Infor Mongoose - Integrating IDOs with External Applications

Release 2020.xx

Important Notices

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Trademark Acknowledgements

Publication Information

Release: Infor Mongoose 2020.xx
Publication Date: January 24, 2020
Document code: mg_2020.xx_mgiiea_mg_en-us

Contents

About this guide

This guide describes what Mongoose IDOs (Intelligent Data Objects) are and how to use them to integrate with external applications.

The purpose of this guide

The purpose of this guide is to provide information for developers wanting to integrate Mongoose with non-Infor external applications.

What is in this guide

Information is provided on:

- The IDO request interface
- The IDO Request XML schema
- Using the Mongoose REST API, both versions 1 and 2
- Using the SOAP web service with Mongoose
- The Mongoose .NET client class library

Intended audience

The primary audience for this guide includes Mongoose form and application developers tasked with integrating non-Infor external applications to work with Mongoose. Secondary audiences include support, marketing, and administrative personnel tasked with supporting Mongoose application development.

Related documents

For more information about IDOs and related topics, see these additional sources:

For information about...	Look in...
...creating and editing IDOs and their methods and properties	...the online help for IDO editing

For information about...	Look in...
...the Application Event System	...the online help for application events
...IDO extension classes	...the *IDO Development Guide*
...customizing forms	...the Design Mode online help (**Core Development** in the help contents)
...licensing custom forms and IDOs	...the *Multi-Site Planning Guide* and the online help for licensing of forms
...performing diagnostics on IDO use; viewing logged data	...the IDO Runtime Development Server online help or Log Monitor online help, on the application server
...dealing with metadata for IDOs; synchronizing metadata from upgrades or add-on products; importing or exporting IDO-related metadata	... the App Metadata Transport utility online help …the FormSync online help
...defining configurations (named combinations of application, forms, and objects databases)	...the Configuration Manager online help

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

If we update this document after the product release, we will post the new version on the Infor Support Portal. To access documentation, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: About the IDO Request Interface

For Mongoose applications, the Intelligent Data Object (IDO) layer resides on the Application Server, sitting between the clients and the database. An IDO consists of references to tables in the Application database, definitions of properties, and definitions of specific methods.

IDO definitions are stored as metadata in an Objects database and are accessed through IDO editing forms. Through configurations, application databases are linked with an Objects database and a Forms database. Optionally the objects and forms databases are embedded in the application database.

Developers can create logic associated with IDOs using the Application Event System (AES) and can incorporate .NET IDO extension class code as well; but the Mongoose framework implements the IDO Request Interface for all IDOs. This interface consists of these operations, plus session and metadata operations:

- **Load Collection** generates and executes SQL to retrieve a collection of rows from the database.
- **Update Collection** takes a set of rows marked for insert, update, or delete, and generates and executes the appropriate SQL code against the database.
- **Invoke** allows you to execute a specific method defined for the IDO.

Each of the operations in the IDO Request Interface consists of a request and a response. The caller builds an IDO request and sends it to the IDO runtime engine to be executed. The IDO runtime engine builds and returns a response to the caller containing the results of the requested action.

The Mongoose UI runs entirely by means of the IDO Request Interface, using the IDO Request XML schema described elsewhere in this document, using HTTP(S).

Integrating to IDOs involves understanding this IDO Request Interface and leveraging it using the variety of technologies provided and described in this document (REST, SOAP, .NET class library, XML/HTTP). The REST version of the IDO Request Interface is also provided in Infor's ION API repository.

For those wishing to understand integrating with IDOs, understanding the IDO Request XML schema is the first step, as that is the native Mongoose interface. The others all map to/from that interface.

The IDO Runtime Service

The application server for Mongoose is the IDO Runtime, which is usually run as a service, though for development, it might also be run as an executable file that has its own UI, named the IDO Runtime Development Server (IDORuntimeHost.exe).

This service has a front-end in IIS and is the code that implements the IDO Request Interface.

Integrating with IDOs involves establishing communications to the IDO Runtime by means of a URL, using one of the technologies described in the next section.

Interfacing options

Mongoose provides these technology options for accessing the IDO Request Interface:

- The IDO Request XML schema, using HTTP(S)
- REST, including ION API
- SOAP
- The Mongoose .NET client class library

Each of these options are explained in the subsequent chapters.

Permissions and licensing for the IDO Request Interface

Sessions established with the Mongoose UI involve a different model for permissions and licensing than sessions established programmatically using the IDO Request Interface.

For UI sessions, the user account must be granted authorizations, or permissions, for each form the user attempts to run. This is granted at a group or user level in the **Groups** or **Users** form. This authorization can also include policy as to which operations are enabled, and even field-level permissions as well. Similarly, the users must be granted at least one license module that contains the form they are trying to run, unless the Usage-Tracking-Only mode of licensing is in effect.

By contrast, for Isessions created through an API, the user account permissions and licensing are checked against the IDOs being accessed, not the forms. So, the user account for the session needs to have been granted permissions to the IDOs for which Load Collection, Update Collection, and Invoke requests are made. Those permissions might further restrict which operations that user account can use for various IDOs, as well as property-level restrictions. Finally, the user account must have been granted at least one license module which includes the IDO being accessed, unless the Usage-Tracking-Only mode of licensing is in effect.

Note: When the REST option is used within the ION API, the user account is drawn from the browser, and single-sign-on occurs. The resulting user account, however, still requires IDO-level permissions and licensing.

For details on configuring permissions and licensing, see the online help for the **Users** and **Groups** forms.

Chapter 2: The IDO Request XML Schema

The IDO Request XML Schema is the native Mongoose format for the IDO Request Interface. The Mongoose UI uses this transport format for all aspects of the execution of the user interface. The other options provided for the IDO Request Interface (REST, SOAP, .NET class library) are mappings to and from this protocol.

Understanding the IDO Request XML Schema is, therefore, helpful, even if you are leveraging other options, because it shows all the actual capabilities. You can “wiretap” a session in your application and see this format of the traffic between the client and application server. You can also use this format directly, by sending and receiving these XMLs to the Mongoose web server from custom client code.

The IDO Request XML headers

The IDO Request XMLs include requests and responses for the three key operations, as well as session logon and logout provided by the IDO Request Interface.

These are the IDO Request XML headers:

- OpenSession
- LoadCollection
- UpdateCollection
- Invoke
- CloseSession

OpenSession

An OpenSession request/response document validates the user’s identity, creates a new session in the application, and returns a session ID. This is essentially the same as the user logging into the application where the caller can communicate with the IDO.

You must submit an OpenSession request before you can submit other requests.

Request data

An OpenSession request document has this payload structure that contains data, such as the user's login information (user ID, password, configuration name). These data are required to complete the request.

```
<RequestData>
   <UserID>userid</UserID>
   <LanguageID>culture</LanguageID>
   <ConfigName>config</ConfigName>
   <MachineName>machinename</MachineName>
   <DomainUserName>domainusername</DomainUserName>
   <ApplicationName>appname</ApplicationName>
   <AllowCloseExistingSessions>true|false</AllowCloseExistingSessions>
   <Password Encrypted="N">password</Password>
   <Workstation>workstation</Workstation>
</RequestData>
```

Response data

An OpenSession response document has this payload structure and includes requested data and return values.

```
<ResponseData>
   <UserID>userid</UserID>
   <LanguageID>culture</LanguageID>
   <ProductVersion>version</ProductVersion>
   <DeadlockRetry>0</DeadlockRetry>
   <License Status="status">
     <Message>message</Message>
   </License>
   <RegionalSettings MessageLanguageID="langID" LocaleID="localeID" Deci
malSeparator="dec-sep" DigitGroupSeparator="dig-sep" DigitsInGroup="dig">re
gional-settings</RegionalSettings>
  <AdditionalFailureInformation>failureinfo</AdditionalFailureInformation>

   <ServerDate>serverdatetime</ServerDate>
   <LoginResult>loginstatus</LoginResult>
   <PrimaryGroupName>group</PrimaryGroupName>
   <DaysUntilPasswordExpires>days</DaysUntilPasswordExpires>
   <EditLevel>editlevel</EditLevel>
   <SuperUser>su-indicator</SuperUser>
   <StartupMethods>startup-method-spec</StartupMethods>
   <AuditingEnabled>true|false</AuditingEnabled>
</ResponseData>
```

Element descriptions

Element	Description
AdditionalFailureInformation	If the login fails, this element can contain additional information about why.

Element	Description
AllowCloseExistingSessions	Optional. If this is **True**, existing sessions can be closed if the concurrent login limit prevents user from logging in. Default = **False**
ApplicationName	Optional. This is the name of the application being used for the session. This element is used primarily for diagnostics.
AuditingEnabled	This element is for internal use only.
ConfigNamne	This is the name of the configuration where the session runs. You can get a list of configurations by using this in the request.
DaysUntilPasswordExpires	This is the number of days until the specified user's current password expires.
DeadlockRetry	This is no longer used.
DomainUserName	Optional. This is the client's NT user name. This element is used primarily for diagnostics.
EditLevel	This is the level of editing permission assigned to the user, as specified on the **Users** form: • 0 = None • 1 = Basic • 2 = Full User • 3 = Site Developer • 4 = Vendor Developer For information about the particulars of these editing/permission levels, see the online help topic, "Editing permissions".
LanguageID	Optional. This is the culture value being used for the session. This corresponds to a value in the LanguageIDs table. Default = **en-US**
License	If the login is successful, this indicates the status of the application database's license (VALID or INVALID). If INVALID, the message contains additional info, such as when the license expired.

Element	Description
LoginResult	This is the result of the specified user's login attempt. These are the possible values: • Success • InvalidCredentials • AccountDisabled • AccountLocked • PasswordExpired • PasswordWillExpire • SessionLimit • ConcurrentSessionLimit • InvalidConfiguration • UnknownFailure **Note:** This response should never present. • LicenseInconsistency **Note:** This response indicates that someone has tampered with the licensing.
MachineName	Optional. This is the name of the computer being used for the session. This element is used primarily for diagnostics.
Password	This is the user's password. If **Encrypted** = **Y**, the password is encrypted. However, **Encrypted** is usually set to **N**.
PrimaryGroupName	This is the name of the primary group defined for the user as specified on the **Users** form.
ProductVersion	This is the version of the application.
RegionalSettings	This is the regional settings and locale settings used for this session. This information comes from the LanguageIDs table.
ServerDate	This is the date on the Application server. The CURDATE() and CURTIME() keywords must yield values that match the server for the application, not the client machine. This value is used to compute the difference between the client machine's date and time and the Application server's date and time. This is done so that each time CURDATE() and CURTIME() are evaluated, Mongoose can produce the correct server-based result.
StartupMethods	This element is for internal use only.

Element	Description
SuperUser	This element indicates whether the specified user is a "super user". • 1 = Yes • 0 = No For more information about super users, see the online help topic, "Super User".
UserID	This is the user's ID, which must be defined for the specified configuration.
Workstation	Optional. This element is for internal use only.

Examples for OpenSession

This example request opens a session using the configuration MG_DEV for a user whose ID is **jdelacruz**.

```
<IDORequest ProtocolVersion="6.03" SessionID="">
   <RequestHeader Type="OpenSession">
      <InitiatorType />
      <InitiatorName />
      <SourceName />
      <SourceConfig />
      <TargetName />
      <TargetConfig />
      <RequestData>
        <UserID>jdelacruz</UserID>
        <LanguageID />
        <PrefsLanguageID />
        <ConfigName>MG_DEV</ConfigName>
        <AllowCloseExistingSessions>true</AllowCloseExistingSessions>
        <Password Encrypted="N">Passwe1rd</Password>
        <Workstation />
        <Passcode />
        <TrustedClient>false</TrustedClient>
      </RequestData>
   </RequestHeader>
</IDORequest>
```

This example response contains the session ID which can be used for submitting other requests.

```
<IDOResponse ProtocolVersion="6.03" SessionID="04269518-2d4b-4b13-938b-
65ed2f561c53">
   <ResponseHeader Type="OpenSession">
      <InitiatorType />
      <InitiatorName />
      <SourceName />
      <SourceConfig />
      <TargetName />
```

```
      <TargetConfig />
      <ResponseData>
         <UserID>sa</UserID>
         <LanguageID />
         <PrefsLanguageID />
         <ProductVersion>10.08.00</ProductVersion>
         <PendoGuidesAPIKey />
         <FormsVersion />
         <License Status="VALID">
            <Message />
            <BusinessLogicalID>default</BusinessLogicalID>
         </License>
         <RegionalSettings MessageLanguageID="1033" LocaleID="1033" Deci
malSeparator="." DigitGroupSeparator="," DigitsInGroup="3">
            <StringTableName>Strings</StringTableName>
            <LanguageSubDir>en-US</LanguageSubDir>
            <HelpSubDir>en-US</HelpSubDir>
            <LanguageCode>ENU</LanguageCode>
         </RegionalSettings>
         <LastChangeDateIDOMetadata>0x00000000000278d0</LastChangeDateI
DOMetadata>
         <LastChangeDateFormsMetadata />
         <AdditionalFailureInformation />
         <ClientRetryMinutes>0</ClientRetryMinutes>
         <ServerDate>20190813 10:12:17.203</ServerDate>
         <LoginResult>Success</LoginResult>
         <PrimaryGroupName />
         <DaysUntilPasswordExpires>2147483647</DaysUntilPasswordExpires>
         <UserDesc />
         <TokenAuthenticationMessage />
         <HostEnvironment>InforPAAS</HostEnvironment>
         <EditLevel>4</EditLevel>
         <SuperUser>1</SuperUser>
         <UserLanguageID />
       <StartupMethods>AccessAs.StartupMethodSp(RVAR V(Parm_Site),RVAR
V(Parm_DisplayReportHeaders),RVAR V(AccessAs),RVAR V(Parm_DefaultStarting
ToEnding))</StartupMethods>
       <AuditingEnabled>false</AuditingEnabled>
     </ResponseData>
   </ResponseHeader>
</IDOResponse>
```

LoadCollection

A LoadCollection request/response document uses the LoadCollection method of an IDO to query either an IDO collection or a database table and return the results to the user.

Request data

A LoadCollection request document has this payload structure and contains data such as the collection name, property list, and so on, which are required to complete the request.

```
<RequestData>
  <LoadCollection Name="collection" LoadCap="loadcap">
    <LoadType>loadtype</LoadType>
    <ReadMode>readmode</ReadMode>
    <Cursor />
    <PropertyList>
      <propertyname>property01</propertyname>
      <propertyname>property02</propertyname>
      <propertyname>property...</propertyname>
    </PropertyList>
    <RecordCap>recordcap</RecordCap>
    <Filter>filter</Filter>
    <OrderBy>orderby</OrderBy>
    <PostQueryCmd>command</PostQueryCmd>
  </LoadCollection>
</RequestData>
```

Resonse data

A LoadCollection response document has this payload structure and includes requested data and return values.

```
<ResponseData>
  <LoadCollection Name="collection" LoadCap="loadcap">
    <PropertyList>
      <propertyname>property01</propertyname>
      <propertyname>property02</propertyname>
      <propertyname>property...</propertyname>
    </PropertyList>
    <LoadType>loadtype</LoadType>
    <RecordCap>recordcap</RecordCap>
    <Filter />
    <OrderBy />
    <PostQueryCmd />
    <Items>
      <Item ID="itemid01">
        <P>propertyvalue</P>
        <P>propertyvalue</P>
        <P>propertyvalue</P>
      </Item>
      <Item ID="itemid02">
        <P>propertyvalue</P>
        <P>propertyvalue</P>
        <P>propertyvalue</P>
      </Item>
      <Item ID="itemid...">
        <P>propertyvalue</P>
        <P>propertyvalue</P>
```

```
        <P>propertyvalue</P>
      </Item>
    </Items>
    <FilterIDO />
    <MoreItems>moreitems</MoreItems>
  </LoadCollection>
</ResponseData>
```

Element descriptions

Element	Description
Cursor	The response data can include a cursor value that is a "bookmark" in the collection, to tell where the retrieval stopped. If you want to make multiple requests using NEXT or PREV to retrieve data, include a single <cursor /> element in your FIRST request. The response document provides a cursor value. Include the cursor value of the response document (as <cursor>value</cursor>) in the NEXT request document, so the system can tell where to start the next retrieval request. Continue updating the cursor value as you provide additional NEXT or PREV requests. A request of LAST should use an empty <cursor /> element.
Distinct	Retrieves a set of data representing only the distinct combinations of properties requested. For example: <LoadCollection Name="MGCore.Users"> <LoadType>FIRST</LoadType> <PropertyList> <UserID /> </PropertyList> <Distinct /> </LoadCollection>

Element	Description

`Filter`	This element restricts the result set and corresponds to the FILTER or FILTERPERM keyword in Design Mode. You can use simple expressions, such as property names, comparison operators, Boolean operators, and so on.
	Some examples:
	`<Filter>CoNum like 'ABC%'</Filter>`
	`<Filter>QtyOnHand < 20 AND Whse = 'OH'</Filter>`
	Not allowed are things like subqueries, function calls, or anything else that could be used for an injection attack.
	For cloud environments, filters are always validated in the cloud. For on-premises installations, you can disable validation using the Filter Validation process default.
	In the response document, this element contains any IDO-level filter specifications that were used.

`Item`	The **`ID`** attribute contains a value used within the system to identify the item.

`LinkBy`	This element is similar to the LINKBY keyword. The information within this element tells how the child collection is linked to the parent collection.
	For example:
	`<LinkBy>`
	`<PropertyPair Child="Item" Parent="Item" />`
	`<!-- Optional: More sets of PropertyPair tags -->`
	`</LinkBy>`

`LoadCollection`	This element has these possible attributes: • **Name** - This is the name of the collection to be loaded (for example, GroupNames) • **LoadCap** - This attribute applies only to subcollections. This is the maximum number of subcollections to include in the returns. These are the valid values for this attribute: • **`-1`** - Includes all subcollections • **`1`** - (Default) Only queries the subcollection records for the first parent item. • Any other number - Specifies the maximum number of subcollections to retrieve.

Element	Description
LoadType	This element specifies one of these types: • FIRST • NEXT • PREV • LAST The first time you submit a LoadCollection request, it must be called FIRST or LAST. For subsequent requests, you can use NEXT or PREV.

OrderBy	This element provides a comma-delimited list of properties that specify how the response document result set should be sorted. The DESC keyword can appear after a property name to sort that property descending.
	For example:
	`<LoadCollection Name="MGCore.Users">`
	`  <LoadType>FIRST</LoadType>`
	`  <PropertyList>`
	`  <UserID />`
	`  <Username />`
	`  <Location />`
	`  </PropertyList>`
	`  <OrderBy>UserID, Username</OrderBy>`
	`</LoadCollection>`

P (**P**roperty Value)	For each item, this list contains values corresponding the properties listed.

PropertyList	In request data, you only need to list the properties you want returned. Each property is used as an element. The response data mimics the request list.
	For example:
	`<PropertyList>`
	`  <Acct />`
	`  <AcctUnit1 />`
	`  <AcctUnit2 />`
	`  <AcctUnit3 />`
	`  <AcctUnit4 />`
	`  . . .`
	`</PropertyList>`

Element	Description
ReadMode	This element specifies the collection "read mode", which controls the isolation level used when executing queries. See the online help for "Process Defaults". These are the valid attributes for this element: • **ReadCommitted** • **ReadUncommitted** • **Default** If this element is empty or omitted, **Default** is assumed. If this element in the request is anything except **Default**, the read mode value appears in the response.
RecordCap	This element specifies how many records are to be retrieved in one request. These are the valid values: • `-1` - (Default) 200 records are to be retrieved by system default. For other possible system record caps, see the online help topic "About caps". • `0` - No cap, all records are to be retrieved. • Any other number - The specified number of records are to be retrieved.

