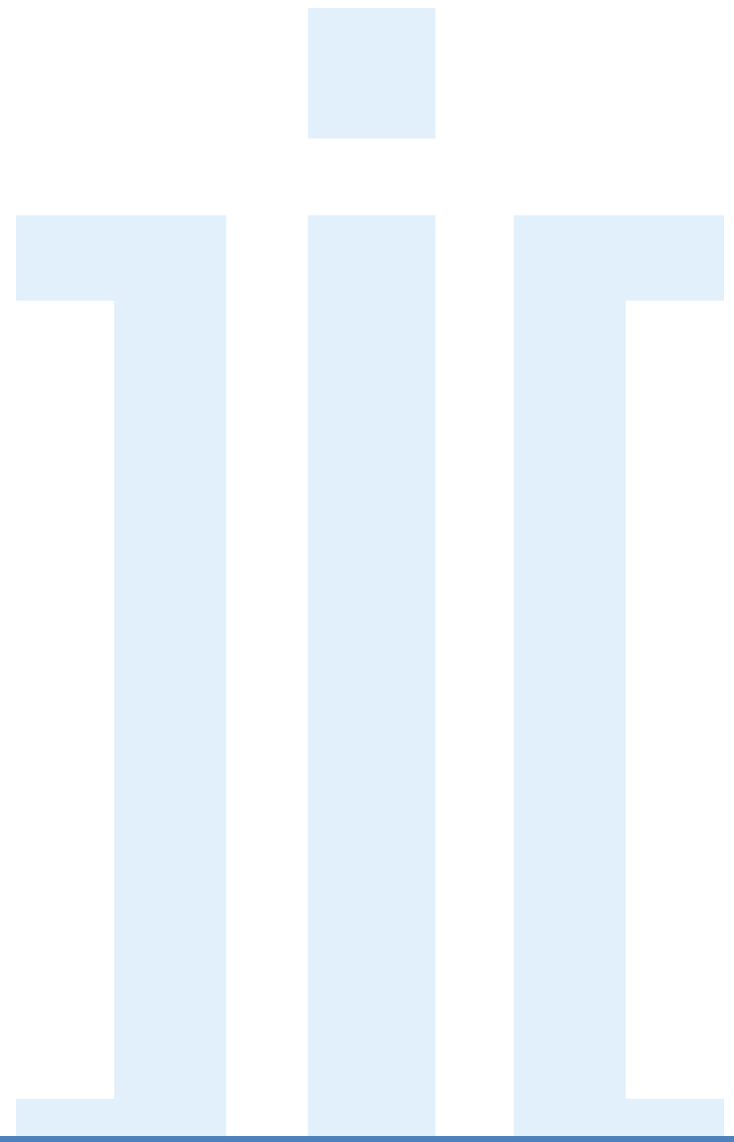


LEVEL WEB SERVICES GUIDE

2019Q1 RELEASE | March 2019





©2019 Ipreo

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

iGet™, iGetArray™, iGetIRR™, iGetAggregate™, and iPut™ are trademarks of iLEVEL Solutions, LLC.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

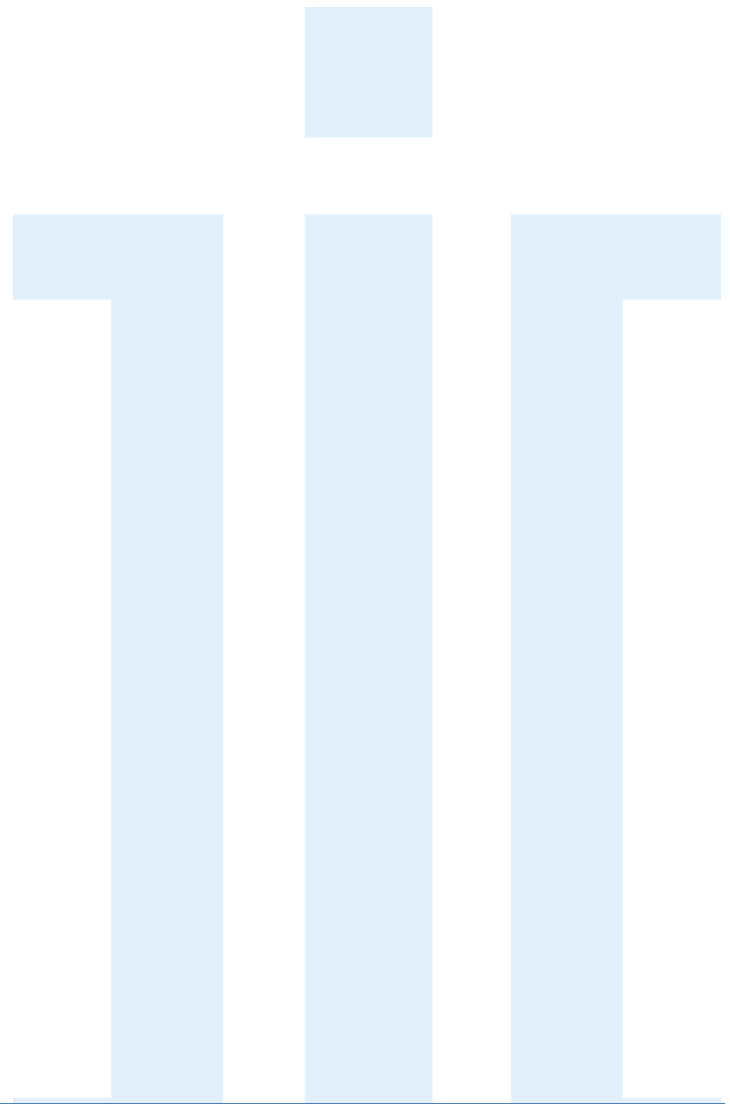


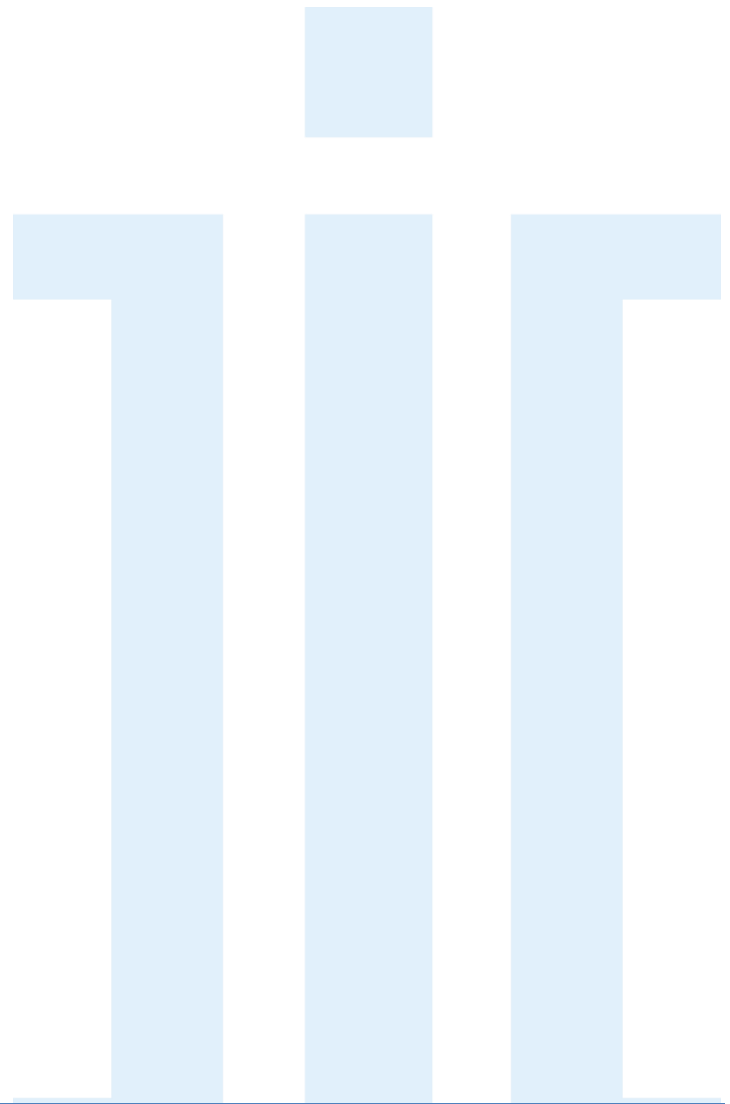
TABLE OF CONTENTS

ABOUT THIS DOCUMENT	4
iLEVEL & WEB SERVICES	5
iLEVEL Overview	6
Architecture	6
Feature Set.....	6
Web Services Overview	8
Getting Started with the API	8
API CALLS AND DESCRIPTIONS.....	9
Web Services API Calls	10
API Call Descriptions.....	14
Entities	14
Entity Relationships	19
Data Items	24
Cash Transactions	32
Currency Rates.....	33
Documents	34
Permissions	37
Web Service Namespace and Backward Compatibility	38
Initiating the iLEVEL Web Service client using Visual Studio.....	39
EXAMPLES & APPLICATIONS	41
Examples.....	42
Microsoft Client.....	42
Non-Microsoft Client (SoapUI).....	50
iLEVEL Data Migration	55
Migrating iLEVEL Data to a Data Warehouse	55
Migrating iLEVEL Data Changes (Deltas) to a Data Warehouse	67
Setting Custom Properties for Assets and Funds	70
Entity Management	70
Web Service Methods for Custom Currency Feeds.....	73
SUPPORT.....	76
Frequently Asked Questions (FAQ)	77
General Questions.....	77
Sandbox and Production.....	78
Data Entry	79
Data Retrieval.....	80
Currency Rates.....	81
Issues/Errors	81
Glossary of Terms.....	82

