdo:item-identifier-conflict old-identifier="I0" new-identifier="I1"/>	The framework cannot create a new item with the specified identifier because one already ex-

Errors which may result from schema validation have been omitted.

7.2 Invalid Constraint condition types

The CDO specific condition "invalid-constraint" enables the server to indicate logical invalidity of a data-sync packet. The use of a single condition for this purpose provides a consistent reporting mechanism to the sender, but the invalidity must be described with the "type" attribute as follows:

Type	Description
instance-identifier-required	Data-sync packets of event type "update" and "retire" MUST specify the target instance.
instance-type-prohibited	Data-sync packets of event type "update" and "retire" MUST NOT specify an instance type.
instance-type-required	Data-sync packets of event type "create" MUST specify an instance type.
item-required	Data-sync packets of event type "update" MUST include at least one data-sync item.
items-prohibited	Data-sync packets of event type "retire" MUST NOT include any data-sync items.
item-event-prohibited	Data-sync packets of event type "create" MUST ONLY include data-sync items of event type "create."
item-identifier-required	Data-sync items of event type "update" and "delete" MUST include the target identifier.
item-update-style-prohibited	Data-sync items of event type "create" and "delete" MUST NOT specify an update style.
item-value-prohibited	Data-sync items of event type "delete" MUST NOT contain a value.
item-value-required	Data-sync items of event type "create" and "update" MUST contain a value.
item-version-prohibited	Data-sync items of event type "create" MUST NOT specify a version.
item-version-required	Data-sync items of event type "update" and "delete" MUST specify a version.
item-xpath-prohibited	Data-sync items of event type "update" and "delete" MUST NOT specify an XPath.
item-xpath-required	Data-sync items of event type "create" MUST specify an XPath.

7.3 Special Case: Outdated version error

It is possible for a notional conflict to exist between two clients attempting to update a CDO instance. In this case, two separate clients submit data sync packets containing at least one

overlapping item. When this occurs, the first packet to arrive at the framework is executed, but the error information reported back to the client of the second packet MAY be extended to describe the notional conflict. This additional error information is meant to facilitate collaboration between the clients.

The additional error information provide includes:

- The resource identifier of the client originating the data sync packet with which this error conflicts.
- The timestamp of when the originating data sync packet was executed.
- The item element of the originating data sync packet with which this error conflicts.

```
<cdo:item-version-outdated identifier='I' version='V'>
  <cdo:conflict resource-identifier='R' execution-timestamp='T'>
    <item identifier='I' version='V1' event='E'>
      <value>VALUE</value>
      <attribute name='N'>VALUE</attribute>
    </cdo:conflict>
  </cdo:item-version-outdated>
```

8 Implementation Notes

8.1 Service

To properly handle the protocol described in this JEP. A server side service will need to be able to process CDO packets and IQ messages. This means at a minimum, the service will need to be able to:

1. Filter for data-sync and IQ protocol extensions as described above
2. Handle packet extensions that appear as a child element to the message element
3. Provide an underlying framework that can manage versioning of data-sync messages. If designed accordingly, it may be possible for this framework to be re-used by clients as well.

Additionally, a server MUST ignore any 'to' address on a cdo related IQ "set", and MUST treat any cdo IQ "set" as applying to the sender.

8.2 Client

For a client to be able to exchange CDO messages it will need to provide capabilities similar to those described above:

1. Filter and respond to data-sync and IQ protocol extensions described in this specification
2. Properly handle packet extensions that appear as a child elements to the message element
3. Provide an underlying framework that can manage the version of data-sync messages. (see 3. above)
4. Provide a form based GUI to properly construct the CDO messages.
5. Provide the ability to construct data-sync related IQ messages

Additionally, a client SHOULD check the "from" address of a cdo query (incoming IQ of type "result" containing a cdo type) to ensure that it is from a trusted source; specifically, the stanza MUST either have no 'from' attribute (i.e., implicitly from the server) or have a 'from' attribute whose value matches the user's bare JID (of the form <user@domain>) or full JID (of the form <user@domain/resource>); otherwise, the client SHOULD ignore the cdo query result.

9 Security Considerations

9.1 User Authentication and Authorization

No form of authentication or authorization is defined by this specification. However, if required, RFC 3920 describes channel encryption and strong authentication via TLS and SASL that may fulfill this requirement.

9.2 End-to-End Encryption

No end-to-end message or session encryption method defined in this specification. Users SHOULD NOT trust a service to exchange secret CDO messages.

10 IANA Considerations

This JEP requires no interaction with the Internet Assigned Numbers Authority (IANA).

11 XMPP Registrar Considerations

11.1 Protocol Namespace

Until this specification advances to a status of Draft, its associated namespace shall be "http://www.xmpp.org/extensions/xep-0204.html#ns" (along with relevant "sub-namespaces" in which "#ns" is followed by the "-" character and a subname string); upon

advancement of this specification, the XMPP Registrar shall issue a permanent namespace in accordance with the process defined in Section 4 of [XMPP Registrar Function](#)².