PostQueryCmd	This element specifies a method to execute once for each row in the result set, after the query is completed. This is equivalent to the **PQ** option in Load/Save overrides and uses the same syntax.
	For example:
	`<PostQueryCmd>`
	`MyPostQuerySp( Property1, Property2, REF Property3 )`
	`</PostQueryCmd>`

Example 1 - Basic Load Collection

This example request queries the user ID, username, and user description from the users table.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-ed844ffe7129">
  <RequestHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <LoadCollection Name="UserNames" LoadCap="0">
        <PropertyList>
```

```
          <UserDesc />
          <UserId />
          <Username />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
      </LoadCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response contains user information such as user ID, username, and user description from the users table.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <LoadCollection Name="UserNames" LoadCap="0">
        <PropertyList>
          <UserDesc />
          <UserId />
          <Username />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-13 14:14:23.487]
 UserNames.ID=[919f93d7-2427-489a-b78b-839f37a757db]">
            <P>Will Smith</P>
            <P>38</P>
            <P>wsmiith</P>
          </Item>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-13 14:14:23.453]
 UserNames.ID=[122cb5af-c5fb-4109-a95b-a331cb4e258c]">
            <P>Juan Dela Cruz</P>
            <P>41</P>
            <P>jdelacruz</P>
          </Item>
        </Items>
        <FilterIDO />
```

```
        <MoreItems>false</MoreItems>
      </LoadCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Example 2 - Custom Load Collection

One variation of the LoadCollection request is to specify an IDO method for loading data into the collection. It includes these elements:

```
<CustomLoadMethod Name="methodname">
  <Parameters>
    <Parameter>parametervalue</Parameter>
    <Parameter>parametervalue</Parameter>
  </Parameters>
</CustomLoadMethod>
```

This example request queries the note content and description from the object notes table using the custom load method GetNotesSp.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <RequestHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <LoadCollection Name="ObjectNotes" LoadCap="0">
        <PropertyList>
          <SpcnNoteContent />
          <SpcnNoteDesc />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <ReadMode>ReadCommitted</ReadMode>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
        <CustomLoadMethod Name="GetNotesSp">
          <Parameters>
            <Parameter>UserNames</Parameter>
            <Parameter>4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c</Parameter>
          </Parameters>
        </CustomLoadMethod>
      </LoadCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response contains the note contents and description from the object notes table.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <LoadCollection Name="ObjectNotes" LoadCap="0">
        <PropertyList>
          <SpcnNoteContent />
          <SpcnNoteDesc />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
        <Items>
          <Item ID="">
            <P>Send in your weekly report every Friday before leaving the
 office</P>
            <P>Weekly Report</P>
          </Item>
        </Items>
        <MoreItems>false</MoreItems>
      </LoadCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Example 3 - Nested Load Collection

LoadCollection requests can be hierarchichal. For example, this sample request queries the users and user emails table in a single request, by nesting the UserEmails IDO LoadCollection request inside the UserNames IDO LoadCollection request.

```
<IDORequest ProtocolVersion="6.03" SessionID="71e1c28d-4e00-495e-bff0-
571b26179649">
  <RequestHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <LoadCollection Name="UserNames" LoadCap="0">
        <PropertyList>
```

```
          <UserDesc />
          <UserId />
          <Username />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
        <LoadCollection Name="UserEmails" LoadCap="0">
          <LinkBy>
            <PropertyPair Parent="UserId" Child="UserId" />
          </LinkBy>
          <PropertyList>
            <EmailAddress />
            <EmailType />
          </PropertyList>
          <LoadType>FIRST</LoadType>
          <RecordCap>-1</RecordCap>
          <Filter />
          <OrderBy />
          <PostQueryCmd />
        </LoadCollection>
      </LoadCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response contains a set of records retrieved from the users and user emails table.

```
<IDOResponse ProtocolVersion="6.03" SessionID="71e1c28d-4e00-495e-bff0-
571b26179649">
  <ResponseHeader Type="LoadCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <LoadCollection Name="UserNames" LoadCap="0">
        <PropertyList>
          <UserDesc />
          <UserId />
          <Username />
        </PropertyList>
        <LoadType>FIRST</LoadType>
        <RecordCap>-1</RecordCap>
        <Filter />
        <OrderBy />
        <PostQueryCmd />
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-13 14:14:23.487]
```

```
 UserNames.ID=[919f93d7-2427-489a-b78b-839f37a757db]">
            <P>Will Smith</P>
            <P>38</P>
            <P>wsmiith</P>
            <LoadCollection Name="UserEmails" LoadCap="0">
              <PropertyList>
                <EmailAddress />
                <EmailType />
              </PropertyList>
              <LoadType>FIRST</LoadType>
              <RecordCap>-1</RecordCap>
              <Filter />
              <OrderBy />
              <PostQueryCmd />
              <Items>
                <Item ID="PBT=[UserEmail] ue.ID=[2732c8a5-1400-4417-88a4-
28f6e8dcb9fd] ue.DT=[2019-08-13 14:12:57.393]">
                  <P>will.smith@infor.com</P>
                  <P>P</P>
                </Item>
                <Item ID="PBT=[UserEmail] ue.ID=[9916ae94-07ef-443b-8ae9-
a9ce4e24b4b9] ue.DT=[2019-08-13 14:14:23.513]">
                  <P>wsmith@business.com</P>
                  <P>S</P>
                </Item>
              </Items>
              <FilterIDO />
              <MoreItems>false</MoreItems>
            </LoadCollection>
          </Item>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-13 14:14:23.453]
 UserNames.ID=[122cb5af-c5fb-4109-a95b-a331cb4e258c]">
            <P />
            <P>41</P>
            <P>jdelacruz</P>
            <LoadCollection Name="UserEmails" LoadCap="0">
              <PropertyList>
                <EmailAddress />
                <EmailType />
              </PropertyList>
              <LoadType>FIRST</LoadType>
              <RecordCap>-1</RecordCap>
              <Filter />
              <OrderBy />
              <PostQueryCmd />
              <Items>
                <Item ID="PBT=[UserEmail] ue.ID=[28685ab7-c635-4e86-b465-
f21e6497a263] ue.DT=[2019-08-13 14:13:13.803]">
                  <P>juan.delacruz@infor.com</P>
                  <P>P</P>
                </Item>
              </Items>
              <FilterIDO />
              <MoreItems>false</MoreItems>
```

```
            </LoadCollection>
          </Item>
        </Items>
        <FilterIDO />
        <MoreItems>false</MoreItems>
      </LoadCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

UpdateCollection

An UpdateCollection request/response document modifies a collection (inserting, updating, or deleting records) using the UpdateCollection method of an IDO.

When performing an update or deletion request, identify the records you want to update or delete. There are two ways to do this:

- By item ID – To update or delete by item ID, you must perform a LoadCollection first to retrieve the item ID, and then include it in the `<Item>` node's ID attribute when performing the UpdateCollection update or delete operation.
- By the key data – In this case, specify the attribute `UseKeys="Y"` on the `<Item>` node and set the appropriate properties to the key data when performing the update or delete operation.

For update requests, you can also specify the locking option to be used during an IDO item update by using the UpdateLocking attribute on the `<Item>` node. There are two options for this:

- Row – (Default) This option extracts the row's item ID to get the Row Pointer and Record Date, and use them to construct the WHERE clause.
- Property – This option includes the row's key values and original property values to construct the WHERE clause. This attribute ensures that any intermediate change on the row is overwritten with the latest values. This property also enables simultaneous users to edit on the same row on different columns.

For insert requests, this information is not needed.

Request data

An UpdateCollection request document has this payload structure and contains data such as the collection name and the rows that are subjected to change, which are required to complete the request:

```
<RequestData>
  <UpdateCollection Name="collection" RefreshAfterUpdate="Y">
    <Items>
      <Item ID="itemid" ItemNo="0" Action="actiontype">
        <Property Name="propertyname">propertyvalue</Property>
        <Property Name="propertyname">propertyvalue</Property>
      </Item>
      <Item ID="itemid" ItemNo="0" Action="actiontype">
        <Property Name="propertyname">propertyvalue</Property>
        <Property Name="propertyname">propertyvalue</Property>
```

```
      </Item>
    </Items>
  </UpdateCollection>
</RequestData>
```

Response data

An UpdateCollection response document has this payload structure and includes requested data and return values.

```
<ResponseData>
  <UpdateCollection Name="collection" RefreshAfterUpdate="Y">
    <CollectionID />
    <Items>
      <Item ID="itemid" ItemNo="0" Action="actiontype">
        <Property Name="propertyname">propertyvalue</Property>
        <Property Name="propertyname">propertyvalue</Property>
      </Item>
      <Item ID="itemid" ItemNo="0" Action="actiontype">
        <Property Name="propertyname">propertyvalue</Property>
        <Property Name="propertyname">propertyvalue</Property>
      </Item>
    </Items>
  </UpdateCollection>
</ResponseData>
```

Element descriptions

Element	Description
CollectionID	Optional. This element is used when reporting errors that occur while processing items in an UpdateCollection request. If this element is omitted in the request, the IDO name is used instead for any error response. An UpdateCollection request can contain hierarchical (nested) requests. Having a collection ID allows the system response to include both the error message and which item caused the error. (See example 3.)

Element	Description

`Item`	This element can contain these attributes:
	• **Action** – This attribute can be Insert, Update, or Delete. For Insert operations, if the InitiatorType is set to **`Replication`**, the insert is treated as an update if the record already exists.
	• **ItemNo** – This number is user-defined and optional. It must be a valid 32-bit signed integer. If RefreshAfterUpdate is **`Y`** and this number is specified, the response includes the record number value, so that you can match updated records with the originals and see more easily what has changed.
	• **ID** – This value is used within the system to identify the item. When requesting an update or deletion of existing records, you must include either this, the item ID attribute, or the UseKeys attribute. Omitting the ID attribute defaults to the same behavior as **`UseKeys="Y"`**.
	• **UseKeys**:
	  • **`Y`** (the default value) indicates that the update or delete request is to use key data, rather than item IDs.
	  • **`N`** indicates that the request is to include item IDs.
	• **UpdateLocking**:
	  • **Row** (the default value) extracts the row's indentifying property (**ItemID**) to get the RowPointer and RecordDate, and use them to construct the WHERE clause.
	  • **Property** includes the row's key values and original values to construct the WHERE clause. RecordDate and Row-Pointer are omitted from the WHERE clause.

`Property`	Include in your request any properties that you plan to update, as well as all non-nullable properties that do not have a default value. You can omit properties that have default values that do not change because of the update. This element can contain these attributes: • **Name** – This is the name of the property. • **Modified** – This value specifies whether the property was updated. Only values flagged as `"Modified=Y"` are updated in the database. • **OriginalValue** – This attribute contains the original value of the property. Use this attribute when the `<Item>` node's `UpdateLocking="Property"`.

Element	Description

UpdateCollection	This element can include these attributes:
	• **Name** – This attibute contains the name of the IDO collection.
	• **RefreshAfterUpdate**:
	  • **Y** indicates that the response document should show information for inserted or updated items that was updated after the update was done. Deleted items are not updated.
	  • **N** indicates that the response document information was not updated after the modification.
	• **TxnScope** – Optional. This attibute can be set to **Collection** (the default value) or **Item**.
	When this is set to **Item**, each individual item in the UpdateCollection request is saved in a separate transaction. For a hierarchical (nested) UpdateCollection request, the value of the TxnScope attribute at the root level determines the behavior for the entire UpdateCollection request. If an exception occurs while processing an UpdateCollection request for an item using the TxnScope attribute, records that were saved before the exception are still committed, but no additional records are saved.
	If the TxnScope attribute is omitted or is set to **Collection**, the request is processed in a single transaction.
	This attribute has no effect when it is included in an inner UpdateCollection within hierarchical requests.

LinkBy	This element is valid only in CONTAINS relationships, much like the LINKBY keyword.
	The information within this pair of tags tells how the CONTAINS relationship is linked to the parent collection.
	For example:
	`<LinkBy>`
	`  <PropertyPair Child="Item" Parent="Item" />`
	`  <!-- Optional: more sets of PropertyPair tags -->`
	`</LinkBy>`

Example 1 - Using UpdateCollection for Insert, Update, and Delete operations

This example request shows how you can perform different actions for each row in the collection within a single request. This request 1) creates a new user, 2) updates the description of user **wsmith**, and 3) deletes the record of user **cdelune**.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-ed844ffe7129">
  <RequestHeader Type="UpdateCollection">
    <InitiatorType />
```

```
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-13 14:14:23.453]
 UserNames.ID=[122cb5af-c5fb-4109-a95b-a331cb4e258c]" ItemNo="0" Action="In
sert">
            <Property Name="UserDesc">John Doe</Property>
            <Property Name="Username">jdoe</Property>
          </Item>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 17:30:09.870]
 UserNames.ID=[b3aedc23-722f-4058-b12c-f14bdfd955b4]" ItemNo="0" Action="Up
date">
            <Property Name="UserDesc">Will Smith Sr.</Property>
          </Item>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 17:30:52.253]
 UserNames.ID=[018ae411-42c3-48cb-a50b-b943dcb70dbe]" ItemNo="2" Ac
tion="Delete" />
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response includes the updated description of user **wsmith**, and the details of the new user **jdoe**.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <CollectionID />
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 17:32:43.233]
 UserNames.ID=[b3aedc23-722f-4058-b12c-f14bdfd955b4]" ItemNo="0" Action="Up
date">
            <Property Name="UserDesc">Will Smith Sr.</Property>
          </Item>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 17:32:43.283]
 UserNames.ID=[1a69ed89-a34e-4634-95fd-dad989ef3a39]" ItemNo="0" Action="In
sert">
            <Property Name="UserDesc">John Doe</Property>
```

```
            <Property Name="Username">jdoe</Property>
            <Property Name="InWorkflow">0</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Example 2 - Nested UpdateCollection

UpdateCollection requests can be hierarchichal. For example, this request document inserts a user-defined type and its value in a single request, by nesting UserDefinedTypeValues IDO UpdateCollection request inside the UserDefinedTypes IDO UpdateCollection request.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <RequestHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <UpdateCollection Name="UserDefinedTypes" RefreshAfterUpdate="Y">
        <Items>
          <Item ID="" ItemNo="0" Action="Insert">
            <Property Name="Description">An amount of time used with cal
endars</Property>
            <Property Name="Name">Month</Property>
           <UpdateCollection Name="UserDefinedTypeValues" RefreshAfterUp
date="Y">
              <LinkBy>
                <PropertyPair Parent="Name" Child="TypeName" />
              </LinkBy>
              <Items>
                <Item ID="" ItemNo="0" Action="Insert">
                  <Property Name="TypeName">Month</Property>
                  <Property Name="Value">January</Property>
                </Item>
              </Items>
            </UpdateCollection>
          </Item>
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response contains the complete information regarding the new user-defined type and its value.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <UpdateCollection Name="UserDefinedTypes" RefreshAfterUpdate="Y">
        <CollectionID />
        <Items>
          <Item ID="PBT=[UserDefinedTypes] UserDefinedTypes.ID=[f22b0d6b-
fdba-4b14-b008-4fd418b0cd75] UserDefinedTypes.DT=[2019-08-16 09:54:34.450]"
 ItemNo="0" Action="Insert">
            <Property Name="Description">An amount of time used with cal
endards</Property>
            <Property Name="Name">Month</Property>
            <Property Name="InWorkflow">0</Property>
           <UpdateCollection Name="UserDefinedTypeValues" RefreshAfterUp
date="Y">
              <LinkBy>
                <PropertyPair Parent="Name" Child="TypeName" />
              </LinkBy>
              <CollectionID />
              <Items>
                <Item ID="PBT=[UserDefinedTypeValues] UserDefinedTypeVal
ues.ID=[54a99f45-7a9e-4c3b-a974-7fe496db8c8b] UserDefinedTypeVal
ues.DT=[2019-08-16 09:54:34.463]" ItemNo="0" Action="Insert">
                  <Property Name="TypeName">Month</Property>
                  <Property Name="Value">January</Property>
                  <Property Name="InWorkflow">0</Property>
                </Item>
              </Items>
            </UpdateCollection>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Example 3 - UpdateCollection using item IDs

This example request updates an existing record using the **ID** attribute. To find a specific ID, perform a LoadCollection request on the data before performing the UpdateCollection.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
```

```
  <RequestHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 17:32:43.233]
 UserNames.ID=[b3aedc23-722f-4058-b12c-f14bdfd955b4]" ItemNo="0" Action="Up
date">
            <Property Name="UserDesc">Will Smith</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response includes the updated description of user **wsmith**.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <CollectionID />
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 18:38:30.520]
 UserNames.ID=[b3aedc23-722f-4058-b12c-f14bdfd955b4]" ItemNo="0" Action="Up
date">
            <Property Name="UserDesc">Will Smith</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Example 4 - UpdateCollection using keys

This example request updates an existing record using the **UseKeys** attribute which indicates that the update or delete request will use key data.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <RequestHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <Items>
          <Item ItemNo="0" Action="Update" UseKeys="Y">
            <Property Name="UserDesc">John Mark Doe</Property>
            <Property Name="UserId" Modified="N">45</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response includes the updated description of the user with user ID **45**.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-
ed844ffe7129">
  <ResponseHeader Type="UpdateCollection">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <UpdateCollection Name="UserNames" RefreshAfterUpdate="Y">
        <CollectionID />
        <Items>
         <Item ID="PBT=[UserNames] UserNames.DT=[2019-08-15 18:46:16.520]
 UserNames.ID=[1a69ed89-a34e-4634-95fd-dad989ef3a39]" ItemNo="0" Action="Up
date">
            <Property Name="UserDesc">John Mark Doe</Property>
            <Property Name="UserId" Modified="N">45</Property>
          </Item>
        </Items>
      </UpdateCollection>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

Invoke

An Invoke request/response document executes an IDO method. This method can be code in a custom assembly, or it can be a stored procedure.

Request data

An Invoke request document has this payload structure and contains data such as the collection name, method name, and the method parameter values, which are required to complete the request.

```
<RequestData>
  <Name>collection</Name>
  <Method>methodname</Method>
  <Parameters>
    <Parameter>parametervalue</Parameter>
    <Parameter>parametervalue</Parameter>
    <Parameter ByRef="Y" />
  </Parameters>
</RequestData>
```

Response data

An Invoke response document has this payload structure and includes requested data and return values.

```
<ResponseData>
  <Name>collection</Name>
  <Method>methodname</Method>
  <Parameters>
    <Parameter>parametervalue</Parameter>
    <Parameter>parametervalue</Parameter>
    <Parameter ByRef="Y">parametervalue</Parameter>
  </Parameters>
  <ReturnValue>returnvalue</ReturnValue>
</ResponseData>
```

Element descriptions

Element	Description
Method	This is the name of the IDO that contains the method.
Name	This is the name of the method being invoked.

Element	Description
Parameter	This is the name of a parameter for the method. This element can contain the **ByRef** attribute. This attribute is optional. When used, parameters that are marked as **ByRef** in the IDO metadata are returned in the response, regardless of the **ByRef** value in the request. If set to **Y**, the parameter is input/output. If set to **N**, or if this attribute is omitted, the parameter is input-only. **Note:** Parameters should be listed in the order that the method is expecting them. If you use the obsolete **Seq** attribute, it is ignored.
ReturnValue	Used only in the response document, this element contains the return value from the method.

Examples for Invoke

This example request determines the user attributes of user **jdoe** using the IDO method GetUserAttributes.

```
<IDORequest ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-ed844ffe7129">
  <RequestHeader Type="Invoke">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData>
      <Name>UserNames</Name>
      <Method>GetUserAttributes</Method>
      <Parameters>
        <Parameter>jdoe</Parameter>
        <Parameter ByRef="Y" />
        <Parameter ByRef="Y" />
        <Parameter ByRef="Y" />
        <Parameter ByRef="Y" />
      </Parameters>
    </RequestData>
  </RequestHeader>
</IDORequest>
```

This example response contains the user attributes of the user **jdoe**.

```
<IDOResponse ProtocolVersion="6.03" SessionID="bde8caa4-1343-4808-9bca-ed844ffe7129">
  <ResponseHeader Type="Invoke">
    <InitiatorType />
    <InitiatorName />
```

```
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData>
      <Name>UserNames</Name>
      <Method>GetUserAttributes</Method>
      <Parameters>
        <Parameter>jdoe</Parameter>
        <Parameter ByRef="Y">2</Parameter>
        <Parameter ByRef="Y">1</Parameter>
        <Parameter ByRef="Y" />
        <Parameter ByRef="Y" />
      </Parameters>
      <ReturnValue>0</ReturnValue>
    </ResponseData>
  </ResponseHeader>
</IDOResponse>
```

CloseSession

A CloseSession request/response document closes an existing session.