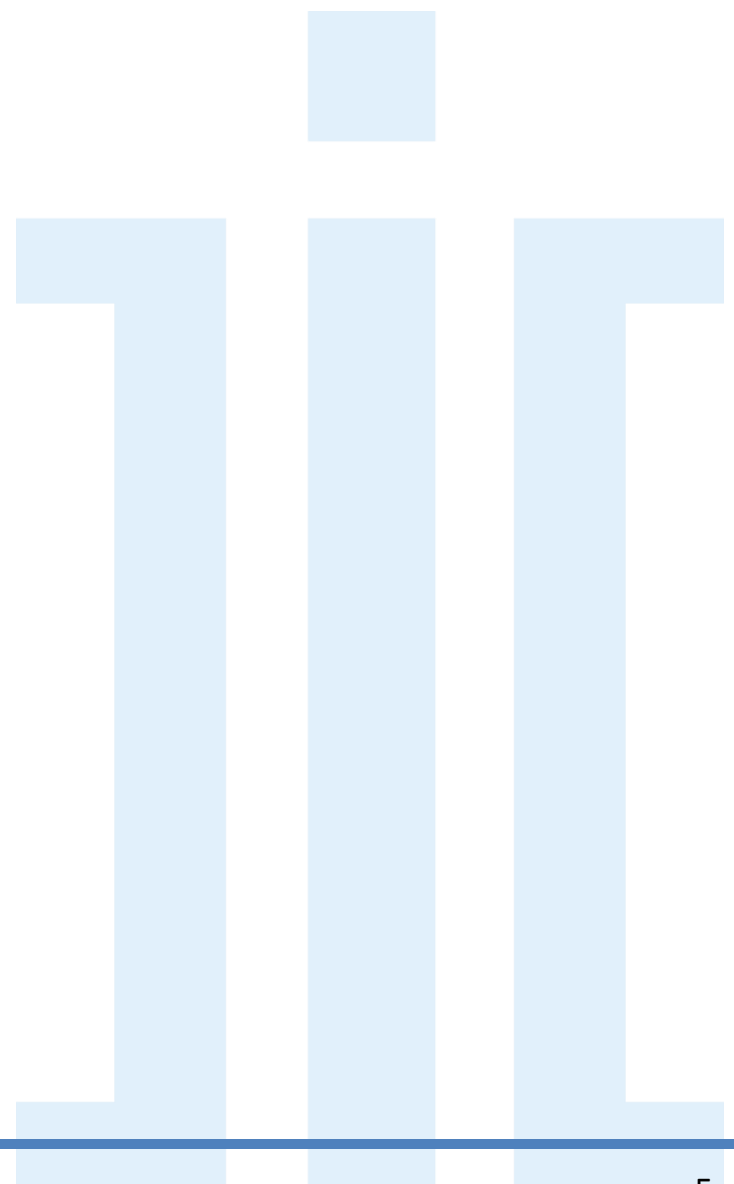
ABOUT THIS DOCUMENT

This document provides an overview of the iLEVEL Platform components, architecture, and feature-set and will describe how to use the iLEVEL Web Services API.

The beginning of the document will cover high-level information about the iLEVEL platform and the Web Services. The rest of this guide will give you, the user, a detailed description of the calls and capabilities you will be able to utilize to manipulate your data, as well as examples and use-case scenarios that cover common situations one may face when using the Web Services.



iLEVEL & WEB SERVICES



iLEVEL Overview

The iLEVEL Private Capital Data Platform is a cloud-based solution built specifically for the Private Capital markets that enables fund managers and investors to control information and gain actionable insights by providing visibility into data collection, investment analysis and performance reporting. The iLEVEL platform streamlines and consolidates data in the cloud, making it easier to access, update, analyze, verify, and share with colleagues and clients.

In addition, iLEVEL provides many ways for users to extend and customize its capabilities to match their firm's precise workflow needs.

Key Features:

- **Automate Data Collection** across your portfolio and consolidate the data into a central repository that serves as a single source of truth for the entire firm.
- **Perform Cross-Portfolio Analytics** and track investment performance with real-time, standardized reporting on the web, in your Excel models or on an iPad.
- **Use Native Excel Add-ins** to flow data securely between desktop spreadsheets and the cloud. Populate spreadsheets—and trace data back to the source document with a click.
- **Generate DDQs and Quarterly Reports** and share actionable insights with investors in any format using flexible, template-based reporting.

Architecture

The iLEVEL Platform architecture is based on the .NET framework. It features a formula-driven approach for Microsoft Excel, a custom Portal framework with a proprietary document library and a dynamic template-based workflow. At the core is a data model and a calculation engine that drive real-time analytics.

Feature Set

There are **four main components** that make up the iLEVEL feature set:

- 1) **iLEVEL Database**, which is the cloud-based data storage system dedicated for each iLEVEL client
- 2) **iLEVEL Excel Add-in**, an Excel plug-in that can be used to create/maintain iLEVEL data collection templates that provides access to iLEVEL Excel formulas
- 3) **iLEVEL Web Portal**, the area to perform administrative tasks such as managing users, entities, data items and workflow
- 4) **iLEVEL Web API**, a set of methods which enable you to enter and/or retrieve data from iLEVEL to your internal data warehouse or other applications

iLEVEL Excel Add-In

The iLEVEL Excel Add-In allows you to create Excel spreadsheets for managing portfolio company data. Installing the Add-In adds an iLEVEL ribbon to the standard Excel toolbar.

There are two important features that come with the Excel Add-in: the iGet and iPut features.

- iPut formulas load data into the database based on specific iPut parameters. The formula consists of the value, asset, scenario, data item, period length and scale.
- iGet formulas retrieve data from the database for reporting on it.

iLEVEL Portal

The iLEVEL Portal lets you automate the collection, analysis, reporting, and exchange of financial and operating performance information for your portfolio companies.

Using the iLEVEL Portal, you can easily create and maintain:

- Individually-permissioned users
- Assets and Portfolio Companies
- Funds
- Asset-specific Charts of Accounts (out-of-the-box accounts are also provided)
- Packages of worksheets taken from Excel workbooks stored in the Document Library

Additionally, you can use the iLEVEL Portal to:

- View performance, valuation, and debt information for the entire portfolio or for specific portfolio companies
- Schedule and send data collection templates to portfolio companies
- Manage the data collection workflow by approving and (if necessary) un-approving portfolio company data and by explicitly “rolling” a period forward or back
- Access industry standard portfolio-level or company-specific reports
- Manage documents in the Document Library, complete with versioning and check-in and check-out functionality
- Produce PDF files by running package

Web Services Overview

The iLEVEL Web Services is an additional tool in the iLEVEL toolkit for users to manage their data. It is a set of methods that facilitate the retrieval of asset and fund data from the iLEVEL database so that it can be exported to another application or database. This tool is largely beneficial to those who have large quantities of data to load in an automated fashion as well as those with the technical resources to write and support the API calls.

This part of the document describes the methods (processes) that are available for the different types of Web Service calls, any unique parameters, and the information they retrieve. Additionally, examples of different use cases for the API calls are provided in the “Examples/Applications” section.

Please refer to the Portal API Support section for more detailed information. This section is available for the API user when logging into iLEVEL.

Getting Started with the API

To get started using the API, the following steps will occur:

1. IHS Markit will send the Data Service sample application. This sample will help your team get started on building out API calls.
2. IHS Markit will create a Sandbox version of your Production environment which can be enabled with a weekly refresh.
3. IHS Markit will enable the “Web Services” setting on your respective Sandbox environment.
4. An API user will need to be created within your Sandbox environment which can be used to test the API.
5. When you are ready to enable the functionality in Production, we will ask you to send a code sample to the PCM Web Services Support team for code review.
6. If the functionality works as expected upon code review, we will enable the “Web Services” setting in your respective Production environment.

[illegible]

Web Services API Calls

Below is the list of the methods, or calls, that are currently available within the Web Services system:

Note: The API user can access an interactive version of this list through the web portal.

Name	Description
AddOrUpdateAssets*	Adds or Updates Assets. Result of the operation will be returned for each asset in the batch.
AddOrUpdateDataItems	Adds or Updates DataItems. Result of the operation will be returned for each data item in the batch.
AddOrUpdateFunds*	Adds or updates Funds. Result of the operation will be returned for each fund in the batch.
AddOrUpdateInvestments	Add or updates Fund\Asset, Fund\Fund, or Asset\Asset associations. Result of the operation will be returned for each investment in the batch.
AddOrUpdateInvestmentTransactions	Adds or Updates InvestmentTransactions. Result of the operation will be returned for each investment transaction in the batch.
AddOrUpdateListTypeDataItemOptions	Adds or Updates options of list type DataItems. Result of the operation will be returned for each option in the batch.
AddOrUpdateSecurities	Adds or Updates Securities. Result of the operation will be returned for each association in the batch.
DeleteDocuments	Deletes Documents from the document library. Result of the operation will be returned for each document in the batch.
DeleteListTypeDataItemOptions	Deletes options of list type DataItems. Result of the operation will be returned for each option in the batch.
DeleteObjects	Deletes objects by their IDs. Result of the operation will be returned for each option in the batch.
DownloadDocument	Downloads a single document from document library by Id.
GetAssets	Gets asset(s) full info by filter or by Ids.
GetCurrencyRates	Gets Currency Rates if custom currency rates are enabled. Returns max 10000 values per 1 call.
GetDataItems	Gets data item(s) full info by filter or by Ids.
GetDefaultValues	Gets a list of default values for corresponding request type.
GetDeletedData	Gets a list of record Ids which were deleted from StandardizedData. Use this method for delta service.
GetDeletedObjects	Gets list of object Ids of a given objectType that were deleted within the specified time range.
GetDependentDataItemIds	Gets a list of data item ids which depend on a given data item.
GetDocuments	Gets document(s) full info by filter or by Ids.
GetFolders	Gets document library folder structure.
GetFunds	Gets fund(s) full info by filter or by Ids.
GetInvestments	Gets relationships (investments) between objects by filters.
GetInvestmentTransactions	Gets list of investment transactions by fund Ids / asset Ids / transactions date (range) / transaction type(s) / scenario(s).

Name	Description
<i>GetListTypeDataItems</i>	Gets list type DataItems with their available list values (options).
<i>GetObjectCustomAttributes</i>	Gets Object custom attributes (custom DataItems defined in category "Company Attributes" for assets and "Fund Attributes" for funds). For now, supported only for assets or funds.
<i>GetObjectRelationships</i>	Gets all relationships (investments) between objects.
<i>GetObjectRelationshipsByIds</i>	Gets relationships (investments) between objects by object relationship ids. Use this method for delta service.
<i>GetObjects</i>	Gets a list of objects within a specific type.
<i>GetObjectsByIds</i>	Gets a list of objects within a specific type by Ids.
<i>GetObjectType</i>	Gets the type of requested object by Id.
<i>GetScenarios</i>	Gets all scenarios.
<i>GetSecurities</i>	Gets security(ies) full info by name or by Ids.
<i>GetUpdatedData</i>	Gets what was added/updated in StandardizedData within the date range and other filter parameters. Filter parameters works as OR for values inside filter and AND between filters. Can return up to 1M Ids. Calls to iGetBatch with these Ids should be batched by no more than 20K. Use this method for delta service.
<i>GetUpdatedObjects</i>	Gets list of object Ids of a given objectType that were added/updated within the specified time range.
<i>iGetBatch</i>	Returns the data for the batch of iGets provided in serviceRequest parameter. Batch has a limit of 20k iGets.
<i>iPutData</i>	Pushes the data for the batch of iPuts provided in serviceRequest parameter. Batch has a limit of 20k iPuts.
<i>SetObjectCustomAttributes</i>	Sets custom attributes for asset or fund (custom DataItems defined in category "Company Attributes" for assets and "Fund Attributes" for funds).
<i>UpdateCurrencyRates</i>	Adds new or updates existing currency rates.
<i>UpdateDocuments</i>	Updates Document(s) metadata. Result of the operation will be returned for each document in the batch.
<i>UploadDocument</i>	Uploads a single document with it's metadata to the document library.