Note: As this protocol is currently used in implemented software, the namespaces are of the form "http://www.mitre.org/mtp/cdo", "http://www.mitre.org/mtp/cdo/types", etc.

12 XML Schema

12.1 Description Language

```
<?xml version='1.0' encoding='UTF-8'?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:cdodl="http://mitre.org/MTP/CDO-DL"
  targetNamespace="http://mitre.org/MTP/CDO-DL"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <xsd:element name="Definition" type="cdodl:DefinitionType"/>

  <xsd:complexType name="DefinitionType">
    <xsd:sequence>
      <xsd:element name="MetaData" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Label" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="Version" minOccurs="1" maxOccurs="1">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:pattern value="
                    [1-9][0-9]*\.[0-9]+\.[0-9]+\.[0-9]+"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
            <xsd:element name="Description" minOccurs="1" maxOccurs="
              1"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Type" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="xsd:schema" minOccurs="0" maxOccurs="1
              "/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

²XEP-0053: XMPP Registrar Function <<http://xmpp.org/extensions/xep-0053.html>>.

```

        <xsd:attribute name="rootElement" use="required" type="
            xsd:QName"/>
        <xsd:attribute name="schemaLocation" use="optional" type="
            xsd:anyURI"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="Layouts" minOccurs="1" maxOccurs="1">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Layout" minOccurs="1" maxOccurs="
                unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Description"
                            minOccurs="1"
                            maxOccurs="1"
                            type="xsd:string"/>
                        <any namespace="##other" minOccurs="1" maxOccurs="
                            unbounded"/>
                    </xsd:sequence>
                    <xsd:attribute name="default"
                        use="optional"
                        default="false"
                        type="xsd:boolean"/>
                    <xsd:attribute name="uuid" use="required" type="
                        xsd:string"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="Methods" minOccurs="1" maxOccurs="1">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Method" minOccurs="1" maxOccurs="
                unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Description"
                            minOccurs="1"
                            maxOccurs="1"
                            type="xsd:string"/>
                        <any namespace="##other" minOccurs="1" maxOccurs="
                            unbounded"/>
                    </xsd:sequence>
                    <xsd:attribute name="uuid" use="required" type="
                        xsd:string"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="States" minOccurs="1" maxOccurs="1">
    <xsd:complexType>
        <xsd:sequence>
            <any namespace="##other" minOccurs="1" maxOccurs="
                unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="uuid" use="required" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>

```

12.2 Data Synchronization Protocol

```

<?xml version='1.0' encoding='UTF-8'?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:cdo="http://www.xmpp.org/extensions/xep-0204.html#ns-data-
        sync"
    targetNamespace="http://www.xmpp.org/extensions/xep-0204.html#ns-
        data-sync"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified">
<xsd:element name="data-sync" type="cdo:DataSyncPacketType" />
<xsd:complexType name="DataSyncPacketType">
<xsd:sequence>
    <xsd:element name="item"
        type="cdo:DataSyncItemType"
        minOccurs="0"
        maxOccurs="unbounded" />
</xsd:sequence>
<xsd:attribute name="protocol" use="required">
<xsd:simpleType>
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[1-9][0-9]*\.[0-9]+" />
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="packetID" use="required" type="xsd:string" />
<xsd:attribute name="event" use="required">
<xsd:simpleType>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="create" />
        <xsd:enumeration value="update" />
        <xsd:enumeration value="retire" />
    </xsd:restriction>
</xsd:simpleType>

```

```

        <xsd:enumeration value="info" />
    </xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="uuid" use="optional" type="xsd:string" />
<xsd:attribute name="type" use="optional" type="xsd:string" />
</xsd:complexType>
<xsd:complexType name="DataSyncItemType">
    <xsd:sequence>
        <xsd:element name="value" type="xsd:string" minOccurs="0"
            maxOccurs="1" />
        <xsd:element name="attribute"
            type="cdo:DataSyncAttributeType"
            minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="type" use="optional" default="field">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="field" />
                <xsd:enumeration value="method" />
                <xsd:enumeration value="state" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="event" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="create" />
                <xsd:enumeration value="update" />
                <xsd:enumeration value="delete" />
                <xsd:enumeration value="info" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="updateStyle" use="optional" default="exclusive"
        >
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="inclusive" />
                <xsd:enumeration value="exclusive" />
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="uuid" use="optional" type="xsd:string" />
    <xsd:attribute name="ref" use="optional">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:annotation>

```

```
<xsd:documentation xml:lang="en">
  The pattern associated with this restriction
  is intended to model a simple XPath statement
  restricted only to element names. The included
  regular expression is not fully representative
  of all legal XML element names and should be
  extended.
</xsd:documentation>
</xsd:annotation>
<xsd:pattern value="/([a-zA-Z_][\w\.-_]*:)?[a-zA-Z_][\w\.-_]*" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="version" use="optional" type="
  xsd:positiveInteger" />
</xsd:complexType>
<xsd:complexType name="DataSyncAttributeType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="name" type="xsd:QName" use="required" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```