Request data

None

Response data

None

Example for CloseSession

This example request closes a session with the given session ID.

```
<IDORequest ProtocolVersion="6.03" SessionID="71e1c28d-4e00-495e-bff0-
571b26179649">
  <RequestHeader Type="CloseSession">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <RequestData />
  </RequestHeader>
</IDORequest>
```

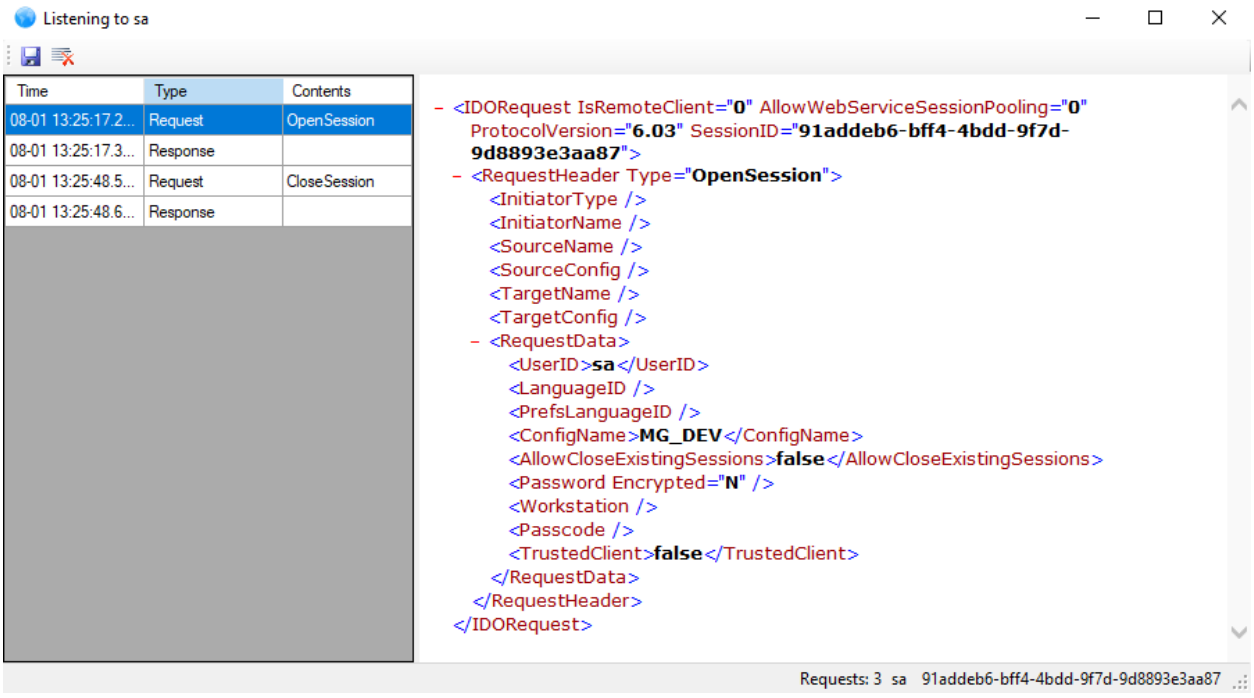
This is an example response of the CloseSession request.

```
<IDOResponse ProtocolVersion="6.03" SessionID="71e1c28d-4e00-495e-bff0-
571b26179649">
  <ResponseHeader Type="CloseSession">
    <InitiatorType />
    <InitiatorName />
    <SourceName />
    <SourceConfig />
    <TargetName />
    <TargetConfig />
    <ResponseData />
  </ResponseHeader>
</IDOResponse>
```

Using Wire Taps

A good way to become more familiar with the structure of the IDO Request XML Schema is to monitor the IDO communication exchange using a Wire Tap. A Wire Tap allows you to monitor the IDO requests and responses that are sent and received in a user session.

In the **Listening To <*user*>** window of a Wire Tap, you can select and view each IDO XML document associated with the session.



Adding a Wire Tap

A Wire Tap allows you to monitor the IDO requests and responses that are sent and received in a user session.

1. Launch the IDO Runtime Development Server (IDORuntimeHost.exe).

   The IDO Runtime Service must not be running on the same machine, unless you include the **`/multiuser`** command line parameter when starting IDORuntimeHost.exe.

2. Launch Mongoose and sign in to the desired configuration and application.
3. In the IDO Runtime Development Server, the configuration panel, expand the configuration to which the you are logged on.
4. Select the user name under which you signed in.
5. Right-click the user name and select **Add Wire Tap**.

   The IDO Runtime Development Server opens the **Listening to <*user*>** window for the selected user. This window displays all activity with respect to response and request documents. You can select from the list which document you want to view.

   You can also save the request/response wire tap information for further examination and analysis.

Chapter 3: Using the REST API

The REST API allows you to access IDOs from a client-side web project, a mobile app, or an Internet of Things (IoT) device to perform IDO-related operations such as loading or updating collections, or running methods. It is another layer on top of the IDO Request Interface, using the IDO Request XML schema.

Integration with ION API

The Mongoose REST APIs are also exposed in Infor ION API which allows customers to have one common authentication mechanism, a consolidated view, and consumption of APIs from many Infor applications. Our APIs are developed to allow ION API to consume them using the OAuth 1.0a Zero-Legged authentication mechanism.

For more information on how to use Infor ION API and how to interact with the swagger documentation for the API methods, see the *Infor ION API Administration Guide*.

REST API versions

The original Mongoose REST API was grown over several years. As a result, it became overly complex and difficult to learn. While preserving it for backward-compatibility, Mongoose added a second version of the API. Both are described later in this chapter.

Authentication

The Mongoose REST API offers security by requiring that either a Mongoose security token or a valid OAuth 2.0 bearer token be passed in the Authorization header of the API requests.

The Mongoose REST API currently supports these authentication schemes:

- Mongoose security token
- OAuth 1.0a Zero-Legged authentication

Mongoose security token

A Mongoose security token can be acquired by using the GetSecurityToken API. This is used for authenticating requests when the REST API is being called directly.

Mongoose security tokens are unique for each user and should be stored securely. These tokens only expire when the user or configuration becomes non-existent, or when the user changes the password.

OAuth 1.0a Zero-Legged authentication

OAuth 1.0a authentication is a signature that is computed using a consumer key and a matching consumer secret, along with the details of the request.

For the Mongoose REST API, the OAuth 1.0 with HMAC-SHA1 algorithm is used when the REST API is being called through the ION API.

This authentication scheme uses these headers to create a Mongoose session using the workstation logon mechanism:

Header	Description
`X-Infor-Identity2`	This is a unique, immutable user identifier within a tenant. It is provided by the ION API.
`X-Infor-MongooseConfig`	This is the Mongoose configuration name, as provided by the user.

These properties must be configured in the IDO Request Service's web.config:

Property name	Property value
enableRestWorkstationLogon	True
restOAuthConsumerKey	Consumer key provided by ION API
restOAuthConsumerSecret	Consumer secret provided by ION API

REST API, Version 1

This section presents the original version of the Mongoose REST API.

Base URL

All endpoints are accessible using HTTP(S) and are located at this URL:

`http://<serverName>/IDORequestService/MGRESTService.svc`

API documentation

Swagger API documentation is available for the Mongoose REST API. It fully describes each API operation that includes both required and optional parameters, Request Data examples, and so on. You can access it using this endpoint:

`http://<serverName>/IDORequestService/MGRESTService.svc/api-docs`

The Swagger API documentation can be transformed into a rich UI that allows you to browse through the API and try out some of the operations.

Content type

Each endpoint allows you to choose any of these content types for the request and response Request Data:

- XML
- JSON

No other content types are supported.

Mongoose REST Version 1 API endpoints

These are the Mongoose REST Version 1 API endpoints:

- AddItem
  - AddItem with refresh
  - AddItems
  - AddItems with refresh
- DeleteItem
- DeleteItems
- Download Document Object
- Download File Stream
- Fire AES Event
- Fire AES Event Advanced
- Get Configurations
- Get Security Token
- IDOInfo
- InvokeMethod
- LoadCollection
  - LoadCollection with properties
  - Advanced LoadCollection
  - Advanced LoadCollection with properties
- Load Property Values
- UpdateItem
  - UpdateItem with refresh
  - UpdateItems

    - UpdateItems with refresh
- Upload File Stream
- Upload Document Object

All API calls, except GetConfigurations and GetSecurityToken APIs, are required to include an Authorization header that contain either a Mongoose security token or a valid OAuth 2.0 bearer token, depending on how the API is being called.

We recommend that you use the prototypes presented here as examples when creating your API calls. All these prototype examples use C# code.

AddItem

The AddItem API inserts a record into a specified IDO collection.

POST	/{responsetype}/{ido}/additem
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/additem	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<IDOUpdateItem
```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <Action>Insert</Action>
  <ItemId>PBT=[Usernames]</ItemId>
  <Properties>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>Username</Name>
      <Value>jdelacruz</Value>
    </UpdateProperty>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>UserDesc</Name>
      <Value>Juan Dela Cruz</Value>
    </UpdateProperty>
  </Properties>
</IDOUpdateItem>
```

Request data in JSON format

```
{
 "Action": 1,
 "ItemId": "PBT=[Usernames]",
   "Properties": [
      {
         "IsNull": false,
         "Modified": true,
         "Name": "Username",
         "Value": "jdelacruz"
      },
      {
         "IsNull": false,
         "Modified": true,
         "Name": "UserDesc",
         "Value": "Juan Dela Cruz"
      }
   ]
}
```

Response data in XML format

```
<MGRestUpdateResponse xmlns="http://schemas.datacontract.org/2004/07/"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Insert successful.</Message>
  <MessageCode>200</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Insert successful.",
   "MessageCode": 200
}
```

Example

This example code inserts **jdoe** as a new user.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/additems";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty username = new UpdateProperty();
   username.Name = "Username";
   username.Value = "jdoe";
   username.IsNull = false;
   username.Modified = true;

   UpdateProperty userDesc = new UpdateProperty();
   userDesc.Name = "UserDesc";
   userDesc.Value = "John Doe";
   userDesc.IsNull = false;
   userDesc.Modified = true;

   IDOUpdateItem idoItem = new IDOUpdateItem();
   idoItem.Action = UpdateAction.Insert;
   idoItem.Properties = new[] { username, userDesc };
   idoItem.ItemId = "PBT=[UserNames]";
   idoItems.Add( idoItem );

   username = new UpdateProperty();
   username.Name = "Username";
   username.Value = "wsmith";
   username.IsNull = false;
   username.Modified = true;

   userDesc = new UpdateProperty();
   userDesc.Name = "UserDesc";
   userDesc.Value = "Will Smith";
```

```
   userDesc.IsNull = false;
   userDesc.Modified = true;

   idoItem = new IDOUpdateItem();
   idoItem.Action = UpdateAction.Insert;
   idoItem.Properties = new[] { username, userDesc };
   idoItem.ItemId = "PBT=[UserNames]";
   idoItems.Add( idoItem );

   // pass the List<IDOUpdateItem> as the request data
  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
      serializer.WriteObject( writer, idoItems );
   }

   // send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
```

```
    Insert = 1,
    Update = 2,
    Delete = 4
}

public enum UpdateLocking
{
    Row = 0,
    Property = 1
}
```

AddItem with refresh

The AddItem with refresh API inserts a record into a specified IDO and then refreshes the collection.

POST	/{responsetype}/{ido}/additem/adv

http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/additem/adv?refresh=ALL&props=Username,UserDesc

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
refresh	Query	No	This parameter provides the instruction to refresh the collection. Supply either of these values: • **ALL** - Refreshes the entire collection and all of its properties. • **PROPS** - Refreshes only the IDO properties.
props	Query	No	This is a comma-delimited list of the properties to refresh.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.

Name	Description
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<IDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <Action>Insert</Action>
  <ItemId>PBT=[Usernames]</ItemId>
  <Properties>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>Username</Name>
      <Value>jdelacruz</Value>
    </UpdateProperty>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>UserDesc</Name>
      <Value>Juan Dela Cruz</Value>
    </UpdateProperty>
  </Properties>
</IDOUpdateItem>
```

Request data in JSON format

```
{
 "Action": 1,
 "ItemId": "PBT=[Usernames]",
   "Properties": [
      {
         "IsNull": false,
         "Modified": true,
         "Name": "Username",
         "Value": "jdelacruz"
      },
      {
         "IsNull": false,
         "Modified": true,
         "Name": "UserDesc",
         "Value": "Juan Dela Cruz"
      }
   ]
}
```

Response data in XML format

```
<MGRestUpdateResponse xmlns="http://schemas.datacontract.org/2004/07/"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Insert successful.</Message>
  <MessageCode>200</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Insert successful.",
   "MessageCode": 200
}
```

Example

This example code inserts **jdelacruz** as a new user and returns the updated property values.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string refresh = "ALL";
   string props = "Username,UserDesc";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/additem/adv?refresh={refresh}&props={props}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs" );

   UpdateProperty username = new UpdateProperty
   {
      Name = "Username",
      Value = "jdelacruz",
      IsNull = false,
      Modified = true
   };

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Juan Dela Cruz",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
```

```
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItem.GetType() );
      serializer.WriteObject( writer, idoItem );
   }

   // pass the IDOUpdateItem as the request data and send the insert re
quest
   HttpResponseMessage response = client.PostAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}
```

```
public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

AddItems

The AddItems API inserts multiple new records into a specified IDO collection.

POST	/{responsetype}/{ido}/additems
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/additems	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<ArrayOfIDOUpdateItem xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.infor.com">
  <IDOUpdateItem>
    <Action>Insert</Action>
    <ItemId>PBT=[Usernames]</ItemId>
```

```
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>Username</Name>
        <Value>jdoe</Value>
      </UpdateProperty>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>John Doe </Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
  <IDOUpdateItem>
    <Action>Insert</Action>
    <ItemId>PBT=[Usernames]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>Username</Name>
        <Value>wsmith</Value>
      </UpdateProperty>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>Will Smith</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
</ArrayOfIDOUpdateItem>
```

Request data in JSON format

```
[
  {
     "Action": 1,
     "ItemId": "PBT=[Usernames]",
     "Properties": [
        {
           "IsNull": false,
           "Modified": true,
           "Name": "Username",
           "Value": "jdoe"
        },
        {
           "IsNull": false,
           "Modified": true,
           "Name": "UserDesc",
           "Value": "John Doe"
        }
```

```
      ]
   },
   {
      "Action": 1,
      "ItemId": "PBT=[Usernames]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": true,
            "Name": "Username",
            "Value": "wsmith"
         },
         {
            "IsNull": false,
            "Modified": true,
            "Name": "UserDesc",
            "Value": "Will Smith "
         }
      ]
   }
]
```

Response data in XML format

```
<MGRestUpdateResponse xmlns="http://schemas.datacontract.org/2004/07/"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Insert successful.</Message>
  <MessageCode>200</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Insert successful.",
   "MessageCode": 200
}
```

Example

This example code inserts users **jdoe** and **wsmith** as new users.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/additems";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
```

```
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty username = new UpdateProperty
   {
      Name = "Username",
      Value = "jdoe",
      IsNull = false,
      Modified = true
   };

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };
   idoItems.Add( idoItem );

   username = new UpdateProperty
   {
      Name = "Username",
      Value = "wsmith",
      IsNull = false,
      Modified = true
   };

   userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Will Smith",
      IsNull = false,
      Modified = true
   };

   idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };
   idoItems.Add( idoItem );

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
) );
```

```
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
      serializer.WriteObject( writer, idoItems );
   }

   // pass the List<IDOUpdateItem> as the request data and send the insert
 request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

AddItems with refresh

The AddItems with refresh API inserts multiple records into a specified IDO and then refreshes the collection.

POST	/{responsetype}/{ido}/additems/adv
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/additems/adv?refresh=ALL&props=Username,UserDesc	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
refresh	Query	No	This parameter provides the instruction to refresh the collection. Supply either of these values: • **ALL** - Refreshes the entire collection and all of its properties. • **PROPS** - Refreshes only the IDO properties.
props	Query	No	This is a comma-delimited list of the properties to refresh.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<ArrayOfIDOUpdateItem xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.infor.com">
  <IDOUpdateItem>
    <Action>Insert</Action>
```

```
    <ItemId>PBT=[Usernames]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>Username</Name>
        <Value>jdoe</Value>
      </UpdateProperty>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>John Doe </Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
  <IDOUpdateItem>
    <Action>Insert</Action>
    <ItemId>PBT=[Usernames]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>Username</Name>
        <Value>wsmith</Value>
      </UpdateProperty>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>Will Smith</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
</ArrayOfIDOUpdateItem>
```

Request data in JSON format

```
[
  {
     "Action": 1,
     "ItemId": "PBT=[Usernames]",
     "Properties": [
        {
           "IsNull": false,
           "Modified": true,
           "Name": "Username",
           "Value": "jdoe"
        },
        {
           "IsNull": false,
           "Modified": true,
           "Name": "UserDesc",
           "Value": "John Doe"
```

```
            }
         ]
      },
      {
         "Action": 1,
         "ItemId": "PBT=[Usernames]",
         "Properties": [
            {
               "IsNull": false,
               "Modified": true,
               "Name": "Username",
               "Value": "wsmith"
            },
            {
               "IsNull": false,
               "Modified": true,
               "Name": "UserDesc",
               "Value": "Will Smith "
            }
         ]
      }
]
```

Response data in XML format

```
<MGRestUpdateResponse xmlns="http://schemas.datacontract.org/2004/07/"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Insert successful.</Message>
  <MessageCode>200</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Insert successful.",
   "MessageCode": 200
}
```

Example

This example code inserts users **jdoe** and **wsmith** as new users and returns the updated property values.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string refresh = "ALL";
   string props = "Username,UserDesc";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
```

```
vice.svc/xml/{ido}/additems/adv?refresh={refresh}&props={props}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5v1903teP0jSDwkFs" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty username = new UpdateProperty
   {
      Name = "Username",
      Value = "jdoe",
      IsNull = false,
      Modified = true
   };

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };
   idoItems.Add( idoItem );

   username = new UpdateProperty
   {
      Name = "Username",
      Value = "wsmith",
      IsNull = false,
      Modified = true
   };

   userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Will Smith",
      IsNull = false,
      Modified = true
   };

   idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };
```

```
   idoItems.Add( idoItem );

   XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
      serializer.WriteObject( writer, idoItems );
   }

   // pass the List<IDOUpdateItem> as the request data and send the insert
 request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
```

```
   Row = 0,
   Property = 1
}
```

DeleteItem

The DeleteItem API removes a single record from a specified IDO collection.

DELETE `/{responsetype}/{ido}/deleteitem`

```
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/
      deleteitem?itemid=PBT=[UserNames] UserNames.DT=[2019-07-
      24 13:28:51.960] UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]
```

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
itemid	Query	Yes	This is the value of the IDO _itemid property.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
```

```
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Delete successful.</Message>
  <MessageCode>202</MessageCode>
</MGRestUpdateResponse
```

Response data in JSON format

```
{
   "Message": "Delete successful.",
   "MessageCode": 202
}
```

Example

This example code deletes a user that has the given _ItemID property value.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
  string itemid = "PBT=[UserNames] UserNames.DT=[2019-07-19 09:35:11.153]
 UserNames.ID=[ece4f2a5-e155-49c6-b2bd-afee64303c3f]";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/deleteitem?itemid={itemid}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

  HttpResponseMessage response = client.DeleteAsync( requestUrl ).Result;


   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
```

```
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

DeleteItems

The DeleteItems API removes multiple records from a specified IDO collection at one time.