Changes from the previous (2018Q3) release:

- **GeneralPartner** no longer exists as an entity. Now all of the general partners are list type dataitem options, and they can be retrieved by **GetListTypeDataItems()** method. If you want to add new general partner or modify existing one, you can use **AddOrUpdateListTypeDataItemOptions()** method. If you want to delete general partner, you can use **DeleteListTypeDataItemOptions()** method.

- **GetDataItems()** method now returns a list of **DataItemObjectEx** instead of a list of **DataItem** (it contains some additional fields, like *IsCarryOver* and etc.)
- **ClientFund** option in every Enum was changed to **Fund** (no functional changes, only naming fix).
- **GetSecurities2()** method renamed to **GetSecurities()** (no functional changes, only naming fix).
- **GetObjects()** method for:
 - **GetObjectTypes.Asset** returns EntityList of AssetIdentifier (no changes)
 - **GetObjectTypes.Fund** returns EntityList of ClientFund (no changes)
 - **GetObjectTypes.DataItem** returns EntityList of DataItem (new)

AssetIdentifier, ClientFund and DataItem are the simpler versions of Asset, Fund, DataItemObjectEx types and they contain only basic properties. If you want to retrieve full information for any of these entities, you should use dedicated methods – **GetAssets()**, **GetFunds()**, **GetDataItems()**.

- **Status** field was removed from **Asset** type. **AssetStatusId** field still contains actual information about Asset status in iLEVEL.
- **IsDirect** flag in the **EntityPath** type is now **nullable**, and by default it's set to **true**.
- **InvestmentTransaction** fields **BaseAmount** and **BaseCurrency** are now called **LocalAmount** and **LocalCurrency** in order to match Portal and Excel Add-in.
- **DeleteListTypeDataItemOptions()** method now returns a list of **ObjectDeletionResult**.

SIX NEW METHODS WERE INTRODUCED:

- **GetFolders()** allows to retrieve the structure of iLevel document library as a set of key-value pairs, where the key is the folder id and the value is a folder full path.
- **GetDocuments()** allows to retrieve documents' metadata by DocumentSearchCriteria or by document's Ids.
- **DeleteDocuments()** allows to delete documents from iLevel document library by ids.
- **DownloadDocument()** allows to download a single document from the iLevel document library by Id.
- **UploadDocument()** allows to upload a single document to a specified path in the iLevel document library.
- **UpdateDocuments()** allows to update documents' metadata.

NOTES:

- Fiscal year mapping in **AddOrUpdateAsset** and **AddOrUpdateFund** methods should be specified with introduced **FiscalYearEnd** structure. Please, keep in mind that using **PeriodMapping** structure can potentially lead to inconsistent data in iLEVEL!

You can see the examples of the correct mapping configuration below:

```
new Asset
{
    Name = "Test Asset",
    CalendarType = CalendarTypes.Monthly,
    FiscalYearEnd = new FiscalYearEnd
    {
        Month = 12,
        Year = FiscalYearEndYear.Current
    },
    //Other properties...
};
```

```
new Asset
{
    Name = "Test Asset",
    CalendarType = CalendarTypes.Quarterly,
    FiscalYearEnd = new FiscalYearEnd
    {
        Month = 12,
        Year = FiscalYearEndYear.Current,
        MonthToCalendarQuarter = CalendarQuarters.Q1
    },
    //Other properties...
};
```

API Call Descriptions

The Web Services methods can be broken up into **six** categories:

- Entities
- Entity Relationships
- Data Items
- Cash Transactions
- Currency Rates
- Documents

Each category represents a step for data to be established in iLEVEL. First, the user needs to establish entities (funds, assets, securities etc.), followed by establishing relationships between the entities. Then, the user must set up specific data items that will label, then store or calculate data about an asset or a fund and set up cash transactions and currency rates. Once all these items are created, then you are able to begin getting and putting data through the iLEVEL system. In the implementation process, an IHS Markit employee will guide you through the process of establishing an initial set of entities and data items and/or cash transactions.

Also, 2019Q1 endpoint allows to work with documents in the iLEVEL document library.

Entities

This section refers to the calls that address the entities a developer interacts with when using Web Services. These entities include assets, funds, securities. To see how these calls can be used in your system, click on the links below each call.

Note: not all calls have corresponding examples in the guide.

GetAssets

Use this method to retrieve information on assets filtered by asset ids, asset's owner ids or data item ids.

Syntax

C# VB

```

List<Asset> GetAssets(
    List<int> assetIds_,
    List<int> fundIds_,
    List<KeyValuePair<int, int>> dataItems_
)

```

Copy

Parameters

assetIds_
Type: System.Collections.Generic.List<Int32>
filter by AssetIds

fundIds_
Type: System.Collections.Generic.List<Int32>
filter by Funds

dataItems_
Type: System.Collections.Generic.List<KeyValuePair<Int32, Int32>>
List type dataitems and values selected to filter results

Return Value
Type: List<Asset>
List of Asset

Figure 1 – GetAssets Syntax

GetFunds

Use this method to retrieve funds information filtered by fund's owner ids or data item ids.

Syntax

C# VB

```

List<Fund> GetFunds(
    List<int> ownerIds_,
    List<KeyValuePair<int, int>> dataItems_
)

```

Copy

Parameters

ownerIds_
Type: System.Collections.Generic.List<Int32>
List of fund Ids to retrieve

dataItems_
Type: System.Collections.Generic.List<KeyValuePair<Int32, Int32>>
List type dataitems and values selected to filter results

Return Value
Type: List<Fund>
List of Fund

Figure 2 - GetFunds Syntax

GetSecurities

Use this method to retrieve a list of securities by desired filter criteria.

Syntax

C# VB

```

List<Security> GetSecurities(
    List<int> assetIds_,
    List<int> fundIds_,
    List<KeyValuePair<int, int>> dataItems_,
    List<int> secIds_
)

```

Copy

Parameters

assetIds_
Type: System.Collections.Generic.List<Int32>
AssetIds for filter

fundIds_
Type: System.Collections.Generic.List<Int32>
FundIds for filter

dataItems_
Type: System.Collections.Generic.List<KeyValuePair<Int32, Int32>>
List type dataitems and values selected to filter results

secIds_
Type: System.Collections.Generic.List<Int32>
Security Ids to filter results

Return Value
Type: List<Security>
List of Security

Figure 3 – GetSecurities Syntax

GetScenarios

Use this method to retrieve a list of all the scenarios specified by a client. For example, simply call this method to retrieve a list of client-specified scenarios, such as Actual, Budget, Forecast, or any scenarios created by the client using the Scenario Configuration settings on the iLEVEL Portal.

Applications: [All Microsoft client examples](#); [Importing Scenarios](#)

Syntax

C# VB

```

EntityList GetScenarios()

```

Copy

Return Value
Type: EntityList
Container with the list of ScenarioObjects inside.

Figure 4 - GetScenarios Syntax

AddOrUpdateAssets

Use this method to add new or update existing assets. The *overwrite* parameter can be used to instruct this method to overwrite existing assets by using the Last Modified Date for conflict resolution. The *fullBatch* parameter is used to instruct this method to process the whole list of requests in one batch.

Applications: [Create a New Asset](#); [Perform Update of Existing Asset](#)

Syntax

C# VB

Copy

```
UpdateResult AddOrUpdateAssets(
    List<Asset> assets,
    bool overwrite,
    bool fullBatch
)
```

Parameters
assets
 Type: System.Collections.Generic.List<Asset>
 list of assets to update
overwrite
 Type: System.Boolean
 overwrite parameter specifying if LastModifiedDate property of the entity should be used for conflict resolution. true if it should not be used (overwrite any conflicts), when set to false, update will not happen if there are any modifications.
fullBatch
 Type: System.Boolean
 process all requests in one batch
Return Value
 Type: UpdateResult
 result of update operation for each row. results are in the same order as in assets parameter

Figure 5 - AddOrUpdateAssets Syntax

AddOrUpdateFunds

Use this method to add new or update existing funds. An *overwrite* parameter can be used to instruct this method to overwrite existing funds by using the Last Modified Date for conflict resolution. The *fullBatch* parameter is used to instruct this method to process the whole list of requests in one batch.

Syntax

C# VB

Copy

```
UpdateResult AddOrUpdateFunds(
    List<Fund> funds,
    bool overwrite,
    bool fullBatch
)
```

Parameters
funds
 Type: System.Collections.Generic.List<Fund>
 List of funds to update
overwrite
 Type: System.Boolean
 overwrite parameter specifying if LastModifiedDate property of the entity should be used for conflict resolution. true if it should not be used (overwrite any conflicts), when set to false, update will not happen if there are any modifications.
fullBatch
 Type: System.Boolean
 process all requests in one batch
Return Value
 Type: UpdateResult
 result of update operation for each row. results are in the same order as in funds parameter

Figure 6 - AddOrUpdateFunds Syntax

AddOrUpdateSecurities

Use this method to add new or update existing Securities. An *overwrite* parameter can be used to instruct this method to overwrite existing associations by using the Last Modified Date for conflict resolution.

Syntax

C# VB

Copy

```
UpdateResult AddOrUpdateSecurities(
    List<Security> securities,
    bool overwrite
)
```

Parameters

securities
Type: `System.Collections.Generic.List<Security>`
List of Security(s) to update

overwrite
Type: `System.Boolean`
overwrite parameter specifying if LastModifiedDate property of the entity should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return Value
Type: `UpdateResult`
result of update operation for each row. results are in the same order as in securities parameter

Figure 7 - AddOrUpdateSecurities Syntax

Entity Relationships

This section refers to the calls that manipulate the relationships between different entities. To see how these calls can be used in your system, click on the links alongside each call.

GetInvestments

Use this method to retrieve a list of entities associations using filter parameters.