DELETE	/{responsetype}/{ido}/deleteitems
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/deleteitems	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Reqiest data in XML format

```
<?xml version="1.0"?>
<ArrayOfIDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <IDOUpdateItem>
    <Action>Delete</Action>
    <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.860] User
Names.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>false</Modified>
        <Name>_ItemId</Name>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
  <IDOUpdateItem>
    <Action>Delete</Action>
    <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.997] User
Names.ID=[3faaaaab-47ef-4643-8255-756976238911]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>false</Modified>
        <Name>_ItemId</Name>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
</ArrayOfIDOUpdateItem>
```

Request data in JSON format

```
[
   {
      "Action": 4,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.860]
```

```
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": false,
            "Name": "_ItemId",
            "Value": null
         }
      ]
   },
   {
      "Action": 4,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.997]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": false,
            "Name": "_ItemId",
            "Value": null
         }
      ]
   }
]
```

Response data in XML format

```
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Delete successful.</Message>
  <MessageCode>202</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Delete successful.",
   "MessageCode": 202
}
```

Example

This example code deletes multiple users that have the given _ItemID property values.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
```

```
vice.svc/xml/{ido}/deleteitems";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty itemId = new UpdateProperty
   {
      Name = "ItemId",
      Value = null,
      IsNull = false,
      Modified = false
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Delete,
      Properties = new[] { itemId },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.860]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]"
   };
   idoItems.Add( idoItem );

   itemId = new UpdateProperty
   {
      Name = "ItemId",
      Value = null,
      IsNull = false,
      Modified = false
   };

   idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Delete,
      Properties = new[] { itemId },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:58:55.997]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]"
   };
   idoItems.Add( idoItem );

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
      serializer.WriteObject( writer, idoItems );
   }

  // pass the List<IDOUpdateItem> as the request data and send the delete
 request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
```

```
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

Download Document Object

The Download Document Object API downloads a streamed file from a document object.

GET	/io/downloaddocobj

```
http://localhost/IDORequestService/MGRESTService.svc/xml/io/xml/downloaddocobj?
   ido=UserNames&&rp=7dbf8871-e688-44af-b4dd-111fce42395b&docname=WeeklyReport
```

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 121.

Parameters

Name	In	Required?	Description
ido	Query	Yes	This is the name of the IDO collection.
rp	Query	No	This is the value of an IDO row pointer. **Note:** This is optional if the **filter** parameter is used.
docname	Query	No	This is the name of the document object. **Note:** This parameter is optional if the **refseq** parameter is used.
refseq	Query	No	This is the reference sequence number of the document object. **Note:** This parameter is optional if the **docname** parameter is used.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

The response is an unencoded binary stream of the file, with the content-type of application/octet-stream.

Example

This example code downloads a document object from the referenced IDO and IDO rowpointer.

```
string xml = string.Empty;

using ( var client = new HttpClient() )
{
   string ido = "UserNames";
   string docname = "WeeklyReport";
   string rp = "2807a627-577b-462e-9494-aee568152c54";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/io/downloaddocobj/?ido={ido}&rp={rp}&docname={docname}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( var content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Download File Stream

The Download File Stream API downloads a streamed file from an IDO collection. The IDO collection must have a property for storing binary data.

GET	/io/uploadfile

```
http://localhost/IDORequestService/MGRESTService.svc/io/downloadfile/?
   ido=UserNames&prop=UserImage&rp=2807a627-577b-462e-9494-aee568152c54
```

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 123.

Parameters

Name	In	Required?	Description
ido	Query	Yes	This is the name of the IDO collection.
prop	Query	Yes	This is the name of an IDO property.
rp	Query	No	This is the value of an IDO row pointer. **Note:** This is optional if the **filter** parameter is used.

Name	In	Required?	Description
filter	Query	No	This applies a filter for the required row. If multiple rows returned, the first row is selected. For optimal performance, use a filter that returns only one row. **Note:** This parameter is optional if the **rp** parameter is used.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

The response is an unencoded binary stream of the file, with the content-type of application/octet-stream.

Example

This example code downloads a user image from the users table.

```
var xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string prop = "UserImage";
   string rp = "2807a627-577b-462e-9494-aee568152c54";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/io/downloadfile/?ido={ido}&prop={prop}&rp={rp}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
```

```
    {
        Task<string> result = content.ReadAsStringAsync();
        xml = result.Result;
    }
}
```

Fire AES Event

The Fire AES Event API fires an Application Event System (AES) event and, optionally, returns all parameters.

POST	/{responsetype}/aes?eventname={eventname}&parms={parms}

```
http://localhost/IDORequestService/MGRESTService.svc/xml/aes?eventname=
   DivideNumbers&parms=Dec1(20),Dec2(5),ProductVal(0),Infobar()&returnparms=true&debug=true
```

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 124.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • js
eventname	Query	Yes	This is the name of the AES event.
parms	Query	No	This is a comma-delimited list of event parameters. Use this format for parameters: `parameter1(value), parameter2(value), ...`
returnparms	Query	No	This parameter is used as a flag for returning parameter values.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<FireAESEventParmsInUrlXMLResponse xmlns="http://www.infor.com">
  <FireAESEventParmsInUrlXMLResult xmlns:a="http://schemas.datacon
tract.org/2004/07/" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <a:EventName>DivideNumbers</a:EventName>
    <a:Message>
      Input Parameters:&#xD;
      Name: {Num1}, Value: {20}&#xD;
      Name: {Num2}, Value: {5}&#xD;
      Name: {Quotient}, Value: {0}&#xD;
      Event execution success: False&#xD;
      Output Parameters:&#xD;
      Name: {Num1}, Value: {20}&#xD;
      Name: {Num2}, Value: {5}&#xD;
      Name: {Quotient}, Value: {4}&#xD;
      AES Event execution complete&#xD;
    </a:Message>
    <a:MessageCode>600</a:MessageCode>
    <a:Parameters>
      <Parameter>
        <Name>Num1</Name>
        <Value>20</Value>
        <Return>true</Return>
      </Parameter>
      <Parameter>
        <Name>Num2</Name>
        <Value>5</Value>
        <Return>true</Return>
      </Parameter>
      <Parameter>
        <Name>Quotient</Name>
        <Value>4</Value>
        <Return>true</Return>
      </Parameter>
    </a:Parameters>
  </FireAESEventParmsInUrlXMLResult>
</FireAESEventParmsInUrlXMLResponse>
```

Response data in JavaScript format

```
{
    "EventName": "DivideNumbers",
    "Parameters": [
        {
            "Name": "Num1",
            "Value": "20",
            "Return": true
        },
```

```
        {
            "Name": "Num2",
            "Value": "5",
            "Return": true
        },
      {
         "Name": "Quotient",
         "Value": "4",
         "Return": true
      }
    ],
    "Message": "\tInput Parameters:\r\n\tName: {Num1}, Value:
{20}\r\n\tName: {Num2}, Value: {5}\r\n\tName: {Quotient}, Value:
{0}\r\n\tEvent execution success: False\r\n\tOutput Parameters:\r\n\tName:
 {Num1}, Value: {20}\r\n\tName: {Num2}, Value: {5}\r\n\tName: {Quotient},
 Value: {4}\r\n\tAES Event execution complete\r\n",
    "MessageCode": 600
}
```

Example

This example code executes an AES event that divides two numbers and returns the quotient as an output parameter.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use js as the response type
   string eventname = "DivideNumbers";
   string parms = "Num1(120), Num2(6), Quotient(0)";
   bool returnparms = true;
   bool debugmode = true;
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/aes/?eventname={eventname}&parms={parms}&returnparms={return
parms}&debugmode={debugmode}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.PostAsync( requestUrl, null
).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Fire AES Event Advanced

The Fire AES Even tAdvanced API fires an Application Event System (AES) event with the option to include parameters in the request body. This is ideal for large amounts of data that need to be sent when firing an event.

POST	/{responsetype}/aes/adv

http://localhost/IDORequestService/MGRESTService.svc/xml/aes/adv?debug=true

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
debug	Query	Yes	This parameter is used as a flag for returning verbose debug and test information.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<FireAESEventRequest
 xmlns="http://www.infor.com"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <EventName>DivideNumbers</EventName>
  <Parameters>
    <Parameter>
      <Name>Num1</Name>
      <Value>200</Value>
      <Return>false</Return>
    </Parameter>
    <Parameter>
```

```
      <Name>Num2</Name>
      <Value>25</Value>
      <Return>false</Return>
    </Parameter>
    <Parameter>
      <Name>Quotient</Name>
      <Value>0</Value>
      <Return>true</Return>
    </Parameter>
    <Parameter>
      <Name>Infobar</Name>
      <Value></Value>
      <Return>true</Return>
    </Parameter>
  </Parameters>
</FireAESEventRequest>
```

Request data in JSON format

```
{
   "EventName": "DivideNumbers",
   "Parameters": [
      {
         "Name": "Num1",
         "Return": false,
         "Value": 200
      },
      {
         "Name": "Num2",
         "Return": false,
         "Value": 25
      },
      {
         "Name": "Quotient",
         "Return": true
      },
      {
         "Name": "Infobar",
         "Return": true
      }
   ]
}
```

Response data in XML format

```
<MGFireEventResponse>
  <EventName>DivideNumbers</EventName>
  <Parameters>
    <Parameter>
      <Name>Quotient</Name>
      <Return>true</Return>
      <Value>8</Value>
    </Parameter>
```

```
    <Parameter>
      <Name>Infobar</Name>
      <Return>true</Return>
      <Value>success</Value>
    </Parameter>
  </Parameters>
  <Message>Success</Message>
  <MessageCode>600</MessageCode>
</MGFireEventResponse>
```

Response data in JSON format

```
{
   "EventName": "DivideNumbers",
   "Parameters": [
      {
         "Name": "Quotient",
         "Value": "8",
         "Return": true
      },
      {
         "Name": "Infobar",
         "Value": "success",
         "Return": true
      }
   ],
   "Message": "Success",
   "MessageCode": 600
}
```

Example

This example code executes an AES event that divides two numbers and returns the quotient as an output parameter. The AES event details are passed as a request payload instead of including it in the request URL.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   bool debug = true;
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/aes/adv/?debug={debug}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   AESEventParameter num1 = new AESEventParameter();
   num1.Name = "Num1";
   num1.Return = false;
```

```
   num1.Value = "200";

   AESEventParameter num2 = new AESEventParameter();
   num2.Name = "Num2";
   num2.Return = false;
   num2.Value = "25";

   AESEventParameter quotient = new AESEventParameter();
   quotient.Name = "Quotient";
   quotient.Return = true;
   quotient.Value = "0";

   AESEventParameter[] parameters = new[] { num1, num2, quotient };
   FireAESEventRequest payload = new FireAESEventRequest();
   payload.EventName = "DivideNumbers";
   payload.Parameters = parameters;

   XDocument xdoc = new XDocument();
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
payload.GetType() );
      serializer.WriteObject( writer, payload );
   }

   // pass the AES event name and parameters as the request data and send
 the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Get Configurations

The Get Configurations API returns a list of the application configuration names available on the server.

GET	/{responsetype}/configurations
http://localhost/IDORequestService/MGRESTService.svc/xml/configurations?configgroup=DEV	

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 127.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply either of these values: • xml • json
configgroup	Query	No	This is the name of a Configuration Group. Use this parameter to get the configurations from a specific configuration group.

Headers

None

Request data

None

Response data in XML format

```
<ArrayOfstring xmlns="http://schemas.microsoft.com/2003/10/Serialization/Ar
rays" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <string>CSI_DALS</string>
  <string>CSI_EMEA</string>
  <string>CSI_LA</string>
</ArrayOfstring>
```

Response data in JSON format

```
[
   "CSI_DALS",
   "CSI_EMEA",
   "CSI_LA"
]
```

Example

This example code retrieves an array of configuration names from the specified configuration group through the ConfigServer.aspx page in IDORequestService:

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string configGroup = "DEV";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
```

```
vice.svc/xml/configurations?configgroup={configGroup}";

   // send the request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // get the xml response containing the configuration list
      xml = result.Result;
   }
}
```

Get Security Token

The Get Security Token API returns a Mongoose security token for a specific user, which can be used to authenticate requests when calling the API directly.

GET	/{responsetype}/token/{config}
http://localhost/IDORequestService/MGRESTService.svc/js/token/CSI_DALS	

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 132.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json • js
config	Path	Yes	This is the name of a configuration available on the application server.

Headers

Name	Description
userid	Mongoose username
password	Mongoose user password

Request data

None

Response data in XML format

If successful, the call returns a neatly packaged security token.

```
<MGRestTokenResponse xmlns="http://schemas.datacontract.org/2004/07/"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Success</Message>
  <Token>b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDb...YOdU2ThSw==</Token>
</MGRestTokenResponse>
```

Response data in JSON format

If successful, the call returns a neatly packaged security token.

```
{
   "Message": "Success",
   "Token": "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDb...YOdU2ThSw=="
}
```

Response data in JS format

If successful, the call returns the security token itself (no envelope).

```
b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDb...YOdU2ThSw==
```

Example

This example code retrieves a Mongoose security token for the given user ID, password, and configuration name.

```
string token = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // provide your Mongoose credentails in the request url
   string config = "CSI_DALS";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/js/token/{config}";

   // provide your Mongoose credentials as headers
   client.DefaultRequestHeaders.Add( "userid", "sa" );
   client.DefaultRequestHeaders.Add( "password", "" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // this contains your token
      token = result.Result;
```

```
    }
}
```

IDOInfo

The IDOInfo API returns the property metadata for an IDO collection.

GET	/{responsetype}/idoinfo/{ido}
http://localhost/IDORequestService/MGRESTService.svc/xml/idoinfo/UserNames	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply either of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<ArrayOfRestIdoItemPropInfo xmlns="http://schemas.datacon
tract.org/2004/07/" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <RestIdoItemPropInfo>
    <defaultValue/>
    <labelStringID>sUserID</labelStringID>
    <name>UserId</name>
```

```
    <required>true</required>
    <dataType>Decimal</dataType>
    <length>11</length>
    <readOnly>true</readOnly>
  </RestIdoItemPropInfo>
  <RestIdoItemPropInfo>
    <defaultValue/>
    <labelStringID>sUsername</labelStringID>
    <name>Username</name>
    <required>true</required>
    <dataType>String</dataType>
    <length>128</length>
    <readOnly>false</readOnly>
  </RestIdoItemPropInfo>
  <RestIdoItemPropInfo>
    ...
  </RestIdoItemPropInfo>
</ArrayOfRestIdoItemPropInfo>
```

Response data in JSON format

```
 [
   {
      "defaultValue": "",
      "labelStringID": "sUserID",
      "name": "UserId",
      "required": true,
      "dataType": "Decimal",
      "length": 11,
      "readOnly": true
   },
  {
  "defaultValue": "",
  "labelStringID": "sUsername",
  "name": "Username",
  "required": true,
  "dataType": "String",
  "length": 128,
  "readOnly": false
  },
  { ... }
]
```

Example

This example code retrieves an array of properties and their attributes from a specified IDO.

```
string xml = string.Empty;

using ( var client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
```

```
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/idoinfo/{ido}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // get the xml response
      xml = result.Result;
   }
}
```

Invoke Method

The Invoke Method API calls into use an IDO method.

GET	/{responsetype}/method/{ido}/{method}?parms={parms}
http://localhost/IDORequestService/MGRESTService.svc/xml/method/GetUserAttributes?parms=sa,,,,	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json • js
ido	Path	Yes	This is the name of the IDO collection.
method	Path	Yes	This is the name of the method to be invoked.
parms	Query	No	This is a comma-delimited list of the paramters required by the method being invoked.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<MGRestInvokeMethodResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message i:nil="true" />
  <MessageCode>0</MessageCode>
  <Parameters>
    <p>sa</p>
    <p>4</p>
    <p>1</p>
    <p>CoreFormsAdmin</p>
    <p></p>
  </Parameters>
  <ReturnValue>0</ReturnValue>
</MGRestInvokeMethodResponse>
```

Response data in JSON format

```
{
   "Message": null,
   "MessageCode": 0,
   "Parameters": [
      "sa",
      "4",
      "1",
      "CoreFormsAdmin",
      ""
   ],
   "ReturnValue": "0"
}
```

Response data in JSON Stream format

```
{
   "Parameters": {
      "Username": "sa",
      "EditLevel": 4,
      "SuperUserFlag": 1,
      "FormGroupName": "",
      "Infobar": ""
   },
   "ReturnValue": "0",
   "Message": null,
   "MessageCode": 0
}
```

Example

This example code determines the user attributes of the 'sa' user invoking the GetUserAttributes IDO method.

```
string xml = string.Empty;

using ( var client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string method = "GetUserAttributes";
   string parms = "sa,,,,";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/method/{ido}/{method}?parms={parms}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // get the xml response
      xml = result.Result;
   }
}
```

LoadCollection

The LoadCollection API is a basic load collection API that returns the first 200 records of a specified IDO collection.

GET	/{responsetype}/{ido}/{props}

```
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/
  UserId,Username,UserDesc
```

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 135.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
props	Path	Yes	This is a comma-delimited list of properties for which to return values. You can also use an asterisk (*)

Headers

Name	Description
Authorization	If the API is called directly, then the Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through the ION API, then a valid OAuth2.0 bearer token is provided by the ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Note: LoadCollection has a number of variations that can load a collection with different query parameters:

- One variation of this API (LoadCollection with Properties on page 87) accepts a property list as a query parameter. This is useful for longer lists of properties that might exceed URL length restrictions.
- Another variation (Advanced LoadCollection on page 89) offers various options to filter the return results.
- A third variation (Advanced LoadCollection with Properties on page 93) accepts a property list as a query parameter, and offers various options to filter the return results. This is useful for longer lists of properties that might exceed URL length restrictions.

Request data

None

Response data in XML format

```
<MGRestResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Items>
    <item>
      <p>1</p>
      <p>sa</p>
      <p>System Admin</p>
    </item>
    <item>
      <p>2</p>
      <p>jdoe</p>
      <p>John Doe</p>
    </item>
  </Items>
  <Message i:nil="true"/>
  <MessageCode>0</MessageCode>
</MGRestResponse>
```

Response data in JSON format

```
{
   "Items": [
      [ "1", "sa", "System Admin" ],
      [ "2", "jdoe", "John Doe" ]
   ],
   "Message": null,
   "MessageCode": 0
}
```

Example

This example code retrieves records that contains the user ID, username, and user description from the Users table.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
           // optionally, you can use json as the response type
   string ido = "UserNames";
   string properties = "UserId, Username, UserDesc";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/{properties}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDb...YOdU2ThSw==" );
```

```
    HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

    using ( HttpContent content = response.Content )
    {
        Task<string> result = content.ReadAsStringAsync();
        xml = result.Result;
    }
}
```

LoadCollection with properties

This variation of the LoadCollection API is a basic load collection API that returns the first 200 records of a specified IDO collection. Unlike the basic LoadCollection API, this endpoint includes the property list as a query parameter instead of a path parameter.

GET	/{responsetype}/{ido}?props={props}

```
http://localhost/IDORequestService/MGRESTService.svc/xml/
  UserNames?props=UserId,Username,UserDesc
```

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply either of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
props	Query	No	This is either a comma-delimited list of properties, or it is an asterisk (*), which includes all properties except subcollection properties. If this parameter is excluded or left blank, the server retrieves data for all properties.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<MGRestResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Items>
    <item>
      <p>1</p>
      <p>sa</p>
      <p>System Admin</p>
    </item>
    <item>
      <p>2</p>
      <p>jdoe</p>
      <p>John Doe</p>
    </item>
  </Items>
  <Message i:nil="true"/>
  <MessageCode>0</MessageCode>
</MGRestResponse>
```

Response data in JSON format

```
{
   "Items": [
      [ "1", "sa", "System Admin" ],
      [ "2", "jdoe", "John Doe" ]
   ],
   "Message": null,
   "MessageCode": 0
}
```

Example

This example code retrieves records that contain the user ID, username, and user description from the Users table. Notice that the properties are provided in the request as a query parameter.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
          // optionally, you can use json as the response type
   string ido = "UserNames";
   string properties = "UserId, Username, UserDesc";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}?props={properties}";

   // provide token in the Authorization header
```

```
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDb...YOdU2ThSw==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Advanced LoadCollection

This variation of the LoadCollection API returns the records of a specified IDO collection. Unlike the basic LoadCollection API and the LoadCollection with properties API, this variation offers various options to filter the return results.

GET	/{responsetype}/{ido}/{props}/adv

```
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/UserId,Username/
     adv?filter=Username%20LIKE%20'sa'
```

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json • js • schema
ido	Path	Yes	This is the name of the IDO collection.
props	Path	Yes	This is a comma-delimited list of properties.
filter	Query	No	This parameter restricts the result set. This can be any valid SQL WHERE clause.
orderby	Query	No	This parameter is a comma-delimited list of properties that specifies the order in which the result set should be sorted. To have a property sorted in descending order, use the DESC keyword after the property name.