Syntax

C# VB

Copy

```

List<Investment> GetInvestments(
    List<int> owners_,
    List<int> investments_,
    List<KeyValuePair<int, int>> dataItems_,
    List<int> associationIds_
)

```

Parameters

owners_
Type: System.Collections.Generic.List<Int32>
FundIds/AssetIds for filter by Owners

investments_
Type: System.Collections.Generic.List<Int32>
FundIds/AssetIds for filter by Investments

dataItems_
Type: System.Collections.Generic.List<KeyValuePair<Int32, Int32>>
List type dataitems and values selected to filter results

associationIds_
Type: System.Collections.Generic.List<Int32>
Ids of Owner\Investment associations to filter

Return Value
Type: List<Investment>
List of Investment

Figure 8 - GetInvestments Syntax

GetObjectRelationships

This method is similar to GetInvestments but returns all existing investments.

Applications: [Importing Object Associations](#)

Syntax

C# VB

Copy

```

List<ObjectRelationship> GetObjectRelationships()

```

Return Value
Type: List<ObjectRelationship>
List of ObjectRelationship

Figure 9 - GetObjectRelationships Syntax

GetObjectRelationshipsByIds

This method is similar to previous but returns only records for the requested IDs. Use this method for Delta Service, like: `GetUpdatedObjects(UpdatedObjectTypes.FundAsset)`, then call `GetObjectRelationshipsByIds`.

Syntax

C# VB

```

List<ObjectRelationship> GetObjectRelationshipsByIds(
    List<int> ids
)

```

Copy

Parameters
ids
 Type: `System.Collections.Generic.List<Int32>`
 ids for which ObjectRelationships will be returned

Return Value
 Type: `List<ObjectRelationship>`
 List of ObjectRelationship

Figure 10 - GetObjectRelationshipsByIds Syntax

AddOrUpdateInvestments

Use this method to add new or update existing fund to fund, fund to asset, or asset to asset associations. An *overwrite* parameter can be used to instruct this method to overwrite existing associations by using the Last Modified Date for conflict resolution.

Syntax

C# VB

```

UpdateResult AddOrUpdateInvestments(
    List<Investment> investments,
    bool overwrite
)

```

copy

Parameters
investments
 Type: `System.Collections.Generic.List<Investment>`
 List of investments to update

overwrite
 Type: `System.Boolean`
 overwrite parameter specifying if LastModifiedDate property of the entity should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return values
 Type: `UpdateResult`
 result of update operation for each row. results are in the same order as in investments parameter

Figure 11 – AddOrUpdateInvestments Syntax

GetObjects

Use this method to retrieve the list of objects for a specified object type (Asset, Fund, Security, Segment, Segment Info, Data Item, Data Item Category, Scenario, Investment Transaction Type, Client Entity, etc.). For example, if a user wants to retrieve a list of Scenarios, they can specify the object type as “Scenario”.

Applications: [All Microsoft client examples](#); Importing [Assets](#), [Funds](#), [Segments](#) and [Securities](#), Importing Object Associations- [Securities to Assets Associations](#) and [Segments to Assets Associations](#)

Syntax

C# VB

```
EntityList GetObjects(
    GetObjectTypes type
)
```

Copy

Parameters

type
 Type: `DataServiceSample.DataService.GetObjectTypes`
 Type of the object.

Return Value
 Type: `EntityList`
 Container with the list of `DataIdentifierObjects` inside.

Figure 12 - GetObjects Syntax

GetObjectsByIds

Use this method to retrieve the list of objects by object type using a supplied list of object ids. The method supports the same set of object types as `GetObjects` does.

Syntax

C# VB

```
List<NamedEntity> GetObjectsByIds(
    GetObjectTypes objectType,
    List<int> objectIds
)
```

Copy

Parameters

objectType
 Type: `DataServiceSample.DataService.GetObjectTypes`
 object type for objects to retrieve

objectIds
 Type: `System.Collections.Generic.List<Int32>`
 list of Ids of objects to retrieve

Return Value
 Type: `List<NamedEntity>`
 List of Objects (AssetIdentifier, ClientFund, DataItem, Security, etc)

Figure 131 - GetObjectsByIds Syntax

GetUpdatedObjects

Use this method to get a list of object ids for a given object type that was updated or added within a specified date range. The method supports the following object types: Asset, Fund, Security, Segment, Data Item, Scenario, Investment Transaction, Fund-to-Asset Investment, Fund-to-Fund Investment and Asset-to-Asset Investment.

Syntax

C# VB

```

List<int> GetUpdatedObjects(
    UpdatedObjectTypes objectType,
    DateTime startDate,
    DateTime endDate
)

```

Copy

Parameters

objectType
Type: `DataServiceSample.DataService.UpdatedObjectTypes`
Object type of updated objects to retrieve

startDate
Type: `System.DateTime`
start date for a range

endDate
Type: `System.DateTime`
end date for a range

Return Value
Type: `List<Int32>`
List objectids which were added/updated

Figure 14 - GetUpdatedObjects Syntax

GetDeletedObjects

Use this method to retrieve a list of objects (using object type, start date and end date parameters) that have been deleted.

Syntax

C# VB

```

List<int> GetDeletedObjects(
    DeletedObjectTypes objectType,
    DateTime startDate,
    DateTime endDate
)

```

Copy

Parameters

objectType
Type: `DataServiceSample.DataService.DeletedObjectTypes`
Object type of deleted objects to retrieve

startDate
Type: `System.DateTime`
start date for a range

endDate
Type: `System.DateTime`
end date for a range

Return Value
Type: `List<Int32>`
list of recordids which were deleted

Figure 15 - GetDeletedObjects Syntax

GetObjectType

Use this method to retrieve the type of an object using the object id. For example, if a user wants to retrieve the type of an object, the user can specify that object id. The service will return the type for that object (e.g., object is an “Asset”, or object is a “Fund”).

Applications: [All Microsoft client examples](#); Importing [Assets](#), [Funds](#), [Segments](#) and [Securities](#); Importing Object Associations- [Securities to Assets Associations](#) and [Segments to Assets Associations](#)

Syntax

C# VB

```

ObjectTypes GetObjectType(
    int objectId
)

```

Copy

Parameters
objectId
 Type: [System.Int32](#)
 Id of the object.

Return Value
 Type: [ObjectTypes](#)
 Objects enumeration member describing the type of the object.

Figure 16 - GetObjectType Syntax

DeleteObjects

Use this method to delete objects by their type and list of ids. The method supports deletion of Assets, Funds, Relationships and Investment Transactions.

Syntax

C# VB

```

List<ObjectDeletionResult> DeleteObjects(
    RemovableObjectType objectType,
    List<int> objectIDs
)

```

Copy

Parameters
objectType
 Type: [DataServiceSample.DataService.RemovableObjectType](#)
 Type of deleted objects.
objectIDs
 Type: [System.Collections.Generic.List<Int32>](#)
 IDs of the objects to be removed.

Return Value
 Type: [List<ObjectDeletionResult>](#)
 Array of deletion results

Figure 17 - DeleteObjects Syntax

Data Items

This section refers to all calls that affect various data items. To see how these calls can be used in/applied to your system, click on the links alongside each call.

GetDataItems

Use this method to retrieve either the entire list of data items for a client or just the global data items. Additionally, it is possible to filter list to contain data items which fall into specified data item categories.

Applications: [Importing Data Items](#)

Syntax

C# VB

copy

```

List<DataItemObjectEx> GetDataItems(
    DataItemsSearchCriteria dataItemsSearchCriteria
)

```

Parameters

dataItemsSearchCriteria
Type: [DataServiceSample.DataService.DataItemsSearchCriteria](#)

Return values

Type: [List\(DataItemObjectEx\)](#)
List of DataItem objects

Figure 18 - GetDataItems Syntax

GetListTypeDataItems

Use this method to retrieve a list of “list type” data items with associated list values.

Syntax

C# VB

Copy

```

List<ListTypeDataItem> GetListTypeDataItems()

```

Return Value

Type: [List<ListTypeDataItem>](#)
List of ListTypeDataItem objects

Figure 19 - GetListTypeDataItems Syntax

AddOrUpdateDataItems

Use this method to add new or update existing data items. An *overwrite* parameter can be used to instruct this method to overwrite existing data items by using the Last Modified Date for conflict resolution.

Syntax

C# VB

Copy

```
UpdateResult AddOrUpdateDataItems(
    List<DataItemObjectEx> dataItems,
    bool overwrite
)
```

Parameters

dataItems
Type: `System.Collections.Generic.List<DataItemObjectEx>`
List of DataItem(s) to update

overwrite
Type: `System.Boolean`
overwrite parameter specifying if LastModifiedDate property of the data item should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return Value
Type: `UpdateResult`
result of update operation for each row. results are in the same order as in dataItems parameter

Figure 20 - AddOrUpdateDataItems Syntax

AddOrUpdateListTypeDataItemOptions

Adds or Updates list type DataItems options. To add a new option put negative value to the option id field.

Syntax

C# VB

copy

```
UpdateResult AddOrUpdateListTypeDataItemOptions(
    List<ListTypeDataItemOption> dataItemsOptions,
    bool overwrite
)
```

Parameters

dataItemsOptions
Type: `System.Collections.Generic.List<ListTypeDataItemOption>`
List of options to update

overwrite
Type: `System.Boolean`
overwrite parameter specifying if LastModifiedDate property of the data item should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return values
Type: `UpdateResult`
result of update operation

Figure 21 – AddOrUpdateListTypeDataItemOptions Syntax

DeleteListTypeDataItemOptions

Deletes list type DataItems options.

Syntax

C# VB
copy

```

List<ObjectDeletionResult> DeleteListTypeDataItemOptions(
    List<ListTypeDataItemOptionToDelete> dataItemsOptions,
    bool overwrite
)
    
```

Parameters

dataItemsOptions
 Type: [System.Collections.Generic.List<ListTypeDataItemOptionToDelete>](#)
 List of options to remove

overwrite
 Type: [System.Boolean](#)
 Overwrite parameter specifying if LastModifiedDate property of the data item should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return values
 Type: [List<ObjectDeletionResult>](#)

Figure 22 - DeleteListTypeDataItemOptions Syntax

GetDependentDataItemIds

Use this method to retrieve list of data item ids which are dependent on a given data item. It allows to determine which data points are affected by a particular iPut.

Syntax

C# VB
Copy

```

List<int> GetDependentDataItemIds(
    int dataItemId
)
    
```

Parameters

dataItemId
 Type: [System.Int32](#)

Return Value
 Type: [List<Int32>](#)

Figure 23 - GetDependentDataItemIds Syntax

GetObjectCustomAttributes

Use this method to get the custom attributes for assets, funds (those data items defined in company attributes for assets and fund attributes for funds).