Name	In	Required?	Description
recordcap	Query	No	This parameter specifies how many records are to be retrieved in each request. These are the valid values: • `-1`- This specifies that the system default record cap is to be used. This is typically `200`, but it can be any number as set by the system administrator. • `0` - This value specifies that all records are to be retrieved, regardless of any record cap settings. • Any other positive integer specifies the number of records that are to be retrieved.
distinct	Query	No	This parameter specifies that a set of data that represents only distinct combinations of requested properties is to be returned.
clm	Query	No	This parameter is the name of a custom load method. This parameter can be used in conjunction with the clmparms parameter.
clmparms	Query	No	This parameter is used for a comma-delimited list of custom load methods.
loadtype	Query	No	This parameter is used for load collection paging. It is used in conjunction with the bookmark parameter. To use, specify one of these types: • FIRST • NEXT • PREV • LAST
bookmark	Query	No	This parameter is for the bookmark ID. Bookmark IDs serve as a reference when you want to get to the next or previous records in a collection. The “bookmark” value uses this format: `<B><P><p>UserId</p></P><D><f>false</f></D><F><v>1</v></F><L><v>2</v></L></B>` where: • `<B></B>` is the bookmark envelope tag. • `<P></P>` is the property name of the primary key. • `<D></D>` is a flag that, if TRUE, specifies that records are to be read in descending order. • `<f></f>` is the value of the `<D></D>` element (TRUE or FALSE). • `<F></F>` is the first row in the collection. • `<L></L>` is the last row in the collection. • `<v><v>` is the value of the row for the given property name. For an example of this, see Example: Bookmark IDs in LoadCollection responses on page 174.

Name	In	Required?	Description
pqc	Query	No	This parameter specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the WinStudio **PQ** option in Load/Save Overrides and uses the same syntax.
ro	Query	No	This parameter specifies that the return results are to be marked as "Read Only". When this is set to **True**, retrieved records do not include the **_ItemID** property, which can be substantial for update and delete operations.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<MGRestResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Items>
    <item>
      <p>1</p>
      <p>sa</p>
    </item>
  </Items>
  <Message i:nil="true"/>
  <MessageCode>0</MessageCode>
</MGRestResponse>
```

Response data in JSON format

```
{
   "Items": [
      [ "1", "sa" ]
```

```
   ],
   "Message": null,
   "MessageCode": 0
}
```

Response data in JSON Stream format

```
{
 "Message": "Success",
 "MessageCode": 0,
 "Items": [
      {
         "UserId": 1,
         "Username": "sa"
      }
 ]
}
```

Response data in Full Schema format

```
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:MainDataT
able="UserNames" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="UserNames">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="UserId" msdata:Caption="sUserID"
type="xs:decimal"/>
                <xs:element name="Username" msdata:Caption="sUsername"
type="xs:string"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <UserNames>
    <UserId>1</UserId>
    <Username>sa</Username>
  </UserNames>
</NewDataSet>
```

Example

This example code retrieves the user ID, username, and user description from the Users table for the 'sa' user and uses the filter parameter.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string properties = "UserId, Username, UserDesc";
   string filter = "Username%20LIKE%20'sa'";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/{properties}/adv?filter={filter}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Advanced LoadCollection with properties

This variation of the LoadCollection API returns the records of the specified IDO collection. This endpoint includes the property list as a query parameter instead of a path parameter. Unlike the basic LoadCollection API and the LoadCollection with properties API, this variation offers various options to filter the result.

GET	/{responsetype}/{ido}/adv
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/adv?props=UserId,Username&filter=Username%20LIKE%20'sa'	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json • js • schema
ido	Path	Yes	This is the name of the IDO collection.
props	Query	Yes	This is a comma-delimited list of properties.
filter	Query	No	This parameter restricts the result set. This can be any valid SQL WHERE clause.
orderby	Query	No	This parameter is a comma-delimited list of properties that specifies the order in which the result set should be sorted. To have a property sorted in descending order, use the DESC keyword after the property name.
recordcap	Query	No	This parameter specifies how many records are to be retrieved in each request. These are the valid values: • `-1`- This specifies that the system default record cap is to be used. This is typically `200`, but it can be any number as set by the system administrator. • `0` - This value specifies that all records are to be retrieved, regardless of any record cap settings. • Any other positive integer specifies the number of records that are to be retrieved.
distinct	Query	No	This parameter specifies that a set of data that represents only distinct combinations of requested properties is to be returned.
clm	Query	No	This parameter is the name of a custom load method. This parameter can be used in conjunction with the clmparms parameter.
clmparms	Query	No	This parameter is used for a comma-delimited list of custom load methods.
loadtype	Query	No	This parameter is used for load collection paging. It is used in conjunction with the bookmark parameter. To use, specify one of these types: • FIRST • NEXT • PREV • LAST

Name	In	Required?	Description
bookmark	Query	No	This parameter is for the bookmark ID. Bookmark IDs serve as a reference when you want to get to the next or previous records in a collection. The “bookmark” value uses this format: `<B><P><p>UserId</p></P><D><f>false</f></D><F><v>1</v></F><L><v>2</v></L></B>` where: • `<B></B>` is the bookmark envelope tag. • `<P></P>` is the property name of the primary key. • `<D></D>` is a flag that, if TRUE, specifies that records are to be read in descending order. • `<f></f>` is the value of the `<D></D>` element (TRUE or FALSE). • `<F></F>` is the first row in the collection. • `<L></L>` is the last row in the collection. • `<v><v>` is the value of the row for the given property name. For an example of this, see Example: Bookmark IDs in LoadCollection responses on page 174.
pqc	Query	No	This parameter specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the WinStudio **PQ** option in Load/Save Overrides and uses the same syntax.
ro	Query	No	This parameter specifies that the return results are to be marked as "Read Only". When this is set to **True**, retrieved records do not include the **_ItemID** property, which can be substantial for update and delete operations.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data in XML format

```
<MGRestResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Items>
    <item>
      <p>1</p>
      <p>sa</p>
    </item>
  </Items>
  <Message i:nil="true"/>
  <MessageCode>0</MessageCode>
</MGRestResponse>
```

Response data in JSON format

```
{
   "Items": [
      [ "1", "sa" ]
   ],
   "Message": null,
   "MessageCode": 0
}
```

Response data in JSON Stream format

```
{
   "Message": "Success",
   "MessageCode": 0,
   "Items": [
      {
         "UserId": 1,
         "Username": "sa"
      }
   ]
}
```

Response data in Full Schema format

```
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:MainDataT
able="UserNames" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="UserNames">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="UserId" msdata:Caption="sUserID"
```

```
type="xs:decimal"/>
                  <xs:element name="Username" msdata:Caption="sUsername"
type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
       </xs:choice>
     </xs:complexType>
   </xs:element>
  </xs:schema>
  <UserNames>
    <UserId>1</UserId>
    <Username>sa</Username>
  </UserNames>
</NewDataSet>
```

Example

This example code retrieves the user ID, username, and user description of the 'sa' user from the Users table. Notice that the properties are provided as a query parameter.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string properties = "UserId, Username, UserDesc";
   string filter = "Username%20LIKE%20'sa'";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/adv?props={properties}&filter={filter}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Load Property Values

The Load Property Values API is a load collection API that returns a list of values for a single property.

GET	/{jsonlist}/{ido}/{prop}/adv

http://localhost/IDORequestService/MGRESTService.svc/jsonlist/UserNames/Username/adv?filter=EditLevel%20LIKE%202

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
prop	Path	Yes	This is the name of an IDO property.
filter	Query	No	This parameter restricts the result set. This can be any valid SQL WHERE clause.
orderby	Query	No	This parameter is a comma-delimited list of properties that specifies the order in which the result set should be sorted. To have a property sorted in descending order, use the DESC keyword after the property name.
rowcap	Query	No	This parameter specifies how many records are to be retrieved in each request. These are the valid values: • `-1`- This specifies that the system default record cap is to be used. This is typically `200`, but it can be any number as set by the system administrator. • `0` - This value specifies that all records are to be retrieved, regardless of any record cap settings. • Any other positive integer specifies the number of records that are to be retrieved.
distinct	Query	No	This parameter specifies that a set of data that represents only distinct combinations of requested properties is to be returned.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

```
[
   "jdelacruz",
   "jdoe"
]
```

Example

This example code returns a list of values for the specified property.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string prop = "Username";
   string filter = "EditLevel%20LIKE%202";
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/jsonlist/{ido}/{prop}/adv?filter={filter}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

UpdateItem

The UpdateItem API inserts updates a single record in a specified IDO collection.

PUT	/{responsetype}/{ido}/updateitem
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/updateitem	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<IDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <Action>Update</Action>
  <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930] User
Names.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]</ItemId>
  <Properties>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>UserDesc</Name>
      <Value>John Doe Sr.</Value>
    </UpdateProperty>
  </Properties>
</IDOUpdateItem>
```

Request data in JSON format

```
{
   "Action": 2,
   "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]",
   "Properties": [
```

```
      {
         "IsNull": false,
         "Modified": true,
         "Name": "UserDesc",
         "Value": "John Doe Sr."
      }
   ]
}
```

Response data in XML format

```
<?xml version="1.0"?>
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Update successful.</Message>
  <MessageCode>201</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Update successful.",
   "MessageCode": 201
}
```

Example

This example code updates the description of user **jdoe**.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/updateitem";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5v1903teP0jSDwkFs" );

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe Sr.",
      IsNull = false,
      Modified = true
   };
```

```
   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]"
   };

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItem.GetType() );
      serializer.WriteObject( writer, idoItem );
   }

   // pass the IDOUpdateItem as the request data and send the update re
quest
   HttpResponseMessage response = client.PutAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
```

```
    Update = 2,
    Delete = 4
}

public enum UpdateLocking
{
    Row = 0,
    Property = 1
}
```

UpdateItem with refresh

The UpdateItem with refresh API inserts updates a single record in a specified IDO collection and then refreshes the collection.

PUT	/{responsetype}/{ido}/updateitem/adv

http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/updateitem/adv?refresh=ALL&props=UserDesc

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
refresh	Query	No	This parameter provides the instruction to refresh the collection. Supply either of these values: • **ALL** - Refreshes the entire collection and all of its properties. • **PROPS** - Refreshes only the IDO properties.
props	Query	No	This is a comma-delimited list of the properties to refresh.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.

Name	Description
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<IDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <Action>Update</Action>
  <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930] User
Names.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]</ItemId>
  <Properties>
    <UpdateProperty>
      <IsNull>false</IsNull>
      <Modified>true</Modified>
      <Name>UserDesc</Name>
      <Value>John Doe Sr.</Value>
    </UpdateProperty>
  </Properties>
</IDOUpdateItem>
```

Request data in JSON format

```
{
   "Action": 2,
   "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]",
   "Properties": [
      {
         "IsNull": false,
         "Modified": true,
         "Name": "UserDesc",
         "Value": "John Doe Sr."
      }
   ]
}
```

Response data in XML format

```
<?xml version="1.0"?>
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Update successful.</Message>
  <MessageCode>201</MessageCode>
```

```
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Update successful.",
   "MessageCode": 201
}
```

Example

This example code updates the description of user **jdoe**.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string refresh = "ALL";
   string props = "UserDesc";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/{responsetype}/{ido}/updateitem/adv?refresh={re
fresh}&props={props}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5v1903teP0jSDwkFs" );

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe Sr.",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.930]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]"
   };

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItem.GetType() );
```

```
      serializer.WriteObject( writer, idoItem );
   }

   // pass the IDOUpdateItem as the request data and send the update re
quest
   HttpResponseMessage response = client.PutAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

UpdateItems

The UpdateItems API inserts updates multiple records in a specified IDO collection.

PUT	/{responsetype}/{ido}/updateitems
http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/updateitems	

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<?xml version="1.0"?>
<ArrayOfIDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <IDOUpdateItem>
    <Action>Update</Action>
    <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280] User
Names.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>John Doe Sr.</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
```

```
  <IDOUpdateItem>
    <Action>Update</Action>
    <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960] User
Names.ID=[3faaaaab-47ef-4643-8255-756976238911]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>Will Kevin Smith</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
</ArrayOfIDOUpdateItem>
```

Request data in JSON format

```
 [
   {
      "Action": 2,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": true,
            "Name": "UserDesc",
            "Value": "John Doe Sr."
         }
      ]
   },
   {
      "Action": 2,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": true,
            "Name": "UserDesc",
            "Value": "Will Kevin Smith"
         }
      ]
   }
]
```

Response data in XML format

```
<?xml version="1.0"?>
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Update successful.</Message>
```

```
  <MessageCode>201</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Update successful.",
   "MessageCode": 201
}
```

Example

This example code updates the description of users **jdoe** and **wsmith**.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/updateitems";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe Sr.",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]"
   };
   idoItems.Add( idoItem );

   userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Will Kevin Smith",
      IsNull = false,
      Modified = true
```

```
   };

   idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]"
   };
   idoItems.Add( idoItem );

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
      serializer.WriteObject( writer, idoItems );
   }

  // pass the List<IDOUpdateItem> as the request data and send the update
 request
   HttpResponseMessage response = client.PutAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}
```

```
public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

UpdateItems with refresh

The UpdateItems with refresh API updates multiple records in a specified IDO collection and then refreshes the collection.

PUT	/{responsetype}/{ido}/updateitems/adv

http://localhost/IDORequestService/MGRESTService.svc/xml/UserNames/updateitems/adv?refresh=ALL&props=UserDesc

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json
ido	Path	Yes	This is the name of the IDO collection.
refresh	Query	No	This parameter provides the instruction to refresh the collection. Supply either of these values: • **ALL** - Refreshes the entire collection and all of its properties. • **PROPS** - Refreshes only the IDO properties.
props	Query	No	This is a comma-delimited list of the properties to re-fresh.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data in XML format

```
<?xml version="1.0"?>
<ArrayOfIDOUpdateItem
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.infor.com">
  <IDOUpdateItem>
    <Action>Update</Action>
    <ItemId>PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280] User
Names.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>John Doe Sr.</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
  <IDOUpdateItem>
    <Action>Update</Action>
    <ItemId> PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960] User
Names.ID=[3faaaaab-47ef-4643-8255-756976238911]</ItemId>
    <Properties>
      <UpdateProperty>
        <IsNull>false</IsNull>
        <Modified>true</Modified>
        <Name>UserDesc</Name>
        <Value>Will Kevin Smith</Value>
      </UpdateProperty>
    </Properties>
  </IDOUpdateItem>
</ArrayOfIDOUpdateItem>
```

Request data in JSON format

```
[
   {
```

```
      "Action": 2,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": true,
            "Name": "UserDesc",
            "Value": "John Doe Sr."
         }
      ]
   },
   {
      "Action": 2,
      "ItemId": "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]",
      "Properties": [
         {
            "IsNull": false,
            "Modified": true,
            "Name": "UserDesc",
            "Value": "Will Kevin Smith"
         }
      ]
   }
]
```

Response data in XML format

```
<?xml version="1.0"?>
<MGRestUpdateResponse
 xmlns="http://schemas.datacontract.org/2004/07/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Message>Update successful.</Message>
  <MessageCode>201</MessageCode>
</MGRestUpdateResponse>
```

Response data in JSON format

```
{
   "Message": "Update successful.",
   "MessageCode": 201
}
```

Example

This example code updates the descriptions of users **jdoe** and **wsmith**, and then returns the updated property values.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
```

```
{
   // optionally, you can use json as the response type
   string ido = "UserNames";
   string refresh = "ALL";
   string props = "UserDesc";
   string requestUrl = $"http://localhost/IDORequestService/MGRESTSer
vice.svc/xml/{ido}/updateitems/adv?refresh={refresh}&props={props}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002…5v1903teP0jSDwkFs" );

   List<IDOUpdateItem> idoItems = new List<IDOUpdateItem>();

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe Sr.",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:42:57.280]
UserNames.ID=[4b5b7da7-dd7c-47d7-8746-2ce193bfc43a]"
   };
   idoItems.Add( idoItem );

   userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Will Kevin Smith",
      IsNull = false,
      Modified = true
   };

   idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-07-24 13:28:51.960]
UserNames.ID=[3faaaaab-47ef-4643-8255-756976238911]"
   };
   idoItems.Add( idoItem );

  XDocument xdoc = new XDocument( new XDeclaration( "1.0", "utf-8", "yes"
 ) );
   using ( XmlWriter writer = xdoc.CreateWriter() )
   {
      DataContractSerializer serializer = new DataContractSerializer(
idoItems.GetType() );
```

```
      serializer.WriteObject( writer, idoItems );
   }


   // pass the List<IDOUpdateItem> as the request data and send the update
 request
   HttpResponseMessage response = client.PutAsync( requestUrl, new
StringContent( xdoc.ToString(), Encoding.UTF8, "application/xml" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

Upload File Stream

The Upload File Stream API uploads a streamed file to an IDO collection. The IDO collection must have a property for storing binary data.

POST	/io/{responsetype}/uploadfile
http://localhost/IDORequestService/MGRESTService.svc/xml/io/xml/uploadfile	

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 147.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.
X-IdoName	This is the name of the IDO collection.
X-Property	This is the name of the IDO property.
X-RowPointer	This is the value of the IDO row pointer.

Request data

The request object is an unencoded binary stream from a file.

Response data in XML format

```
<MGRestIOResponse>
  <Success>true</Success>
  <Message>
    Initializing file save;
```

```
    Updating BLOB to IDO;
    Updated item count: 1;
    Upload complete;
  </Message>
  <MessageCode>700</MessageCode>
</MGRestIOResponse>
```

Response data in JSON format

```
{
   "Message": "\"Initializing file save",
   "Updating" "BLOB",
   "to" "IDO",
   "Updated" "item",
   "count": 1,
   "Upload" "complete\",",
   "MessageCode": 700,
   "Success": true
}
```

Example

This example code uploads a user image to the users table.

```
string xml = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/io/xml/uploadfile";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   client.DefaultRequestHeaders.Add( "X-IdoName", "UserNames" );
   client.DefaultRequestHeaders.Add( "X-Property", "UserImage" );
   client.DefaultRequestHeaders.Add( "X-RowPointer", "2807a627-577b-462e-
9494-aee568152c54" );

   // select an image or a file and include it as the request payload
   OpenFileDialog dialog = new OpenFileDialog();
   byte[] file = new byte[] { };

   if ( dialog.ShowDialog() == DialogResult.OK )
   {
      file = File.ReadAllBytes( dialog.FileName );
   }

   // pass the file as the request data and send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
```

```
ByteArrayContent( file ) ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

Upload Document Object

The Upload Document Object API uploads a streamed file to an IDO document object.

POST	/io/{responsetype}/uploaddocobj
http://localhost/IDORequestService/MGRESTService.svc/xml/io/xml/uploaddocobj	

Note: This topic is for REST Version 1. There is also an API for REST Version 2 on page 145.

Parameters

Name	In	Required?	Description
responsetype	Path	Yes	Response request data format: Supply any of these values: • xml • json

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.
X-IdoName	This is the name of the IDO collection.
X-RowPointer	This is the value of the IDO row pointer.
X-DocName	This is the name of the document object.
X-DocExt	This is the file extension used for the document object.
X-DocDesc	This is the description of the document object.

Request data

The request object is an unencoded binary stream of a file.

Response data in XML format

```
<MGRestIOResponse>
  <Success>true</Success>
  <Message>
    Match count: 6
    Updated item count: 1
  </Message>
  <MessageCode>700</MessageCode>
</MGRestIOResponse>
```

Response data in JSON format

```
{
   "Message": "Match count: 6 Updated item count: 1",
   "MessageCode": 700,
   "Success": true
}
```

Example

This example code uploads a file and attaches it as a document object of the referenced IDO and IDO row pointer.

```
string xml = string.Empty;

using ( var client = new HttpClient() )
{
   // optionally, you can use json as the response type
   string requestUrl = $"http://server/IDORequestService/MGRESTSer
vice.svc/io/xml/uploaddocobj";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   client.DefaultRequestHeaders.Add( "X-IdoName", "UserNames" );
   client.DefaultRequestHeaders.Add( "X-RowPointer", "2807a627-577b-462e-
9494-aee568152c54" );
   client.DefaultRequestHeaders.Add( "X-DocName", "WeeklyReport" );
   client.DefaultRequestHeaders.Add( "X-DocExt", "docx" );
   client.DefaultRequestHeaders.Add( "X-DocDesc", "Weekly Report" );

   // select an image or a file and include it as the request payload
   OpenFileDialog dialog = new OpenFileDialog();
   byte[] file = new byte[] { };
```

```
   if ( dialog.ShowDialog() == DialogResult.OK )
   {
      file = File.ReadAllBytes( dialog.FileName );
   }

   // pass the file as the request data and send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
ByteArrayContent( file ) ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      xml = result.Result;
   }
}
```

REST API, Version 2

This section presents the updated version of the Mongoose REST API.