Applications: [Setting Custom Properties for Assets and Funds](#)

Syntax

C# VB

```
Dictionary<int, DataValue> GetObjectCustomAttributes(
    ObjectsWithCustomAttributes objectType,
    EntitiesPath path
)
```

Copy

Parameters

objectType
Type: DataServiceSample.DataService.ObjectsWithCustomAttributes
Type of object: Asset or Fund

path
Type: DataServiceSample.DataService.EntitiesPath
EntitiesPath for the iPut

Return Value
Type: Dictionary<Int32, DataValue>
Dictionary of dataitemIds to DataValue

Figure 24 – GetObjectCustomAttributes Syntax

SetObjectCustomAttributes

Use this method to set the custom attributes for a fund or asset (those data items defined in the “Company Attributes” category for assets and the “Fund Attributes” category for funds).

Syntax

C# VB

```
List<DataValue> SetObjectCustomAttributes(
    ObjectsWithCustomAttributes objectType,
    EntitiesPath path,
    Dictionary<int, Object> values
)
```

Copy

Parameters

objectType
Type: DataServiceSample.DataService.ObjectsWithCustomAttributes
Type of object: Asset or Fund

path
Type: DataServiceSample.DataService.EntitiesPath
EntitiesPath for the iPut

values
Type: System.Collections.Generic.Dictionary<Int32, Object>
Dictionary of dataitemIds and corresponding values which need to be set. value will be set to default Scenario/PeriodEnd/etc same as CompanyName for example

Return Value
Type: List<DataValue>
result of update operation DataValue.RequestIdentifier will be set to key from values parameter

Figure 25 - SetObjectCustomAttributes Syntax

iGetBatch

Use this method to retrieve the data from a batch of submitted iGet formulas. iGet formulas conform to the syntax below:

iGet parameters: *=iget(Object identifier, Scenario, Data Item, Period End, Period Length, As Of Date, Owner, Security, Offset, Currency, Fx Type)*

iGets are provided in the serviceRequest parameter and will return any data for which the user has permissions defined in the system.

ParametersList of **DataServiceRequest** should be filled with number of **AssetAndFundGetRequestParameters**.

Applications: [Perform Individual iGet for a Fund Asset Relationship](#); [Get Total Revenue and EBITDA for All Assets](#);

Syntax

C# VB

```

DataValueList iGetBatch(
    DataServiceRequest serviceRequest
)

```

Copy

Parameters

serviceRequest

Type: *DataServiceSample.DataService.DataServiceRequest*

Contains the list of iGets and other request information.

Return Value

Type: *DataValueList*

A container with *DataValue* objects, containing requests with corresponding results.

Figure 26 - iGetBatch Syntax

iPutData

Use this method to put the data from a batch of submitted iPut formulas.

Applications: [Perform Individual iPut for an Asset](#)

iPut formulas conform to the syntax below:

iPut parameters: *=input(Value, Object identifier, Scenario, Data Item, Period End, Period Length, Scale, Owner, Security, Currency, As Of Date)*

Syntax

C# VB

Copy

```

DataValueList iPutData(
    DataServiceRequest serviceRequest
)

```

Parameters

serviceRequest

Type: [DataServiceSample.DataService.DataServiceRequest](#)

Collection of iPutParameters containing details of the request

Return Value

Type: [DataValueList](#)

Collection of iPut return values and server-side profiler measurements

Figure 27 - iPutData Syntax

iPuts are provided in the `serviceRequest` parameter and put any data for which the user has permissions defined in the system.

ParametersList of **DataServiceRequest** should be filled with number of **AssetAndFundPutRequestParameters**.















	Name	Description
	Comment	User defined comment associated with this iPut.
	CurrencyCode	Holds a 3-letter currency code or a reported currency abbreviation, "RC"
	DataItemId	Identifier of the data item used in the request. (Inherited from BaseRequestParameters .)
	DataItemValue	Value to be pushed by iPut.
	DataValueType	Type of the value stored in the request. (Inherited from BaseRequestParameters .)
	DetailId	ID of the security associated with the specified asset.
	EndOfPeriod	Specifies the end of period for which the data was reported.
	EntitiesPath	Path of entities in iGet/iPut i.e. Fund1/Fund2/Asset1, Fund1/Asset1, Fund1/Fund2 or Asset1/Asset2 etc. (Inherited from BaseRequestParameters .)
	Period	Specifies a length of the requested period.
	ReportedDate	Specifies the point in time when the data was reported.
	RequestIdentifier	Unique Identifier of the request which will be returned inside DataValue so match between BaseRequestParameters and response (DataValue) can be performed. Note: Have to be filled in the WebService caller code. (Inherited from BaseRequestParameters .)
	Scale	Provides multiplier of DataItemValue, if the value is numeric.
	ScenarioId	Identifier of the scenario used in the request. (Inherited from BaseRequestParameters .)
	StandardizedDataId	Identifier of the row in StandardizedData table that is associated with the request. (Inherited from BaseRequestParameters .)

Figure 28 - AssetAndFundPutRequestParameters

GetDefaultValues

Use this method to retrieve the list of default values for all request types or specified request types, such as an iGet, iPut, or iGet Array. Only iGet calls are supported by this Web Service. For instance, if a user wants to get the default values for an iGet, the user can specify “Get” as the request type.

Syntax

C# VB

```
BaseDefaultValues GetDefaultValues(
    RequestTypes requestType
)
```

Copy

Parameters
requestType
 Type: `DataServiceSample.DataService.RequestTypes`
 Request type.

Return Value
 Type: `BaseDefaultValues`
 A set of the default values derived from `BaseDefaultValues`.

Figure 29 - GetDefaultValues Syntax

GetDeletedData

Use this method to retrieve a list of record ids (using start date and end date parameters) that have been deleted from StandardizedData table.

Syntax

C# VB

```
List<int> GetDeletedData(
    DateTime startDate,
    DateTime endDate
)
```

Copy

Parameters
startDate
 Type: `System.DateTime`
 start date for a range
endDate
 Type: `System.DateTime`
 end date for a range

Return Value
 Type: `List<Int32>`
 List of StandardizedDataIds

Figure 30 – GetDeletedData

GetUpdatedData

Use this method to retrieve a list of record ids that have been updated in StandardizedData table within the specified parameters. This method can return up to 1 million ids.

Calls to iGetBatch should be split by batches which have no more than 20,000 requests each.

Syntax

C# VB
Copy

```

List<int> GetUpdatedData(
    DateTime startDate,
    DateTime endDate,
    List<int> entityIds,
    List<int> dataItemIds,
    List<int> scenarioIds,
    List<PeriodTypes> periods,
    Nullable<DateTime> StartPeriodEndDate,
    Nullable<DateTime> EndPeriodEndDate
)

```

Parameters

startDate
Type: System.DateTime
start date of changes

endDate
Type: System.DateTime
end date of changes

entityIds
Type: System.Collections.Generic.List<Int32>
entityIds filter, can be null if you don't need to filter by entities

dataItemIds
Type: System.Collections.Generic.List<Int32>
dataItemIds filter, can be null if you need changes for all dataitems

scenarioIds
Type: System.Collections.Generic.List<Int32>
scenarioIds filter, can be null if you need changes for all scenarios

periods
Type: System.Collections.Generic.List<PeriodTypes>
periods filter, can be null if you need changes for all periods

StartPeriodEndDate
Type: System.Nullable<DateTime>
periodEnd filter startdate, can be null if you need to get changes from the beginning

EndPeriodEndDate
Type: System.Nullable<DateTime>
periodEnd filter endDate, can be null if you need to get changes to the end

Return Value
Type: List<Int32>
List of recordIds which where added/updated

Figure 31 – GetUpdatedData

Cash Transactions

This section refers to all calls that allow you to manage cash transaction entries. To see how these calls can be used, click on the links alongside each call.

AddOrUpdateInvestmentTransactions

Use this method to add new or update existing investment transactions. An *overwrite* parameter can be used to instruct this method to overwrite existing transactions by using the Last Modified Date for conflict resolution.

Applications: [Importing investment transactions to a data warehouse](#)

Syntax

C# VB

Copy

```
UpdateResult AddOrUpdateInvestmentTransactions(
    List<InvestmentTransaction> invTrans,
    bool overwrite
)
```

Parameters
invTrans
 Type: System.Collections.Generic.List<InvestmentTransaction>
 List of investment transactions to update

overwrite
 Type: System.Boolean
 overwrite parameter specifying if LastModifiedDate property of the entity should be used for conflict resolution. true if it should not be used (overwrite any conflicts). when set to false, update will not happen if there are any modifications.

Return Value
 Type: UpdateResult
 result of update operation for each row. results are in the same order as in invTrans parameter

Figure 32 – AddOrUpdateInvestmentTransactions Syntax

GetInvestmentTransactions

Use this method to retrieve a list of investment transactions by owner Ids and/or investment Ids and/or transactions date (range) and/or transaction type(s) and/or scenario(s).

Applications: [Get all cash transactions for an Asset](#); [Perform cash transaction entries](#)

Syntax

C# VB

Copy

```
List<InvestmentTransaction> GetInvestmentTransactions(
    InvestmentTransactionsSearchCriteria filter
)
```

Parameters
filter
 Type: DataServiceSample.DataService.InvestmentTransactionsSearchCriteria

Return Value
 Type: List<InvestmentTransaction>
 List of InvestmentTransaction

Figure 33 - GetInvestmentTransactions Syntax

Currency Rates

This section refers to calls that can be used to load currency rates. For a general application scenario, please see [Web Services method for custom currency feed](#)

GetCurrencyRates

Use this method to retrieve a maximum of 10,000 Currency Rate values. This is possible only if the ability to use custom currency rates is enabled for the client. This method has start and end date parameters as well as CurrencyFrom (optional) and CurrencyTo (optional) parameters.

Syntax

```

C# VB
List<CurrencyRateInfo> GetCurrencyRates(
    DateTime startDate_,
    DateTime endDate_,
    string currencyFrom_,
    string currencyTo_
)

Parameters
startDate_
    Type: System.DateTime
    Start Date for a range
endDate_
    Type: System.DateTime
    End Date for a range
currencyFrom_
    Type: System.String
    currency from which conversion happen
currencyTo_
    Type: System.String
    currency to which conversion happen

Return Value
Type: List<CurrencyRateInfo>
List of currency rates
    
```

Figure 34 - GetCurrencyRates Syntax

UpdateCurrencyRates

Use this method to add new or update existing currency rates. For updates, a match is made by comparing the date and employing the currencyTo and currencyFrom values. Up to 10,000 values can be updated/added at one call. This method will not fill the gaps between dates (non-carryover).

Syntax

```

C# VB
void UpdateCurrencyRates(
    List<CurrencyRateInfo> rates_
)

Parameters
rates_
    Type: System.Collections.Generic.List<CurrencyRateInfo>
    currency rates to add/update
    
```

Figure 35 - UpdateCurrencyRates Syntax

Documents

This section refers to calls that can be used to manage documents in iLEVEL document library.

GetFolders

Use this method to retrieve the document library folders structure. The result is a dictionary, where the key is a folder Id and the value is a folder full path. [Usage in Download / Upload documents samples](#)

Syntax

C# VB

copy

```
Dictionary<int, string> GetFolders()
```

Return values

Type: `Dictionary<Int32, String>`
Dictionary of folder id to folder full path

Figure 36 - GetFolders Syntax

GetDocuments

Use this method to retrieve documents' metadata (title, extension, event date, assigned period end etc.) from document library. The documentsSearchCriteria allows to get documents that apply to a set of filters (folder id, extension, related entities etc.). [Usage in all document's samples](#)

Syntax

C# VB

copy

```
List<DocumentMetadata> GetDocuments(
    DocumentsSearchCriteria documentsSearchCriteria
)
```

Parameters

documentsSearchCriteria
Type: `DataServiceSample.DataService.DocumentsSearchCriteria`
A complex filter for documents

Return values

Type: `List<DocumentMetadata>`
List of Documents

Figure 37 - GetDocuments Syntax

UpdateDocuments

Use this method to update documents' metadata in the document library. An *overwrite* parameter can be used to instruct this method to overwrite existing transactions by using the Last Modified Date for conflict resolution.

Applications: [Update documents with a specific tag](#)

Syntax

C# VB

copy

```

UpdateResult UpdateDocuments(
    List<DocumentMetadata> documents,
    bool overwrite
)

```

Parameters

documents
Type: `System.Collections.Generic.List<DocumentMetadata>`
List of documents to update

overwrite
Type: `System.Boolean`
If LastModifiedDate property of the document should be used for conflict resolution. If true, it should not be used (overwrite any conflicts). If false, update will not happen if there are any modifications.

Return values

Type: `UpdateResult`

Figure 38 - UpdateDocuments Syntax

DeleteDocuments

Use this method to delete the documents from the document library by their Ids.

Applications: [Delete documents with assigned period end](#)

Syntax

C# VB

copy

```

List<ObjectDeletionResult> DeleteDocuments(
    List<int> documentIds
)

```

Parameters

documentIds
Type: `System.Collections.Generic.List<Int32>`
IDs of the documents to be removed.

Return values

Type: `List<ObjectDeletionResult>`
Array of deletion results

Figure 39 - DeleteDocuments Syntax

UploadDocument

Use this method to upload a single document with its metadata to the document library. The maximum size of the document is 100 Mb.

Applications: [Upload a document to a specific folder in document library](#)

Syntax

C# VB

copy

```
DocumentUploadResult UploadDocument(
    DocumentMetadata documentMetadata,
    byte[] documentContent
)
```

Parameters

documentMetadata
Type: [DataServiceSample.DataService.DocumentMetadata](#)
Document metadata

documentContent
Type: [System.Byte\[\]](#)
Binary representation of the document content

Return values

Type: [DocumentUploadResult](#)

Figure 40 - UploadDocument Syntax

DownloadDocument

Use this method to download a single document from the document library by its Id.

Applications: [Download all Excel documents from a specific folder and save them on PC](#)

Syntax

C# VB

copy

```
DocumentContent DownloadDocument(
    int documentId_
)
```

Parameters

documentId_
Type: [System.Int32](#)
Document's ID

Return values

Type: [DocumentContent](#)
DocumentContent

Figure 41 - DownloadDocument Syntax

Permissions

Every call to the Web Service is secured. You should set a valid username and password for the client in order to authenticate on a Web Service. In addition, client-established relevant authorization/permission rules set during user creation are in place.

To use the iLEVEL Web Service, a user must have the “Access Data Service” capability. This capability alone will grant the user permissions to the following methods:

Capability	API Calls Enabled
Access Data Service	<i>iPutData and any assets/funds/data items in the iGet formula</i>
Access Data	<i>iGetBatch; GetObjectCustomAttributes; iPutData</i>
Enable iPuts	<i>iPutData; SetObjectCustomAttributes; User-created data items in “Company Attributes” and “Fund Attributes” data item categories</i>
“Edit Entities” & “Manage Entities”	<i>GetAssets; GetFunds; GetFundToAssetAssociations; GetSecurities; DeleteObjects; AddOrUpdateAssets; AddOrUpdateFunds; AddOrUpdateInvestments; AddOrUpdateSecurities</i>
Edit Cash Flows	<i>GetInvestmentTransactions; AddOrUpdateInvestmentTransactions</i>
Manage Data Items	<i>AddOrUpdateDataItems</i>
View Documents	<i>GetFolders; GetDocuments; DownloadDocument</i>
Edit Documents	<i>UpdateDocuments; UploadDocument</i>
Manage Document Library	<i>DeleteDocuments</i>

Web Service Namespace and Backward Compatibility

The iLEVEL Web Service namespace is updated with every major release to include the year and quarter of the release. This is done as changes are introduced to the Web Service during development and client code needs to be updated. When a new release is deployed, it is a good time to update client code to work with the new release.

Backward compatibility for Data Service is provided for the previous endpoint. This means the previous endpoint (2018Q3) will be usable till the next major release after current endpoint (2019Q1). Once this happens, the backward compatibility for the previous endpoint (2018Q3) will be removed. The Endpoint URL for the Web service will have the year and quarter of the release just like the namespace.

For example, see below of the URL for the Production environment:

<https://services.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc> is the primary Endpoint URL for the current release. This Endpoint URL will become the backward compatible URL in the next release, while the 2018Q3 endpoint could be removed.

<https://services.ilevelsolutions.com/DataService/Service/2018/Q3/DataService.svc> is the backward compatibility URL for the 2018Q3 release. The Endpoint on this URL will have the same namespace as it had when it was the primary endpoint for the release.

Initiating the iLEVEL Web Service client using Visual Studio

- 1) In Visual Studio 2010 or later, create a new project with type Console Application. Name it "DataServiceSample".
- 2) Add Service Reference to <https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc> (or other address provided by iLEVEL Solutions)

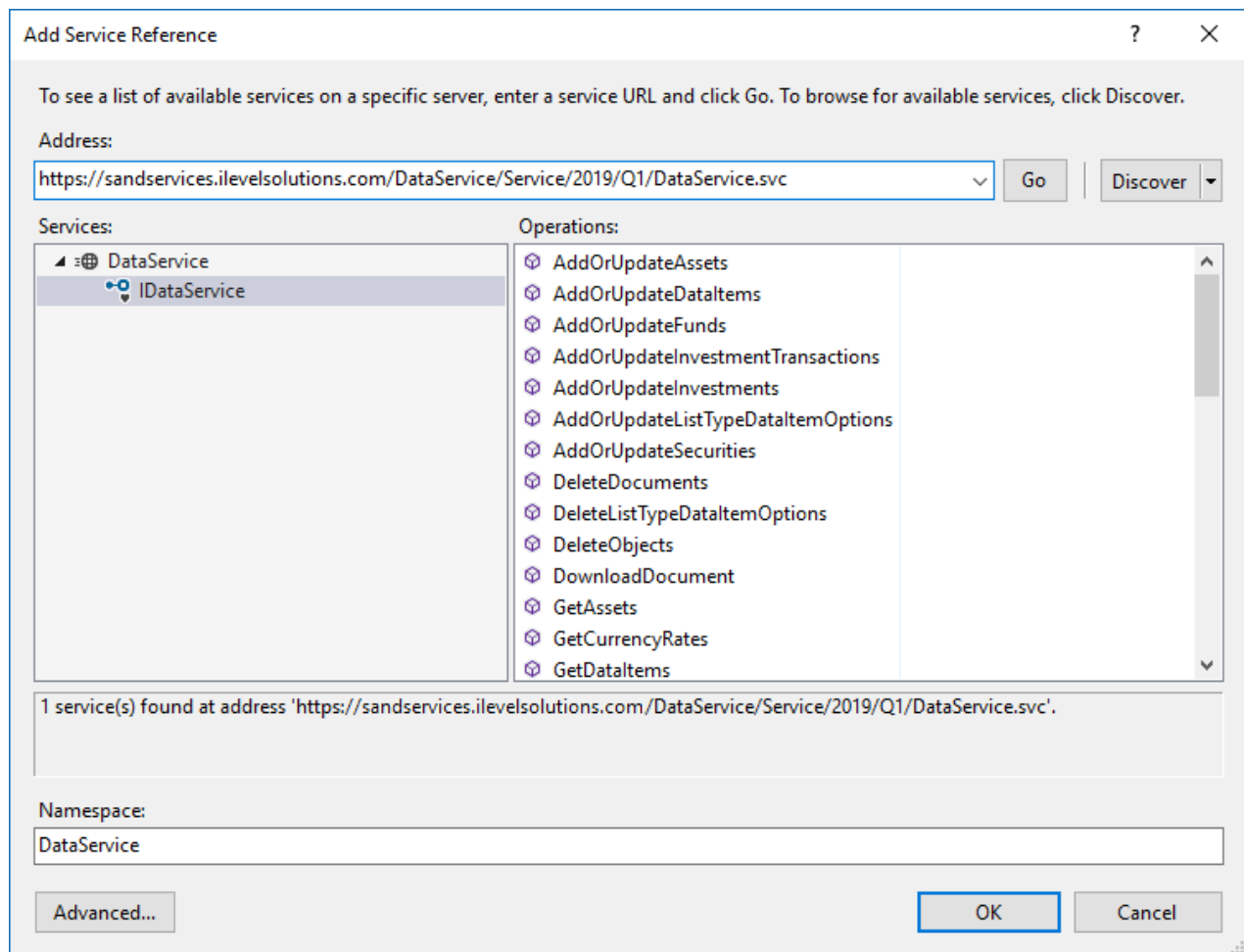
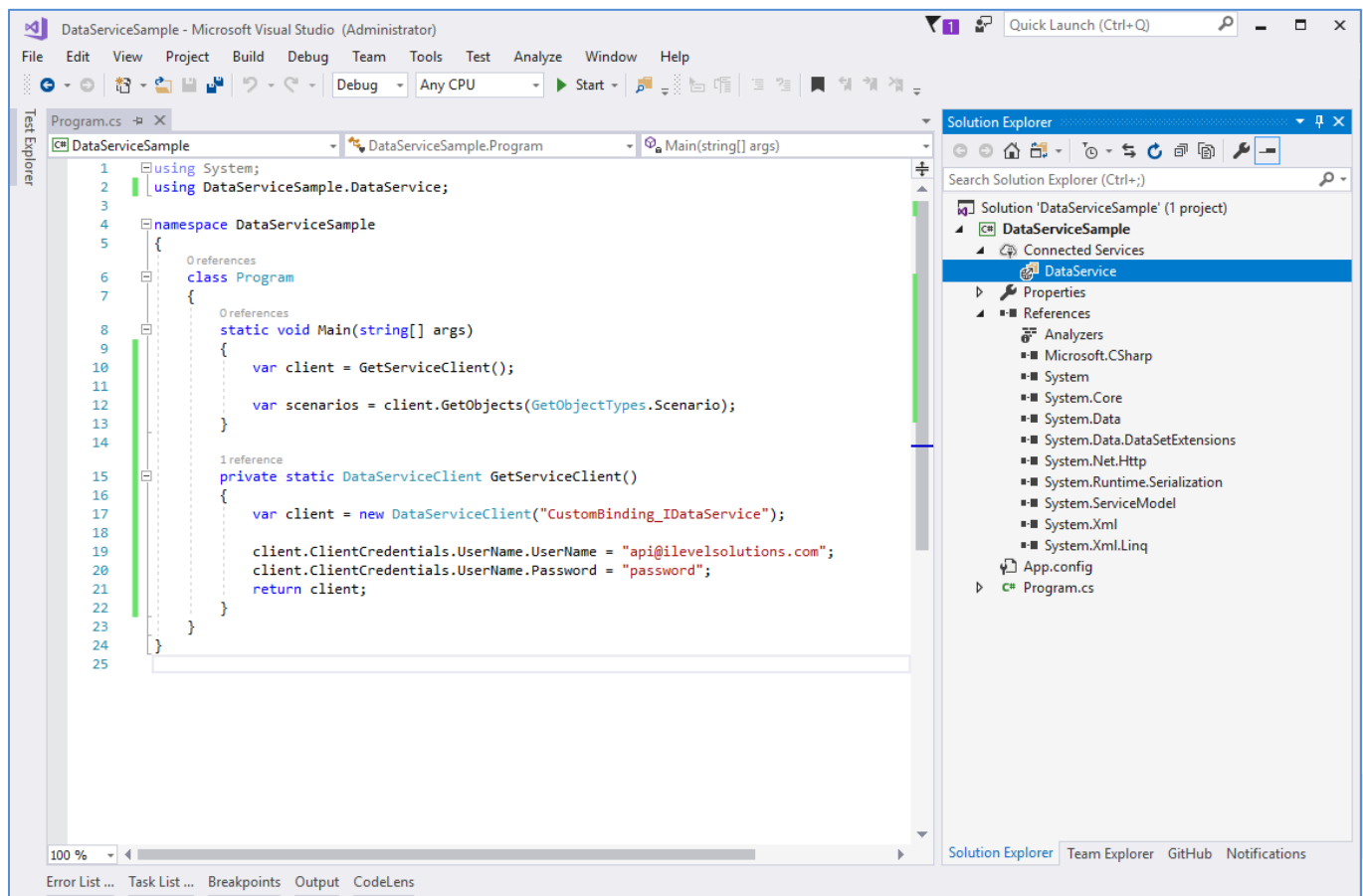