Base URL

All endpoints are accessible using HTTP(S) and are located at this URL:

`http://<serverName>/IDORequestService/ido`

API documentation

Swagger API documentation is available for the Mongoose REST API. It fully describes each API operation that includes both required and optional parameters, Request Data examples, and so on. You can access it using this endpoint:

`http://<serverName>/IDORequestService/ido/api-docs`

The Swagger API documentation can be transformed into a rich UI that allows you to browse through the API and try out some of the operations.

Content type

Each endpoint requires the request and response data to either be empty or JSON-formatted. No other content types are supported.

Mongoose REST Version 2 API endpoints

These are the Mongoose REST Version 2 API endpoints:

- Download Document Object

- Download File Stream
- Fire AES Event
- Get Configurations
- Get Document Objects
- Get Security Token
- Get Property Information
- LoadCollection
- Invoke IDO Method
- UpdateCollection
- Upload File Stream
- Upload Document Object

All API calls, except GetConfigurations and GetSecurityToken APIs, are required to include an Authorization header containing either a Mongoose security token or a valid OAuth 2.0 bearer token, depending on how the API is being called.

We recommend that you use the prototypes presented here as examples when creating your API calls. All these prototype examples use C# code.

DownloadDocumentObject

The DownloadDocumentObject API downloads a streamed file from an IDO document object.

GET	/docobj/{ido}

```
http://localhost/IDORequestService/ido/docobj/UserNames?rowPointer=8c66c936-d348-4c13-8684-69bc6fa10473&name=WeeklyReport&refseq=4
```

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 66.

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
rowPointer	Query	Yes	This is the value of an IDO row pointer.
name	Query	No	This is the name of the document object. **Note:** This parameter is optional if the **refseq** parameter is used.
refseq	Query	No	This is the reference sequence number of the document object. **Note:** This parameter is optional if the **name** parameter is used.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

Unencoded binary stream of the file, with content type of “application/octet-stream”.

Example

This example code downloads a document object from the referenced IDO and IDO rowpointer.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string rowpointer = "4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c";
   string name = "WeeklyReport";
   string requestUrl = $"http://server/IDORequestService/ido/do
cobj/{ido}?rowPointer={rowpointer}&name={name}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

DownloadFileStream

The DownloadFileStream API downloads a streamed file from an IDO collection. The IDO collection must have a property for storing binary data.

GET	/file/{ido}

```
http://localhost/IDORequestService/ido/file/
   UserNames?property=UserImage&rowpointer=4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c
```

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 68.

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
property	Query	Yes	This is the name of an IDO property for storing binary data.
rp	Query	Yes	This is the value of an IDO row pointer.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

Unencoded binary stream of the file, with content-type of “application/octet-stream”.

Example

This example code downloads a user image from the users table.

```
string json = string.Empty;
```

```
using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string property = "UserImage";
   string rowpointer = "4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c";
   string requestUrl = $"http://server/IDORequestSer
vice/ido/file/{ido}?property={property}&rowPointer={rowpointer}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

FireAESEvent

The FireAESEvent fires an Application Event System (AES) event.

POST	/aes/{eventname}
http://localhost/IDORequestService/ido/aes/DivideNumbers	

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 70.

Parameters

Name	In	Required?	Description
eventname	Path	Yes	This is the name of the AES event.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.

Name	Description
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

```
[
      {
         "Name": "Num1",
         "Return": false,
         "Value": 120
      },
   {
      "Name": "Num2",
      "Return": false,
      "Value": 5
   },
      {
         "Name": "Quotient",
         "Return": true
      }
]
```

Response data

```
{
  "Message": null,
  "Success": true,
  "EventName": "DivideNumbers",
  "Parameters": [
    {
      "Name": "Quotient",
      "Value": "1",
      "Return": true
    },
   {
      "Name": "Infobar",
      "Value": "success",
      "Return": true
   }
  ]
}
```

Example

This example code executes an AES event that divides two numbers and returns the quotient as an output parameter.

```
string json = string.Empty;
```

```
using ( HttpClient client = new HttpClient() )
{
   string eventname = "DivideNumbers";
   string requestUrl = $"http://server/IDORequestService/ido/aes/{event
name}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   AESEventParameter num1 = new AESEventParameter
   {
      Name = "Num1",
      Return = false,
      Value = "4"
   };

   AESEventParameter num2 = new AESEventParameter
   {
      Name = "Num2",
      Return = false,
      Value = "4"
   };

   AESEventParameter[] parameters = new[] { num1, num2 };

   // pass the array of parameters as the request data
   string contentStr = JsonConvert.SerializeObject( parameters );

   // send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl.ToString(),
 new StringContent( contentStr, Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

This class was used with the foregoing code snippet:

```
public class AESEventParameter
{
   public string Name { get; set; }
   public string Value { get; set; }
   public bool Return { get; set; }
}
```

GetConfigurations

The GetConfigurations API returns a list of the application configuration names available on the server.

GET	/configurations
http://localhost/IDORequestService/ido/configurations?configgroup=DEV	

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 76.

Parameters

Name	In	Required?	Description
configgroup	Query	No	This is the name of a Configuration Group. Use this parameter to get the configurations from a specific configuration group.

Headers

None

Request data

None

Response data

```
{
   "Message": null,
   "Success": true,
   "Configurations": [
      "CSI_DALS",
      "CSI_EMEA",
      "CSI_LA"
   ]
}
```

Example

This example code retrieves an array of configuration names from a specified configuration group using the ConfigServer.aspx page in IDORequestService.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string configGroup = "DEV";
```

```
   string requestUrl = $"http://server/IDORequestService/ido/configura
tions?configgroup={configGroup}";

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // get the response containing the configuration list
      json = result.Result;
   }
}
```

GetDocumentObjects

The GetDocumentObjects API returns a list of document objects.

GET	/docobj/list

http://localhost/IDORequestService/ido/docobj/list?ido=UserNames

Parameters

Name	In	Required?	Description
ido	Query	No	Thisis the name of the IDO collection that contains the document objects.
rowPointer	Query	No	This is the value of the IDO row pointer.
docName	Query	No	This is the name of the document object to retrieve.
docExt	Query	No	This is the file extension used for the document object to be retrieved.
refSeq	Query	No	This is the reference sequence of the document object to retrieve.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.

Name	Description
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data

```
{
  "Documents": [
   {
      "TableName": "Demo_Products",
      "TableRowPointer": "d629ebd6-974c-4969-be2e-145dd8d84da8",
      "DocumentName": "ultimate_choco_chip_cookies",
      "Description": null,
      "DocumentObject": "/9j/4AAQSkZJRgABAQAASABIAAD…Z",
      "DocumentExtension": "jpg",
      "MediaType": "image/jpeg",
      "RefSequence": "1",
      "_ItemId": "PBT=[DocumentObjectAndRefView] doc.ID=[0c4b201a-5ecd-
4bf9-a242-86bf4c177727] doc.DT=[2018-10-12 09:52:12.153]"
   },
    {
      "TableName": "DemoSubmissionComments",
      "TableRowPointer": "67d4b6c8-3b67-4269-8f4f-7d7a7ee81f71",
      "DocumentName": "ayala_cinema_tickets_1508950029_6fccded3",
      "Description": null,
      "DocumentObject": "/9j/4AAQSkZJRgABAQAASABIAAD…Z",
      "DocumentExtension": "jpg",
      "MediaType": "image/jpeg",
      "RefSequence": "1",
      "_ItemId": "PBT=[DocumentObjectAndRefView] doc.ID=[d84729d2-8023-
4624-8f02-5598def50073] doc.DT=[2018-10-12 10:07:34.433]"
    }
  ],
  "Success": true,
  "Message": null
}
```

Example

This example code retrieves a list of document objects attached to the UserNames IDO.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/ido/do
```

```
cobj/list?ido={ido}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

GetPropertyInformation

The GetPropertyInformation API returns a list of properties and their attributes from an IDO collection.

GET	/info/{ido}

http://localhost/IDORequestService/ido/info/UserNames

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.

Headers

Name	Description
Authorization	If the API is called directly, then the Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through the ION API, then a valid OAuth2.0 bearer token is provided by the ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data


```
{
    "Message": null,
    "Success": false,
   "Keys": [
      "UserId"
   ],
    "Properties": [
        {
            "BooleanFalseValue": "",
            "BooleanTrueValue": "",
            "CaseFormat": "",
            "ClrTypeName": "Decimal",
            "ColumnDataType": "TokenType",
            "DataType": "Decimal",
            "DateFormat": "",
            "DecimalPos": 0,
            "DefaultIMECharset": "",
            "DefaultValue": "",
            "DomainIDOName": "",
            "DomainListProperties": "",
            "DomainProperty": "",
            "InputMask": "",
            "IsItemWarnings": false,
            "JustifyFormat": "R",
            "LabelStringID": "sUserID",
            "Length": 11,
            "Name": "UserId",
            "PropertyClass": "Token",
            "RORecord": false,
            "ReadOnly": true,
            "Required": true
        },
        { ... }
    ],
    "SubCollections": [
        {
            "IDOName": "MGCore.UserGroupMaps",
            "LinkBy": [
                {
                    "Child": "UserId",
                    "Parent": "UserId"
                }
            ],
            "Name": "UserGroupMaps"
        },
        {  ... }
    ]
}
```

Example

This example code retrieves an array of properties and their attributes from a specified IDO.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/ido/info/{ido}";


   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

GetSecurityToken

The GetSecurityToken API returns a Mongoose security token for a specific user, which can be used to authenticate requests when calling the API directly.

GET	/token/{config}/{username}/{password}
http://localhost/IDORequestService/ido/token/CSI_DALS/sa/Passwe1rd	

Note: This topic is for REST Version 1. There is also an API for REST Version 1 on page 78.

Parameters

Name	In	Required?	Description
config	Path	Yes	This is the name of a configuration available on the application server.
username	Path	Yes	This is the username for the Mongoose user.
password	Path	No	This is the password for the Mongoose user.

Headers

None

Request data

None

Response data

```
{
   "Message": null,
   "Success": true,
   "Token": "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…=="
}
```

Example

This example code retrieves a Mongoose security token that can be used in succeeding requests.

```
string json = string.Empty;

using ( var client = new HttpClient() )
{
   string config = "MG_DEV";
   string username = "sa";
   string password = string.Empty;
   string requestUrl = $"http://server/IDORequestService/ido/token/{con
fig}/{username}/{password}";

   // send the get request
   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      // get the response containing the token
      json = result.Result;
   }
}
```

InvokeIDOMethod

The InvokeIDOMethod API returns a list of properties and their attributes from an IDO collection.

POST	/invoke/{ido}
http://localhost/IDORequestService/invoke/UserNames?method=GetUserAttributes	

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
method	Query	Yes	This is the name of the IDO method.

Headers

Name	Description
Authorization	If the API is called directly, then the Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through the ION API, then a valid OAuth2.0 bearer token is provided by the ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

```
[
   "sa",
   null,
   null,
   null,
   null
]
```

Response data

```
{
   "Message": null,
   "Success": true,
   "Parameters": [
      "sa",
      "4",
      "1",
      "CoreFormsAdmin",
      ""
   ],
   "ReturnValue": "0"
}
```

Example

This example code determines the user attributes of user **sa** using the IDO method GetUserAttributes.

```
string json = string.Empty;
```

```
using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string method = "GetUserAttributes";
   string requestUrl = $"http://server/IDORequestService/ido/in
voke/{ido}?method={method}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   string[] parameters = new[] { "sa", "", "", "", "" };

   // pass the array of parameters as the request data
   string contentStr = JsonConvert.SerializeObject( parameters );

   // send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl.ToString(),
 new StringContent( contentStr, Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

LoadCollection

The LoadCollection API is a basic load collection API that returns a set of records from an IDO collection.

GET	/load/{ido}
http://localhost/IDORequestService/ido/load/UserNames?properties=UserId,Username,UserDesc	

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 84.

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
properties	Query	No	This is a comma-delimited list of properties for which to return values. You can also use an asterisk (*) to include all properties except subcollection properties. If this parameter is excluded or left blank, the server retrieves data with all the parameters.

Name	In	Required?	Description
filter	Query	No	This parameter restricts the result set. Any valid SQL WHERE clause syntax is allowed.
orderby	Query	No	This is a comma-delimited list of properties that specifies how the result set is to be sorted. Use the DESC keyword after a property name to sort that property in descending order.
recordcap	Query	No	Use this parameter to determine how many records are to be retrieved in one request. These are the valid values: `-1` returns 200 records. `0` returns all records. Any other number specifies the number of records to be retrieved.
distinct	Query	No	This parameter specifies that the set of data to be returned must represent only distinct combinations of properties.
clm	Query	No	This is the name of a custom load method. This parameter works in conjunction with the clmparam parameter.
clmparam	Query	No	This is a comma-delimited list of custom load method parameters.
loadtype	Query	No	This parameter is used for load collection paging. This parameter is used in conjunction with the bookmark parameter. These are the valid values: • FIRST • NEXT • PREV • LAST

Name	In	Required?	Description
bookmark	Query	No	This parameter is for the bookmark ID. Bookmark IDs serve as a reference when you want to get to the next or previous records in a collection. The “bookmark” value uses this format: <B><P> UserId </P><D><f>false</f></D><F><v>1</v></F><L><v>2</v></L></B> where: • <B></B> is the bookmark envelope tag. • <P></P> is the property name of the primary key. • <D></D> is a flag that, if TRUE, specifies that records are to be read in descending order. • <f></f> is the value of the <D></D> element (TRUE or FALSE). • <F></F> is the first row in the collection. • <L></L> is the last row in the collection. • <v><v> is the value of the row for the given property name. For an example of this, see Example: Bookmark IDs in LoadCollection responses on page 174.
pqc	Query	No	This parameter specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the Design Mode PQ option in Load/Save Overrides and uses the same syntax.
readonly	Query	No	When set to TRUE, this parameter specifies that retrieved records are not to include the _ItemID property, which is substantial for update and delete operations.

Headers

Name	Description
Authorization	If the API is called directly, then the Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through the ION API, then a valid OAuth2.0 bearer token is provided by the ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

None

Response data


```
{
  "Items": [
   {
      "UserId": "1",
      "Username": "sa",
      "UserDesc": "System Admin"
   },
    {
      "UserId": "2",
      "Username": "jdoe",
      "UserDesc": "John Doe"
    }
  ],
  "Bookmark": "<B><P><p>CollectionName</p><p>Development
Flag</p></P><D><f>false</f><f>false</f></D><F><v>ABDataSer
vices</v><v>0</v></F><L><v>ABProjectRoles</v><v>0</v></L></B>",
  "Success": true,
  "Message": null
}
```


Example

This example code retrieves records that contain the user ID, username, and user description from the Users table.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestSer
vice/ido/load/{ido}?properties=UserId,Username,UserDesc";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   HttpResponseMessage response = client.GetAsync( requestUrl ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

UpdateCollection

The UpdateCollection API is used to insert, update, or delete one or more records from an IDO colleciton.

POST	/update/{ido}

http://localhost/IDORequestService/ido/update/UserNames?refresh=true

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
refresh	Query	No	This parameter instructs the system to refresh the collection after the update is complete. Valid values are **TRUE** or **FALSE**.

Headers

Name	Description
Authorization	If the API is called directly, then the Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through the ION API, then a valid OAuth2.0 bearer token is provided by the ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data for an Insert action

```
{
  "Changes": [
   {
      "Action": 1,
      "ItemId": "PBT=[UserNames]",
      "Properties": [
         {
            "Name": "Username",
            "Value": "jdelacruz",
            "Modified": true,
            "IsNull": false
         },
         {
            "Name": "UserDesc",
            "Value": "Juan Dela Cruz",
            "Modified": true,
            "IsNull": false
         }
      ],
      "UpdateLocking": 1
```

```
    }
  ]
}
```

Request data for an Update action

```
{
   "Changes": [
      {
         "Action": 2,
        "ItemId": "PBT=[UserNames] UserNames.DT=[2018-10-02 15:39:02.060]
 UserNames.ID=[4c9a96d0-ba3c-4de4-8657-6d262f9dcd3f]",
         "Properties": [
            {
               "Name": "UserDesc",
               "Value": "John Doe Sr.",
               "Modified": true,
               "IsNull": false
            }
         ],
         "UpdateLocking": 1
      }
   ]
}
```

Request data for a Delete action

```
{
   "Changes": [

      {
         "Action": 4,
        "ItemId": "PBT=[UserNames] UserNames.DT=[2018-12-14 13:52:27.737]
 u.ID=[265df7d2-802c-4582-a774-a81b675a91a1]"
      }
   ]
}
```

Response data for an Insert action

```
{
    "Message": null,
    "Success": true,
    "RefreshItems": [
      {
         "Action": 1,
        "ItemId": "PBT=[UserNames] UserNames.DT=[2019-05-03 13:24:07.080]
 UserNames.ID=[dd896b24-e1dd-47df-ad4f-f0378243c202]",
         "ItemNo": 0,
         "Properties": [
            {
               "IsNestedCollection": false,
               "IsNull": false,
```

```
                "Modified": false,
                "Name": "Username",
                "OriginalValue": null,
                "Value": "jdelacruz"
             },
             {
                "IsNestedCollection": false,
                "IsNull": false,
                "Modified": false,
                "Name": "UserDesc",
                "OriginalValue": null,
                "Value": "Juan Dela Cruz"
             }
          ],
          "UpdateLocking": 0
       }
    ]
}
```


Response data for an Update action


```
{
   "Message": null,
   "Success": true,
   "RefreshItems": [
      {
         "Action": 2,
        "ItemId": "PBT=[UserNames] UserNames.DT=[2019-05-03 13:24:07.080]
 UserNames.ID=[dd896b24-e1dd-47df-ad4f-f0378243c202]",
         "ItemNo": 0,
         "Properties": [
            {
               "IsNestedCollection": false,
               "IsNull": false,
               "Modified": false,
               "Name": "UserDesc",
               "OriginalValue": null,
               "Value": "John Doe Sr."
            }
         ],
         "UpdateLocking": 0
      }
   ]
}
```


Response data for a Delete action


```
{
   "Message": null,
   "Success": true,
   "RefreshItems": null
}
```

Example 1 - Insert using UpdateCollection

This example code inserts **jdelacruz** as a new user.

```
string json = string.Empty;

using ( var client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/ido/up
date/{ido}?refresh=true";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   UpdateProperty username = new UpdateProperty
   {
      Name = "Username",
      Value = "jdelacruz",
      IsNull = false,
      Modified = true
   };

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "Juan Dela Cruz",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Insert,
      Properties = new[] { username, userDesc },
      ItemId = "PBT=[UserNames]"
   };

  UpdateCollectionRequest request = new UpdateCollectionRequest { Changes
 = new[] { idoItem } };

   // pass the UpdateCollectionRequest as the request data
   string contentStr = JsonConvert.SerializeObject( request );

   // send the post request
  HttpResponseMessage response = client.PostAsync( requestUrl.ToString(),
 new StringContent( contentStr, Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
```

```
    }
}
```

Example 2 - Update using UpdateCollection

This example code updates the description of user **jdoe**.

```
string json = string.Empty;

using ( var client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/ido/up
date/{ido}?refresh=true";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   UpdateProperty userDesc = new UpdateProperty
   {
      Name = "UserDesc",
      Value = "John Doe Sr.",
      IsNull = false,
      Modified = true
   };

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Update,
      Properties = new[] { userDesc },
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-05-03 13:24:07.080]
UserNames.ID=[f33ef708-19bb-4ab9-aade-88a6e8853742]"
   };

  UpdateCollectionRequest request = new UpdateCollectionRequest { Changes
 = new[] { idoItem } };

   // pass the UpdateCollectionRequest as the request data
   string contentStr = JsonConvert.SerializeObject( request );

   // send the post request
  HttpResponseMessage response = client.PostAsync( requestUrl.ToString(),
 new StringContent( contentStr, Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

Example 3 - Delete using UpdateCollection

This example code deletes the user with the given _ItemID property value.

```
string json = string.Empty;

using ( var client = new HttpClient() )
{
   string ido = "UserNames";
   string requestUrl = $"http://server/IDORequestService/ido/up
date/{ido}?refresh=true";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   IDOUpdateItem idoItem = new IDOUpdateItem
   {
      Action = UpdateAction.Delete,
      ItemId = "PBT=[UserNames] UserNames.DT=[2019-05-03 13:24:07.080]
UserNames.ID=[dd896b24-e1dd-47df-ad4f-f0378243c202]"
   };

  UpdateCollectionRequest request = new UpdateCollectionRequest { Changes
 = new[] { idoItem } };

   // pass the UpdateCollectionRequest as the request data
   string contentStr = JsonConvert.SerializeObject( request );

   // send the post request
  HttpResponseMessage response = client.PostAsync( requestUrl.ToString(),
 new StringContent( contentStr, Encoding.UTF8, "application/json" ) ).Re
sult;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

You can use these classes to construct the Request Data as demonstrated in the previous code snippets:

```
public class UpdateCollectionRequest
{
   public IDOUpdateItem[] Changes { get; set; }
   public bool RefreshAfterSave { get; set; }
   public string CustomInsert { get; set; }
   public string CustomUpdate { get; set; }
   public string CustomDelete { get; set; }
}
```

```
public class IDOUpdateItem
{
   public UpdateAction Action { get; set; }
   public string ItemId { get; set; }
   public int ItemNo { get; set; }
   public UpdateProperty[] Properties { get; set; }
   public UpdateLocking UpdateLocking { get; set; }
}

public class UpdateProperty
{
   public string Name { get; set; }
   public string Value { get; set; }
   public string OriginalValue { get; set; }
   public bool Modified { get; set; }
   public bool IsNull { get; set; }
}

public enum UpdateAction
{
   Insert = 1,
   Update = 2,
   Delete = 4
}

public enum UpdateLocking
{
   Row = 0,
   Property = 1
}
```

UploadDocumentObject

The UploadDocumentObject API uploads a streamed file to an IDO document object.

POST	/docobj/{ido}

```
http://localhost/IDORequestService/ido/docobj/UserNames?rowPointer=8c66c936-d348-4c13-8684-69bc6fa10473&name=WeeklyReport&desc=Weekly%20Report&ext=xls
```

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 118.

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the referenced IDO collection.
itemid	Query	Yes	This is the value of the _ItemID property for the document object.

Name	In	Required?	Description
rowPointer	Query	No	This is the value of the referenced IDO row pointer. **Note:** Use this parameter to upload a new document object and link it to the referenced IDO and row pointer.
name	Query	No	This is the name of the document object.
desc	Query	No	This is the description of the document object.
ext	Query	No	This is the file extension for the document objects.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.

Request data

Unencoded binary stream of the file.

Response data

```
{
   "Message": null,
   "Success": true,
   "ItemId": "PBT=[DocumentObjectAndRefView] doc.ID=[7018e6d1-7397-466a-
be43-17c52d51c3d6] doc.DT=[2018-11-22 09:30:16.643]"
}
```

Example

This example code uploads a file and attaches it as a document object of the referenced IDO and IDO row pointer.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string rowpointer = "4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c";
   string requestUrl = $"http://server/IDORequestService/ido/do
cobj/{ido}?rowpointer={rowpointer}&name=WeeklyReport&desc=Weekly%20re
port%20for%20this%20week&ext=docx";
```

```
   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // select an image or a file and include it as the request payload
   OpenFileDialog dialog = new OpenFileDialog();
   byte[] file = new byte[] { };

   if ( dialog.ShowDialog() == DialogResult.OK )
   {
      file = File.ReadAllBytes( dialog.FileName );
   }

   // pass the file as the request data and send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
ByteArrayContent( file ) ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

UploadFileStream

The UploadFileStream API uploads a streamed file to an IDO collection. The IDO collection must have a property for storing binary data.

POST	/file/{ido}

```
http://localhost/IDORequestService/ido/file/
   UserNames?property=UserImage&itemid=PBT=[UserNames] UserNames.DT=[2018-09-05
   11:01:00.167] UserNames.ID=[7d38e4fb-e1a8-4af7-936b-50accee5d35b]
```

Note: This topic is for REST Version 2. There is also an API for REST Version 1 on page 116.

Parameters

Name	In	Required?	Description
ido	Path	Yes	This is the name of the IDO collection.
property	Query	Yes	This is the name of the IDO property to be used for storing binary data.
itemid	Query	Yes	This is the value of the _itemID property.

Headers

Name	Description
Authorization	If the API is called directly, then a Mongoose security token is obtained through a call to the GetSecurityToken API. If the API is called through ION API, then a valid OAuth2.0 bearer token is provided by ION API.
X-Infor-MongooseConfig	This is the name of a configuration that is available on the application server. This is required only when using the Mongoose API through the ION API.
X-IdoName	This is the name of the IDO collection.
X-Property	This is the name of the IDO property.
X-RowPointer	This is the value of the IDO row pointer.

Request data

The request data is an unencoded binary stream of the file.

Response data

```
{
   "Message": null,
   "Success": true
}
```

Example

This example code uploads a user image to the Users table.

```
string json = string.Empty;

using ( HttpClient client = new HttpClient() )
{
   string ido = "UserNames";
   string property = "UserImage";
   string itemid = "PBT=[UserNames] UserNames.DT=[2019-05-29 14:12:02.000]
 UserNames.ID=[4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c]";
   string requestUrl = $"http://server/IDORequestSer
vice/ido/file/{ido}?property={property}&itemid={itemid}";

   // provide token in the Authorization header
   client.DefaultRequestHeaders.TryAddWithoutValidation(
      "Authorization",
      "b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQj…==" );

   // select an image or a file and include it as the request payload
   OpenFileDialog dialog = new OpenFileDialog();
   byte[] file = new byte[] { };
```

```
   if ( dialog.ShowDialog() == DialogResult.OK )
   {
      file = File.ReadAllBytes( dialog.FileName );
   }

   // pass the file as the request data and send the post request
   HttpResponseMessage response = client.PostAsync( requestUrl, new
ByteArrayContent( file ) ).Result;

   using ( HttpContent content = response.Content )
   {
      Task<string> result = content.ReadAsStringAsync();
      json = result.Result;
   }
}
```

Chapter 4: Using the SOAP web service

An alternative to the REST web service is the SOAP web service.

When to use a SOAP web service

The SOAP web service is maintained primarily for backward compatibility, as it predates the REST API.

The SOAP web service allows you to access IDOs over the internet and perform IDO-related operations such as loading and updating IDO collections or running IDO methods. The SOAP web service accesses Mongoose through the IDO Request Interface, using the IDO Request XML schema.

Creating SOAP API calls

SOAP web service calls are typically created using Visual Studio.

Perform this procedure using Visual Studio:

Note: The procedure for creating a SOAP web service call assumes that you have added a reference to your project for the IDO .NET web service, as follows.

1. Perform one of these actions:
   - Right-click the **References** node in the **Solution Explorer**.
   - From the **Project** menu, select **Add Service Reference**.
2. Provide the URL for your .NET web service, using this syntax:

   `http://server/IDORequestService/IDOWebService.asmx`

   where *server* is the name of DNS or IP address of the server that hosts your web service.
3. Set the **Namespace** to **`IDOWebServiceReference`**.
4. Click **OK**.

You can redirect calls at runtime (for instance, to a production server rather than a test server) by passing two parameters to the constructor, for example:

```
IDOWebServiceReference.DOWebServiceSoapClient idoWS = new
```

```
IDOWebServiceReference.DOWebServiceSoapClient( "IDOWebServiceSoap",
"http://other_server_name/IDORequestService/IDOWebService.asmx" );
```

Note: You can add a query parameter to the .NET web service URL to specify the configuration group, which applies to the GetConfigurationNames web service API; for example: `http://server / IDORequestService/IDOWebService.asmx?configgroup=`*`group-name`*

where *group-name* corresponds to the name of a configuration group created with the Configuration Manager utility.

When creating your SOAP web service API calls, we recommend that you use the prototypes in this document as examples. All the prototype examples are given in C#.

SOAP web service methods

This section lists and describes the SOAP web service methods available for Mongoose applications integrating with external applications.

GetConfigurationNames

Use the GetConfigurationNames method to retrieve a list of the Mongoose configurations that are available to access from an external application.

Syntax

```
public string[] GetConfigurationNames()
```

Parameters

None

Output

Returns a list of valid configuration names from the Default configuration group.

Example

```
IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();
string[] configs = soapClient.GetConfigurationNames();
```

CreateSessionToken

The CreateSessionToken method validates the username, password, and configuration, and then returns a session token that allows the user to open a session when any of the other web service methods (LoadDataSet, SaveDataSet, and so on) are called.

Syntax

```
public string CreateSessionToken( string strUserId,
                                  string strPswd,
                                  string strConfig )
```

Parameters

Name	Description
strUserId	The Mongoose user's username
strPswd	The Mongoose user's password
strConfig	The name of the application's configuration

Output

Returns a session token string, required by other web service methods.

This is an example of a session token:

```
b/XdI6IQzCviZOGJ0E+002DoKUFOPmVDkwpQDbQjm3w/qkdxDUzmqvSYEZDCmJGWpA23OTlhF
pxRHF
z3WOsvay8V58XdIp/UIsr5TpCdMwtW3QXF2ah
wQYp2O6GzKlJcx50PzAY5KGW7CHLvMml26H13iSRJ
vZB6an1hQXrBH191JCq6PYlPtGvQUiUMGCjVqzqgXHqaK58T6NkJfMbJv52jB1DyRTv
toshS5RGj1Q
VVPCtuuvkG3O659gM2Q+lFivNleD+erIHoiQFvS1MelxKd5L5fH4PQrZ0RTnIln+AfwmOWxXQ
Tadcq
0NsVR8588DWj/QaqIr/usIVqp4tWMg==
```

Example

```
IDOWebService.DOWebServiceSoapClient idoSoapClient = new
IDOWebService.DOWebServiceSoapClient();
string token = idoSoapClient?.CreateSessionToken( "jdoe", "sJKam67s",
"CSI_DALS" );
```

LoadDataSet

The LoadDataSet method uses the LoadCollection method of an IDO to query either an IDO collection or a database table and return the results to the user.

Syntax

```
public DataSet LoadDataSet( string strSessionToken,
                            string strIDOName,
                            string strPropertyList,
                            string strFilter,
                            string strOrderBy,
                            string strPostQueryMethod,
                            int iRecordCap )
```

Parameters

Name	Description
strSessionToken	This is the session token obtained through a call to the CreateSessionToken API.
strIDOName	This is the name of the IDO collection.
strPropertyList	This is a comma-delimited list of the IDO properties.
strFilter	This parameter allows you to filter the results of the operation. You can use ny valid SQL WHERE clause syntax.
strOrderBy	This is a comma-delimited list of properties that govern the order in which the result set should be sorted. To sort a given property in descending order, use the DESC keyword after the property name.
strPostQueryMethod	This parameter specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the PQ option in Load/Save Overrides and uses the same syntax.
iRecordCap	This parameter determines how many records are to be retrieved in one request. Values are: • `-1` – 200 records are retrieved. • `0` – All records are retrieved. • Any other positive integer – The specified number of records are retrieved.

Output

Returns a System.Data.DataSet that contains the results of the query. The returned data can be filtered by any IDO-level filters. For information about IDO filters, see the online help.

This is an example output DataSet:

```
<DataSet xmlns="http://frontstep.com/IDOWebService">
  <xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" id="UserNames">
    <xs:element name="UserNames" msdata:IsDataSet="true" msdata:UseCurrent
Locale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="IDO">
            <xs:complexType>
              <xs:sequence>
              <xs:element name="UserId" type="xs:decimal" minOccurs="0"/>

                <xs:element name="Username" type="xs:string" minOc
curs="0"/>
                <xs:element name="UserDesc" type="xs:string" minOc
curs="0"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
    <UserNames xmlns="">
      <IDO diffgr:id="IDO1" msdata:rowOrder="0">
        <UserId>1</UserId>
        <Username>sa</Username>
        <UserDesc>WinStudio Admin user</UserDesc>
      </IDO>
      <IDO diffgr:id="IDO2" msdata:rowOrder="1">
        <UserId>2</UserId>
        <Username>jdelacruz</Username>
        <UserDesc>Juan Dela Cruz</UserDesc>
      </IDO>
    </UserNames>
  </diffgr:diffgram>
</DataSet>
```

Example

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
string ido = "UserNames";
string properties = "UserId, Username, UserDesc, _ItemID";
string filter = string.Empty;
string orderBy = "UserId";
string postQueryMethod = string.Empty;
int recordCap = -1;

IDOWebService.DOWebServiceSoapClient soapClient = new
```

```
IDOWebService.DOWebServiceSoapClient();
DataSet users = soapClient.LoadDataSet( sessionToken, ido, properties,
filter, orderBy, postQueryMethod, recordCap );
```

LoadJson

The LoadJson method uses the LoadCollection method of an IDO to query either an IDO collection or a database table and return the results to the user in JSON format.

Syntax

```
public string LoadJson( string strSessionToken,
                        string strIDOName,
                        string strPropertyList,
                        string strFilter,
                        string strOrderBy,
                        string strPostQueryMethod,
                        int iRecordCap )
```

Parameters

Name	Description
strSessionToken	This is the session token obtained through a call to the CreateSessionToken API.
strIDOName	This is the name of the IDO collection.
strPropertyList	This is a comma-delimited list of the IDO properties.
strFilter	This parameter allows you to filter the results of the operation. You can use ny valid SQL WHERE clause syntax.
strOrderBy	This is a comma-delimited list of properties that govern the order in which the result set should be sorted. To sort a given property in descending order, use the DESC keyword after the property name.
strPostQueryMethod	This parameter specifies a method to execute once for each row in the result set after the query is completed. This is the equivalent of the PQ option in Load/Save Overrides and uses the same syntax.
iRecordCap	This parameter determines how many records are to be retrieved in one request. Values are: • `-1` – 200 records are retrieved. • `0` – All records are retrieved. • Any other positive integer – The specified number of records are retrieved.

Output

Returns a JSON string containing the results of the query. The returned data can be filtered by any IDO-level filters. For information about IDO filters, see the online help.

This is an example of JSON output:


```
{
  "IDOName": "UserNames",
  "Items": [
   {
      "EditStatus": 0,
      "ID": "PBT=[UserNames] UserNames.DT=[2019-05-03 13:24:07.080] User
Names.ID=[2bebf824-6b0f-4725-9150-da7303cc86ec]",
      "Properties": [
         {
            "Property": "11",
            "Updated": false
         },
         {
            "Property": "sa",
            "Updated": false
         },
         {
            "Property": "System Admin",
            "Updated": false
         }
      ]
   },
    {
      "EditStatus": 0,
      "ID": "PBT=[UserNames] UserNames.DT=[2019-07-25 11:40:06.753] User
Names.ID=[c3878782-0843-4be7-8c0f-dad8dc00a6d7]",
      "Properties": [
        {
          "Property": "32",
          "Updated": false
        },
        {
          "Property": "jdelacruz",
          "Updated": false
        },
        {
          "Property": "Juan Dela Cruz",
          "Updated": false
        }
      ]
    }
  ],
  "PropertyList": [
    "UserId",
    "Username",
    "UserDesc"
  ]
}
```

Example

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
string ido = "UserNames";
string properties = "UserId, Username, UserDesc";
string filter = string.Empty;
string orderBy = "UserId";
string postQueryMethod = string.Empty;
int recordCap = -1;

IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();
string json = soapClient.LoadJson( sessionToken, ido, properties, filter,
 orderBy, postQueryMethod, recordCap );
```

SaveDataSet

The SaveDataSet method modifies a collection (inserting, updating, or deleting records) using the UpdateCollection method of an IDO.

Syntax

```
public DataSet SaveDataSet( string strSessionToken,
                            DataSet updateDataSet,
                            bool refreshAfterSave,
                            string strCustomInsert,
                            string strCustomUpdate,
                            string strCustomDelete )
```

Parameters

Name	Description
strSessionToken	This is the session token obtained through a call to the CreateSessionToken API.
updateDataSet	This parameter specifies the data set that contains records to be updated.
refreshAfterSave	When set to TRUE, this parameter specifies that the data set records to be updated are to be refreshed and returned to the caller.
strCustomInsert	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.
strCustomUpdate	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.
strCustomDelete	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.

Output

Returns a data set containing rows refreshed after the save. If the refreshAfterSave parameter is set to FALSE, this method returns a value of 'null'.

Example 1 – Insert action using SaveDataSet

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
DataSet insertDS = new DataSet( "UserNames" );
DataTable idoTable = insertDS.Tables.Add( "IDO" );
idoTable.Columns.Add( "Username", typeof( string ) );
idoTable.Columns.Add( "UserDesc", typeof( string ) );
idoTable.Rows.Add( new object[] { "wsmiith", "Will Smith" } );

IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();
DataSet updatedDS = soapClient.SaveDataSet( sessionToken, insertDS, true,

string.Empty, string.Empty, string.Empty );
```

Example 2 – Update action using SaveDataSet

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
// GetDataSet makes a call to the LoadDataSet web service method to get
the collection
DataSet ds = GetDataSet();
DataTable idoTable = ds.Tables["IDO"];
// Row to be updated
DataRow row = idoTable.Rows[2];
// Update the record's property value
row["UserDesc"] = "John Doe Sr.";
```

Example 3 – Delete action using SaveDataSet

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
// GetDataSet makes a call to the LoadDataSet web service method to get
the collection
DataSet ds = GetDataSet();
DataTable idoTable = ds.Tables["IDO"];
// Record to be deleted
var row = idoTable.Rows[3];
row.Delete();

DataSet updatedDS = IDOSoapClient.SaveDataSet( sessionToken, ds, true,
string.Empty, string.Empty, string.Empty );
```

SaveJson

The SaveJson method modifies a collection (inserting, updating, or deleting records) using the UpdateCollection method of an IDO, using JSON.

Syntax

```
public string SaveJson( string strSessionToken,
                        string updateJsonObject,
                        string strCustomInsert,
                        string strCustomUpdate,
                        string strCustomDelete )
```

Parameters

Name	Description
strSessionToken	This is the session token obtained through a call to the CreateSessionToken API.
updateJsonObject	This parameter specifies a JSON object that contains records to be updated.
refreshAfterSave	When set to TRUE, this parameter specifies that the records to be updated are to be refreshed and returned to the caller.
strCustomInsert	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.
strCustomUpdate	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.
strCustomDelete	This parameter is a comma-delimited list of methods and/or instructions which override the default save behavior.

Output

None

Example 1 – Insert action using SaveJson

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";

// property values to insert
PropertyStatusPair username = new PropertyStatusPair() { Property =
"jdoe1",
Updated = true };
PropertyStatusPair userDesc = new PropertyStatusPair() { Property = "John
 Doe", Updated = true };

// row to insert
SimpleIDOItem idoItem = new SimpleIDOItem
{
```

```
   EditStatus = SimpleIDOItem.Modified.Inserted,
   Properties = new List<PropertyStatusPair>() { username, userDesc }
};

SimpleIDOItemList insertRequest = new SimpleIDOItemList
{
   IDOName = "UserNames",
   PropertyList = new[] { "Username", "UserDesc" },
   Items = new List<SimpleIDOItem>()
};
insertRequest.Items.Add( idoItem );

IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();

// provide the insertRequest as the payload
string payload = JsonConvert.SerializeObject( insertRequest );
string result = soapClient.SaveJson( sessionToken, payload, string.Empty,

string.Empty, string.Empty );
```

Example 2 – Update action using SaveJson

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5v1903teP0jSDwkFs";

// property values to update
PropertyStatusPair userDesc = new PropertyStatusPair() { Property = "John
 Doe Sr.", Updated = true };

// row to update
SimpleIDOItem idoItem = new SimpleIDOItem
{
   EditStatus = SimpleIDOItem.Modified.Modified,
   ID = "PBT=[UserNames] UserNames.DT=[2019-07-29 12:54:00.937] User
Names.ID=[6565e11a-80d8-43de-b91e-951df7867816]",
   Properties = new List<PropertyStatusPair>() { userDesc }
};

SimpleIDOItemList updateRequest = new SimpleIDOItemList
{
   IDOName = "UserNames",
   PropertyList = new[] { "UserDesc" },
   Items = new List<SimpleIDOItem> { idoItem }
};

IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();

// provide the updateRequest as the payload
string payload = JsonConvert.SerializeObject( updateRequest );
string result = soapClient.SaveJson( sessionToken, payload, string.Empty,

string.Empty, string.Empty );
```

Example 3 – Delete action using SaveJson

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";

PropertyStatusPair userDesc = new PropertyStatusPair() { Property = "John
 Doe Sr.", Updated = true };

// row to delete
SimpleIDOItem idoItem = new SimpleIDOItem
{
   EditStatus = SimpleIDOItem.Modified.Deleted,
   ID = "PBT=[UserNames] UserNames.DT=[2019-07-29 12:55:22.173] User
Names.ID=[41ef60ef-1ec8-4740-9a52-ea361fbe83de]",
   Properties = new List<PropertyStatusPair>()
};

SimpleIDOItemList deleteRequest = new SimpleIDOItemList
{
   IDOName = "UserNames",
   PropertyList = new[] { "UserDesc" },
   Items = new List<SimpleIDOItem>()
};
deleteRequest.Items.Add( idoItem );

IDOWebService.DOWebServiceSoapClient soapClient = new
IDOWebService.DOWebServiceSoapClient();

// provide the deleteRequest as the payload
string json = JsonConvert.SerializeObject( deleteRequest );
string result = soapClient.SaveJson( sessionToken, json, string.Empty,
string.Empty, string.Empty );
```

Helper classes for the examples

You can use these classes to help construct the request as demonstrated in the code snippets above:

```
public class SimpleIDOItemList
{
   public string IDOName { get; set; }
   public string[] PropertyList { get; set; }
   public List<SimpleIDOItem> Items;
}

public class SimpleIDOItem
{
   public enum Modified
   {
      Unmodified,
      Modified,
      Deleted,
      Inserted
   }

   public string ID { get; set; }
```

```
   public List<PropertyStatusPair>  Properties;
   public Modified EditStatus { get; set; }
}

public struct PropertyStatusPair
{
   public string Property { get; set; }
   public bool Updated { get; set; }
}
```

CallMethod

The CallMethod method executes an IDO method that can either be code in a custom assembly or in a stored procedure.