Figure 42 - Add Service Reference

- 3) Click **Advanced** and uncheck **Reuse types in referenced assemblies**.
- 4) Set Collection Type to **System.Collections.Generic.List**. Click **OK**.

Your solution structure will look like this:



5) Once you see the structure above, open App.config for edit.

- Under <system.serviceModel> add the following:

```
<behaviors>
  <endpointBehaviors>
    <behavior name="DataServiceClientBehavior"
      <dataContractSerializer maxItemsInObjectGraph="6553500"/>
    </behavior>
  </endpointBehaviors>
</behaviors>
```

- Find <httpsTransport /> element, add maxReceivedMessageSize and maxBufferSize attributes and set values for them (for example, <httpsTransport maxReceivedMessageSize="134217728" maxBufferSize="134217728" />).
- Find <endpoint /> element and add the following to its attributes: behaviorConfiguration="DataServiceClientBehavior"

This allows you to configure the amount of data you can receive from Web Services.
The default amount in the .NET Framework is 65535 bytes.

EXAMPLES & APPLICATIONS



Examples

Now that the methods and calls above have been outlined and explained, here are some examples as to how you can apply the calls to your own system.

Note: this is not an extensive list but just a few examples.

Microsoft Client

This section provides several common Web Service usage examples. Following each scenario is an example of the relevant code that is responsible for executing the particular Web Service method.

Let's assume that the User Name test3@ilevelsolutions.com and the password *password123* are valid access credentials for the client. This client will be used in the next several examples.

```
var client = new DataServiceClient("CustomBinding_IDataService");
```

```
client.ClientCredentials.UserName.UserName = "test3@ilevelsolutions.com";
client.ClientCredentials.UserName.Password = "password123";
```

Let's also assume that we have following entities for our examples:

- Assets: Company A and Company B
- Funds: Fund 1 and Fund 2

Perform Individual iGet for a Fund/Asset relationship

In this example we are running a request to get the following:

- Scenario: Actual
- Data Item: Total Revenue (value)
- Asset: Company A
- Fund: Fund 1
- Period: Latest Approved

```
//1. Get all assets, this is needed to get id of asset by name, cache it if you need to make multiple iGets
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name);
```

```
//2. Get all funds, this is needed to get id of asset by name, cache it if you need to make multiple iGets
var funds = client.GetObjects(GetObjectTypes.Fund).ToDictionary(v => ((ClientFund)v).ExcelName, v => v);
```

```
//3. Get all data items, this is needed to obtain id by name, cache it if you will need to make multiple iGets
var dataitems = client.GetDataItems(new DataItemsSearchCriteria { GetGlobalDataItemsOnly = false }).ToDictionary(v => v.Name);
```

```
//4. Get all Scenarios, this is needed to obtain id by name, cache it if you will need to make multiple iGets
var scenarios = client.GetScenarios().ToDictionary(v => v.Name);

//5. Prepare iGet
var serviceRequest = new DataServiceRequest();
serviceRequest.DataSourceId = null;
// Parameters for iGets
serviceRequest.ParametersList = new List<BaseRequestParameters>();
int reqId = 1; // this will be identifier by which you can match response with original request
var prm = new AssetAndFundGetRequestParameters(); // this is the actual iGet and its parameters.

prm.DataItemId = dataItems["Total Revenue"].Id; // dataItem Id
prm.DataValueType = (DataValueTypes)dataItems["Total Revenue"].DataValueType; // type of data
prm.ScenarioId = scenarios["Actual"].Id; // scenario
prm.EntitiesPath = new EntitiesPath { Path = new List<int> { funds["Fund 1"].Id, assets["Company A"].Id } }; // fund/asset ids
prm.RequestIdentifier = reqId++; // unique identifier by which you can map response to original request
prm.EndOfPeriod = new Date { Type = DateTypes.Latest }; // period - Latest Approved
prm.ReportedDate = new Date { Type = DateTypes.Latest }; // as of date - Latest Approved
prm.Period = new Period { Type = PeriodTypes.ReportingPeriod }; // period length is Reporting period
prm.Offset = new Period { Type = PeriodTypes.Month, Quantity = 0 }; // no offset
serviceRequest.ParametersList.Add(prm);

//6. Make iGet call to web service
var result = client.iGetBatch(serviceRequest);
```

Get Total Revenue and EBITDA values for all Assets

In this example we are running a request to get the following:

- Scenario: Actual
- Data Items: Total Revenue and EBITDA
- Asset: All (Company A and Company B)
- Period: Latest Approved

```
//1. Get all assets, this is needed to get id of asset by name, cache it if you need to make multiple iGets
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name);

//2. Get all data items, this is needed to obtain id by name, cache it if you will need to make multiple iGets
var dataItems = client.GetDataItems(new DataItemsSearchCriteria { GetGlobalDataItemsOnly = false }).ToDictionary(v => v.Name);

//3. Get all Scenarios, this is needed to obtain id by name, cache it if you will need to make multiple iGets
var scenarios = client.GetScenarios().ToDictionary(v => v.Name);

//4. Prepare iGets
var serviceRequest = new DataServiceRequest();
serviceRequest.DataSourceId = null;
serviceRequest.ParametersList = new List<BaseRequestParameters>(); // parameters for iGets
int reqId = 1; // this will be identifier by which you can match response with original request
```

```
//5. Create iGets for "Total Revenue" and "EBITDA"
foreach (var asset in assets.Values)
{
    var prm = new AssetAndFundGetRequestParameters(); // create iGet Parameters
    prm.DataItemId = dataItems["Total Revenue"].Id; // dataItem Id
    prm.DataValueType = (DataValueTypes)dataItems["Total Revenue"].DataValueType; // type
    prm.ScenarioId = scenarios["Actual"].Id; // scenario
    prm.EntitiesPath = new EntitiesPath { Path = new List<int> { asset.Id } }; // asset
    prm.RequestIdentifier = reqId++; // unique identifier by which you can map response to original request
    prm.EndOfPeriod = new Date { Type = DateTypes.Latest }; // period - Latest Approved
    prm.ReportedDate = new Date { Type = DateTypes.Latest }; // as of date - Latest Approved
    prm.Period = new Period { Type = PeriodTypes.ReportingPeriod }; // period: month, quarter, ytd, etc
    prm.Offset = new Period { Type = PeriodTypes.Month, Quantity = 0 }; // no offset
    serviceRequest.ParametersList.Add(prm);

    prm = new AssetAndFundGetRequestParameters(); // create iGet Parameters
    prm.DataItemId = dataItems["EBITDA"].Id; // dataItem Id
    prm.DataValueType = (DataValueTypes)dataItems["EBITDA"].DataValueType; // type
    prm.ScenarioId = scenarios["Actual"].Id; // scenario
    prm.EntitiesPath = new EntitiesPath { Path = new List<int> { asset.Id } }; // asset
    prm.RequestIdentifier = reqId++; // unique identifier by which you can map response to original request
    prm.EndOfPeriod = new Date { Type = DateTypes.Latest }; // period - Latest Approved
    prm.ReportedDate = new Date { Type = DateTypes.Latest }; // as of date - Latest Approved
    prm.Period = new Period { Type = PeriodTypes.ReportingPeriod }; // period: month, quarter, ytd, etc
    prm.Offset = new Period { Type = PeriodTypes.Month, Quantity = 0 }; // no offset
    serviceRequest.ParametersList.Add(prm);
}

//6. Make iGet call to web service
var results = client.iGetBatch(serviceRequest);
```