Syntax

```
public object CallMethod( string strSessionToken,
                          string strIDOName,
                          string strMethodName,
                          ref string strMethodParameters )
```

Parameters

Name	Description
strSessionToken	This is the session token obtained through a call to the CreateSessionToken API.
strIDOName	This is the name of the IDO that publishes the method.
strMethodName	This is the name of the IDO to call.

strMethodParameters	This parameter is an XML-formatted string that contains the parameters to pass to the method. The argument for this parameters looks like this:
	
	`<Parameters>`
	`  <Parameter ByRef="Y|N>value"</Parameter>`
	`  ...etc...`
	`</Parameters>`

Output

Returns the value that was returned to it by the method.

Example

```
string sessionToken = "b/XdI6IQzCviZOGJ0E+002…5vl903teP0jSDwkFs";
string ido = "UserNames";
string idoMethod = "GetUserAttributes";

List<Parameter> parameters = new List<Parameter>
{
   new Parameter() { Text = "sa", ByRef = "Y" },
   new Parameter() { Text = string.Empty, ByRef = "Y" },
   new Parameter() { Text = string.Empty, ByRef = "Y" },
   new Parameter() { Text = string.Empty, ByRef = "Y" },
   new Parameter() { Text = string.Empty, ByRef = "Y" }
};

XmlDocument xdoc = new XmlDocument();
XPathNavigator nav = xdoc.CreateNavigator();

using ( XmlWriter writer = nav.AppendChild() )
{
   XmlSerializer serializer = new XmlSerializer( typeof( List<Parameter>
 ), new XmlRootAttribute( "Parameters" ) );
   serializer.Serialize( writer, parameters );
}

IDOWebService.DOWebServiceSoapClient soapClient = new IDOWebService.DOWeb
ServiceSoapClient();
// provide the ido parameters as payload
string idoParameters = xdoc.OuterXml;
object result = soapClient.CallMethod( sessionToken, ido, idoMethod, ref
 idoParameters );
```

Helper classes for the example

You can use these classes to help construct the request as demonstrated in the code snippets above:

```
[XmlRoot( ElementName = "Parameter" )]
public class Parameter
{
   [XmlAttribute( AttributeName = "ByRef" )]
   public string ByRef { get; set; }
   [XmlText]
   public string Text { get; set; }
}

[XmlRoot( ElementName = "Parameters" )]
public class Parameters
{
   [XmlElement( ElementName = "Parameter" )]
   public Parameter Parameter { get; set; }
}
```

Chapter 5: The Mongoose .NET client class library

The .NET client class library is ideal to use for external application communications to access and perform IDO-related operations such as loading collections, updating collections, or running methods. It is another layer on top of the IDO Request Interface, using the IDO Request XML schema.

Using .NET commands

To open sessions or access configuration information from an external application, use this basic procedure:

1. Create a .NET application based on .NET 4.7.2, using the language of your choice.
   If you are using Visual Studio, you must use version 2019 or later.
2. Add these references in your project definition to these Mongoose framework's class libraries:
   - IDOProtocol.dll
   - IDORequestClient.dll
   - MGShared.dll
   - WSEnums.dll
   - WSFormServerProtocol.dll

.NET commands

These are the .NET commands supported by Mongoose:

OpenSession

The OpenSession method validates the user's identity and creates a new session in the application. This is essentially the same as the user logging into the application where the caller can communicate with the IDO.

Example

This example code opens a client session that accesses the IDO runtime using credentials such as the user ID, user password, and the configuration name. It also uses the URL in requestServiceURL, where the URL follows this pattern:

`http://servername/IDORequestService/RequestService.aspx`

where *servername* is the DNS or IP address of the server on which the Mongoose application/configuration resides.

```
bool logonSucceeded = false;
Client client = new Client( requestServiceURL, IDOProtocol.Http );
OpenSessionResponseData response = default( OpenSessionResponseData );

using ( client )
{
   // Substitute a valid username, password, and configuration name:
   response = client.OpenSession( username, password, config );
   logonSucceeded = response.LogonSucceeded;
}
```

Note: For the second argument for Client, you can also use IDOProtocol.Ipc. This indicates that the client class should connect to the IDO Runtime Service on the local machine. In this case, requestServiceURL is ignored.

This is equivalent to using the default constructor:

```
Client client = new Client();
```

CloseSession

The CloseSession method is used to close an existing Mongoose session. This is essentially the same as logging out from the application.

Example

This example code terminates an existing session.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );

using ( client )
{
```

```
   client.CloseSession();
}
```

GetConfigurations

The GetConfigurations method returns a ConfigurationInfo class which includes a subset of configuration properties and all the referenced application properties.

The Client class accesses configuration information through the ConfigServer.aspx page in conjunction with the IDO Runtime Service and IDORuntimeHost.exe developer tool. The Web Server application, IDO Runtime Service, and IDORuntimeHost.exe are all installed as part of a basic server installation.

Example 1 – Get a list of configurations

This example code retrieves an array of configuration information classes from the Default configuration group through the ConfigServer.aspx page in IDORequestService, where the URL is like this:

`http://servername/IDORequestService/ConfigServer.aspx`

where *servername* is the DNS or IP address of the server where the Mongoose configurations are installed and accessed.

```
ConfigurationInfoList configList = Client.GetConfigurations();

// Config info can be enumerated as follows
foreach ( ConfigurationInfo config in configList )
{
   Console.WriteLine( config.Name );
}
```

Note: The IDORuntimeHost.exe must be running for this method to work.

Example 2 – Get the configurations for a specific configuration group

This example code retrieves an array of configuration information from a specified configuration server and configuration group name, using the IDORuntimeHost.exe.

```
ConfigurationInfoList configList = Client.GetConfigurations( con
figServerURL, configGroup );

// Config info can be enumerated as follows
foreach ( ConfigurationInfo config in configList )
{
   Console.WriteLine( config.Name );
}
```

Note: The IDORuntimeHost.exe must be running for this method to work.

GetPropertyInfo

The GetPropertyInfo method retrieves a list of properties and their attributes from a collection, or the columns in a database table.

For example, if you ask for the properties of UserNames IDO, the response document returns information like the name of the base table used by the collection and for each property, its data type, label string, class, and so on.

Example

This example code retrieves an array of properties and their attributes from a specified IDO.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
GetPropertyInfoResponseData response = default( GetPropertyInfoResponseData
 );

using ( client )
{
   response = client.GetPropertyInfo( idoName );

   // Property info can be enumerated as follows
   foreach ( PropertyInfo prop in response.Properties )
   {
      // Do something...
      Console.WriteLine( prop.Name );
   }
}
```

LoadCollection

The LoadCollection method uses the LoadCollection method of an IDO to query either an IDO collection or a database table and return the results to the user.

Example 1 – Basic load collection

This example code retrieves the user ID, name, and description from the Users table.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
LoadCollectionResponseData response = default( LoadCollectionResponseData
 );

using ( client )
{
   LoadCollectionRequestData request = new LoadCollectionRequestData()
   {
      IDOName = "UserNames",
      PropertyList = new PropertyList( "UserId, Username, UserDesc" ),
      RecordCap = -1
   };
```

```
   response = client.LoadCollection( request );

   // Property info can be enumerated as follows
   foreach ( IDOItem item in response.Items )
   {
      // Do something...
      foreach ( IDOPropertyValue property in item.PropertyValues )
      {
         Console.WriteLine( property.Value );
      }
   }
}
```

Example 2 – Custom load collection

This example code retrieves the note content and description from the Object Notes table using the custom load method GetNotesSp.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
LoadCollectionResponseData response = default( LoadCollectionResponseData
 );

using ( client )
{
   InvokeParameterList clmParameters = new InvokeParameterList
   {
      "UserNames",
      "4d6cb1eb-e4fc-4e12-aae8-95ff1086ee8c"
   };

   CustomLoadMethod clm = new CustomLoadMethod
   {
      Name = "GetNotesSp",
      Parameters = clmParameters
   };

   LoadCollectionRequestData request = new LoadCollectionRequestData()
   {
      IDOName = "ObjectNotes",
      PropertyList = new PropertyList( "SpcnNoteContent, SpcnNoteDesc" ),

      RecordCap = -1,
      CustomLoadMethod = clm
   };

   response = client.LoadCollection( request );

   // Property info can be enumerated as follows
   foreach ( IDOItem item in response.Items )
   {
      // Do something...
      foreach ( IDOPropertyValue property in item.PropertyValues )
      {
         Console.WriteLine( property.Value );
```

```
        }
    }
}
```

Example 3 – Nested load collection

This example code is for a nested or hierarchical request. This request queries the Users table and the User Emails tables in a single request, to get the usernames and associated email addresses for each user.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
LoadCollectionResponseData response = default( LoadCollectionResponseData
 );

using ( client )
{
   LoadCollectionRequestData emailsRequest = new LoadCollectionRequestData

   {
      IDOName = "UserEmails",
      PropertyList = new PropertyList( "EmailAddress, EmailType" ),
      RecordCap = -1
   };
   // Set the relationship data of the child and parent IDOs
   emailsRequest.SetLinkBy( "UserId", "UserId" );

   LoadCollectionRequestData usersRequest = new LoadCollectionRequestData

   {
      IDOName = "UserNames",
      PropertyList = new PropertyList( "UserId, Username, UserDesc" ),
      RecordCap = -1
   };
   // Nest the user emails LoadCollection request inside the user LoadCol
lection request
   usersRequest.AddNestedRequest( emailsRequest );

   response = client.LoadCollection( usersRequest );

   // Property info can be enumerated as follows
   foreach ( IDOItem item in response.Items )
   {
      // Do something...
      foreach ( IDOPropertyValue property in item.PropertyValues )
      {
         Console.WriteLine( property.Value );
      }
   }
}
```

UpdateCollection

The UpdateCollection method modifies a collection (inserting, updating, or deleting records) using the UpdateCollection method of an IDO.

Example 1 – Basic Insert, Update, and Delete operations using UpdateCollection

This example code shows how you can perform different actions for each row in the collection within a single request. This request creates a new user, updates the description of user **wsmith**, and deletes the record of user **cdelune**.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
UpdateCollectionResponseData response = default( UpdateCollectionResponse
Data );

using ( client )
{
   // item to insert
   IDOUpdateItem inserItem = new IDOUpdateItem( UpdateAction.Insert );
   inserItem.Properties.Add( new IDOUpdateProperty( "Username", "jdoe",
true ) );
   inserItem.Properties.Add( new IDOUpdateProperty( "UserDesc", "John
Doe", true ) );

   // item to update
   IDOUpdateItem updateItem = new IDOUpdateItem( UpdateAction.Update,
"PBT=[UserNames] UserNames.DT=[2019-08-15 17:30:09.870] User
Names.ID=[b3aedc23-722f-4058-b12c-f14bdfd955b4]" );
   inserItem.Properties.Add( new IDOUpdateProperty( "UserDesc", "Will
Smith Sr.", true ) );

   // item to delete
   IDOUpdateItem deleteItem = new IDOUpdateItem( UpdateAction.Delete,
"PBT=[UserNames] UserNames.DT=[2019-08-15 17:30:52.253] User
Names.ID=[018ae411-42c3-48cb-a50b-b943dcb70dbe]" );

  IDOUpdateItems updateItems = new IDOUpdateItems() { inserItem, updateIt
em, deleteItem };
   UpdateCollectionRequestData request = new UpdateCollectionRequestData()

   {
      IDOName = "UserNames",
      Items = updateItems,
      RefreshAfterUpdate = true
   };

   response = client.UpdateCollection( request );

   // Updated IDO items can be enumerated as follows
   foreach ( IDOUpdateItem item in response.Items )
   {
      // Do something...
      foreach ( IDOUpdateProperty property in item.Properties )
```

```
      {
         Console.WriteLine( property.Value );
      }
   }
}
```

Example 2 – Nested update collection operations

This is an example code for a nested or hierarchical request. This request inserts a user-defined type and its value in a single request.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
UpdateCollectionResponseData response = default( UpdateCollectionResponse
Data );

using ( client )
{
   // Create an UpdateCollection request for the child IDO
   IDOUpdateItem userDefTypeVal = new IDOUpdateItem( UpdateAction.Insert
 );
   userDefTypeVal.Properties.Add( new IDOUpdateProperty( "TypeName",
"Month", true ) );
   userDefTypeVal.Properties.Add( new IDOUpdateProperty( "Value", "Jan
uary", true ) );
   IDOUpdateItems userDefTypeValItems = new IDOUpdateItems() {
userDefTypeVal };

   UpdateCollectionRequestData userDefTypeValRequest = new UpdateCollec
tionRequestData
   {
      IDOName = "UserDefinedTypeValues",
      RefreshAfterUpdate = true,
      Items = userDefTypeValItems
   };
   // Set the relationship data of the child and parent IDOs
   userDefTypeValRequest.SetLinkBy( "Name", "TypeName" );

   // Create an UpdateCollection request for the parent IDO
   IDOUpdateItem userDefType = new IDOUpdateItem( UpdateAction.Insert );
   userDefType.Properties.Add( new IDOUpdateProperty( "Description", "An
 amount of time used with calendars", true ) );
   userDefType.Properties.Add( new IDOUpdateProperty( "Name", "Month",
true ) );
   // Nest the user emails LoadCollection request inside the user LoadCol
lection request
   userDefType.AddNestedUpdate( userDefTypeValRequest );
   IDOUpdateItems userDefTypeItems = new IDOUpdateItems() { userDefType
};

   UpdateCollectionRequestData userDefTypeRequest = new UpdateCollection
RequestData
   {
      IDOName = "UserDefinedTypes",
      RefreshAfterUpdate = true,
      Items = userDefTypeItems
```

```
   };

   response = client.UpdateCollection( userDefTypeRequest );

   // Updated IDO items can be enumerated as follows
   foreach ( IDOUpdateItem item in response.Items )
   {
      // Do something...
      foreach ( IDOUpdateProperty property in item.Properties )
      {
         Console.WriteLine( property.Value );
      }
   }
}
```

Invoke

The Invoke method executes an IDO method. This method can be code in a custom assembly, or it can be a stored procedure.

Example

This example code determines the user attributes of user **jdoe** using the IDO method GetUserAttributes.

```
Client client = new Client( requestServiceURL, IDOProtocol.Http );
InvokeResponseData response = default( InvokeResponseData );

using ( client )
{
   InvokeParameterList parameters = new InvokeParameterList
   {
      "jdoe",
      { string.Empty, true },
      { string.Empty, true },
      { string.Empty, true },
      { string.Empty, true }
   };

   InvokeRequestData request = new InvokeRequestData
   {
      IDOName = "UserNames",
      MethodName = "GetUserAttributes",
      Parameters = parameters
   };

   response = client.Invoke( request );

   // Method parameters can be enumerated as follows
   foreach ( InvokeParameter parameter in response.Parameters )
   {
      // Do something...
```

```
        Console.WriteLine( parameter.Value );
    }
}
```

Appendix A: Example: Bookmark IDs in LoadCollection responses

This topic presents an example of a bookmark ID used in a LoadCollection response to an API request.

This request would return the response shown below. It includes the data we requested from the UserNames IDO, such as UserID, UserName and UserDesc and an initial bookmark ID.


```
                {
      "Bookmark":
"<B><P><p>UserId</p></P><D><f>false</f></D><F><v>1</v></F><L><v>2</v></L></B>",

      "Items": [
      [
      {
      "Name": "UserId",
      "Value": "1"
      },
      {
      "Name": "Username",
      "Value": "sa"
      },
      {
      "Name": "UserDesc",
      "Value": "WinStudio Admin user"
      },
      {
      "Name": "_ItemId",
      "Value": "PBT=[UserNames] UserNames.DT=[2019-06-13 14:02:44.003]
UserNames.ID=[407c7a60-c627-436f-88b0-2bd544a07208]"
      }
      ],
      [
      {
      "Name": "UserId",
      "Value": "2"
      },
      {
      "Name": "Username",
      "Value": "FTAutomation"
      },
      {
      "Name": "UserDesc",
      "Value": null
      },
```

```
      {
      "Name": "_ItemId",
      "Value": "PBT=[UserNames] UserNames.DT=[2019-06-13 14:02:44.070]
UserNames.ID=[cfb7d823-0f3c-4c27-b61e-618ad8dc2b5f]"
      }
      ]
      ],
      "Message": "Success",
      "MessageCode": 0
      }
```

Since we specified `rowcap=2`, we only get 2 records initially. If we'd like to get the next 2 records, we need to specify the bookmark ID in our next request along with the load type value:

```
http://server/IDORequestService/MGRESTService.svc/json/UserNames/adv?props=
UserId,Username,UserDesc&rowcap=2&loadtype=Next&bookmark=<B><P><p>UserId</
p></P><D><f>false</f></D><F><v>1</v></F><L><v>2</v></L></B>
```

This request now returns the next 2 rows and a new bookmark ID value that you can use to navigate on your next request:

```
{
    "Bookmark":
"<B><P><p>UserId</p></P><D><f>false</f></D><F><v>3</v></F><L><v>4</v></L></B>",

    "Items": [
        [
            {
                "Name": "UserId",
                "Value": "3"
            },
            {
                "Name": "Username",
                "Value": "mpalanca"
            },
            {
                "Name": "UserDesc",
                "Value": null
            },
            {
                "Name": "_ItemId",
                "Value": "PBT=[UserNames] UserNames.DT=[2019-06-13
14:03:59.083] UserNames.ID=[c869bbbb-e5c0-4812-8e2a-b4447a1eceb9]"
            }
        ],
        [
            {
                "Name": "UserId",
                "Value": "4"
            },
            {
                "Name": "Username",
                "Value": "MarielElleynIna.Palanca@infor.com"
            },
```

```
            {
                "Name": "UserDesc",
                "Value": "MarielElleynIna Palanca"
            },
            {
                "Name": "_ItemId",
                "Value": "PBT=[UserNames] UserNames.DT=[2019-07-02
10:57:01.233] UserNames.ID=[b6bc0447-3621-4e21-9244-be4963eb3c5a]"
            }
        ]
    ],
    "Message": "Success",
    "MessageCode": 0
}
```