Perform Individual iPut for an Asset

In this example we are running a request to get the following:

- Scenario: Actual
- Data Item: Gross Profit
- Asset: Company A
- Period: Current

```
//1. Get all assets, this is needed to get id of asset by name, cache it if you need to make multiple iPuts
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name);

//2. Get all data items, this is needed to obtain id by name, cache it if you will need to make multiple iPuts
var dataItems = client.GetDataItems(false).ToDictionary(v => v.Name);

//3. Get all Scenarios, this is needed to obtain id by name, cache it if you will need to make multiple iPuts
var scenarios = client.GetScenarios().ToDictionary(v => v.Name);
```

```
//4. Prepare iPut request
var serviceRequest = new DataServiceRequest();
serviceRequest.ParametersList = new List<BaseRequestParameters>();
int reqId = 1;

//5. Create iPut for Total Revenue
var prm = new AssetAndFundPutRequestParameters();
prm.DataItemId = dataItems["Gross Profit"].Id;
prm.DataItemValue = 6M;
prm.DataValueType = (DataValueTypes)dataItems["Gross Profit"].DataValueType; // type of data
prm.ScenarioId = scenarios["Actual"].Id; // scenario
prm.EntitiesPath = new EntitiesPath { Path = new List<int> { assets["Company A"].Id } };
prm.RequestIdentifier = reqId++; // unique identifier by which you can map response to original request
prm.EndOfPeriod = new Date { Type = DateTypes.Current };
prm.ReportedDate = new Date { Type = DateTypes.Current }; // as of date - Current period
prm.Period = new Period { Type = PeriodTypes.ReportingPeriod }; // period length is Reporting period
prm.CurrencyCode = "USD";
serviceRequest.ParametersList.Add(prm);

//6. Execute iPut call
var result = client.iPutData(serviceRequest);
```

Get All Cash Transactions for an Asset

In this example we are running a request to get the cash transactions for the following:

- Scenario: Actual
- Asset: Company B

```
// Get all assets, this is needed to get id of asset by name
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name, v => v);

// Get all Scenarios, this is needed to obtain id by name
var scenarios = client.GetScenarios().ToDictionary(v => v.Name);

// Prepare search criteria
var searchCriteria = new InvestmentTransactionsSearchCriteria
{
    InvestmentIds = new List<int>(new int[] { assets["Company B"].Id }),
    ScenarioIds = new List<int>(new int[] { scenarios["Actual"].Id }),
};

// Get the investment transactions according to criteria
var results = client.GetInvestmentTransactions(searchCriteria);
```

Perform Cash Transaction Entries

In this example we are performing a cash transaction entry for the following:

- Scenario: Actual
- Fund: Fund 2
- Asset: Company B

Transaction Type: Additional Investment

```
// Get all assets, this is needed to get id of asset by name
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name, v => v);

// Get all funds, this is needed to get id of fund by name
var funds = client.GetObjects(GetObjectTypes.Fund).ToDictionary(v => ((ClientFund)v).ExcelName, v => v);

// Get all Transaction Types, this is needed to obtain id by name
var transactionTypes = client.GetObjects(GetObjectTypes.InvestmentTransactionType).ToDictionary(v => v.Name);

// Get all Scenarios, this is needed to obtain id by name
var scenarios = client.GetScenarios().ToDictionary(v => v.Name);

// Create investment transaction information
var investmentTransaction = new InvestmentTransaction()
{
    Owner = funds["Fund 2"],
    Investment = assets["Company B"],
    TransactionType = transactionTypes["Additional Investment"],
    Amount = 100000,
    Currency = "USD",
    TransactionDate = new DateTime(2018, 01, 31),
    Scenario = scenarios["Actual"]
};

// Create the list of investment transactions
var investmentTransactionList = new List<InvestmentTransaction>();
investmentTransactionList.Add(investmentTransaction);

// Save the investment transaction
var result = client.AddOrUpdateInvestmentTransactions(investmentTransactionList, true);

// Return investment transaction id if it was saved successfully
if (result.RowResults[0].Error == null)
    return result.RowResults[0].Id;
return 0;
```

Download all Excel documents from a specific folder and save them on PC

In this example we want to download all Excel documents from a specific folder:

- Folder: Reports2018H2
- Extension: xlsx or xls

```
// Get all folders from the document library
var folders = client.GetFolders();

// Get folder id by its name
var folderId = folders.Where(f => f.Value.Contains("Reports2018H2")).Select(f => f.Key).Single();

// Get all documents from that folder using DocumentsSearchCriteria
var criteria = new DocumentsSearchCriteria()
{
    FolderId = folderId,
    FileExtensions = new List<string>() { "xls", "xlsx" }
};

var documents = client.GetDocuments(criteria);

// Specify the folder where you want to save documents
var folderPath = @"C:\Users\Manager\Desktop\Documents\";

// Download the documents and save them one by one
foreach (var document in documents)
{
    var documentName = $"{document.Title}.{document.Extension}";
    var path = Path.Combine(folderPath, documentName);
    var documentContent = client.DownloadDocument(document.Id);

    File.WriteAllBytes(path, documentContent.Content);
}

Console.WriteLine("All documents were downloaded!");
```

Upload a document to a specific folder in document library

In this example we want to upload a new report to a specific folder:

- Folder: Reports2018H2
- Document: NewReport.doc
- Tags: UploadedByAPI
- Related Assets: Company A
- Related Funds: Fund 2

```
// Get all assets, this is needed to get id of asset by name
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(v => v.Name, v => v);

// Get all funds, this is needed to get id of fund by name
var funds = client.GetObjects(GetObjectTypes.Fund).ToDictionary(v => ((ClientFund)v).ExcelName, v => v);

// Get all folders from the document library
var folders = client.GetFolders();

// Get folder id by its name
var folderId = folders.Where(f => f.Value.Contains("Reports2018H2")).Select(f => f.Key).Single();

// Get asset id by its name
var asset = assets["Company A"].Id;

// Get fund id by its name
var fund = funds["Fund 2"].Id;

// Specify the local path where the documents are stored
var uploadPath = @"C:\Users\Manager\Desktop\ForUpload\";

// Getting all information about the folder on the PC into application
DirectoryInfo uploadDirectory = new DirectoryInfo(uploadPath);

// Getting all information about the file to be uploaded
var file = uploadDirectory.GetFiles().Where(x => x.Name == "NewReport.doc").Single();

var fileType = file.Name.Substring(0, file.Name.IndexOf("."));

var content = File.ReadAllBytes(file.FullName);

// Creating the document metadata based on gathered info
var documentMetadata = new DocumentMetadata
{
    Title = fileType,
    Extension = file.Extension,
    FolderID = folderId,
    Tags = new List<string> { "UploadedByAPI" },
    AssetIds = new List<int> { asset },
    FundIds = new List<int> { fund }
};

// Execute UploadDocument call
var result = client.UploadDocument(documentMetadata, content);
```


Update documents with a specific tag uploaded at January 2019

In this example we want to update documents' metadata using search by tag and upload date:

- Tag: JanuaryReport
- Upload Date: January 2019

```
// Get all documents with a specific tag and upload date
var criteria = new DocumentsSearchCriteria()
{
    UserTags = new List<string> { "JanuaryReport" },
    StartUploadDate = new DateTime(2019, 1, 1),
    EndUploadDate = new DateTime(2019, 1, 31)
};

var documents = client.GetDocuments(criteria);

// Updating documents' event date
foreach (var document in documents)
{
    document.EventDate = new DateTime(2019, 2, 1);
}

// Executing UpdateDocuments call
var updateResult = client.UpdateDocuments(documents, false);
```

Delete documents with assigned period end January2019

In this example we want to delete documents with assigned period end January2019:

- AssignedPeriodEnd: January2019

```
// Get all documents with a specific period end
var criteria = new DocumentsSearchCriteria()
{
    AssignedPeriodEnds = new List<string> { "JAN.2019" }
};

var documents = client.GetDocuments(criteria);

// Getting ids of these documents
var documentIds = documents.Select(d => d.Id).ToList();

// Executing delete documents call
var deletionResult = client.DeleteDocuments(documentIds);
```

Non-Microsoft Client (SoapUI)

This section provides Web Service usage examples for non-Microsoft clients.

Let's assume that the User Name test3@ilevelsolutions.com and the password *password123* are valid access credentials for the client.

User should have SoapUI tool installed, it can be downloaded from the following location:

<https://www.soapui.org/downloads/open-source.html>

Steps:

- 1) Run SoapUI
- 2) File -> New SOAP Project, type project name and specify address of WSDL. For **production** WSDL address is <https://services.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc?singleWsdL>, for **sandbox** environment it should be <https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc?singleWsdL>
- 3) Check *Create Requests* checkbox and press OK.
- 4) Remove all CustomBindings but CustomBinding_IDataService2. This endpoint uses SOAP 1.1 protocol without WS-Addressing support.
- 5) Right click on CustomBinding_IDataService2 and choose *Show Interface Viewer*.
- 6) Choose *Service Endpoints* tab and fill in Username (for example, test3@ilevelsolutions.com) and Password (for example, *password123*) text fields. Also, choose *PasswordText* option in *WSS-Type* dropdown menu.
- 7) Expand CustomBinding_IDataService2. It should contain a list all methods that current endpoint supports.

Create a new Asset

In this example we are running a request to create a new Asset which has the following properties:

- Asset Name: Company A
- Asset Status: Current Investment with identifier = 1
- Excel Ticker: Company A
- Reporting Currency: USD
- Reporting Frequency: Monthly

- Fiscal Year End: December 2017
 - Is Fiscal Year End: 2017
 - Initial Collection Period: February 2017
- 1) Expand CustomBinding_IDataService2, then expand AddOrUpdateAssets method and double click on Request 1. It will contain a pre-filled example of SOAP UI request.
 - 2) Remove <soapenv:Body> tag completely and paste the following one:

```
<soapenv:Body>
  <q1:AddOrUpdateAssets>
    <q1:assets>
      <q1:Asset>
        <q1:Name>Company A</q1:Name>
        <q1:ObjectTypeId>Asset</q1:ObjectTypeId>
        <q1:ExcelName> Company A </q1:ExcelName>
        <q1:CalendarType>Monthly</q1:CalendarType>
        <q1:AssetStatusId>1</q1:AssetStatusId>
        <q1:CurrencyCode>USD</q1:CurrencyCode>
        <q1:FiscalYearEnd>
          <q1:Month>12</q1:Month>
          <q1:Year>Current</q1:Year>
        </q1:FiscalYearEnd>
        <q1:InitialPeriod>2017-02-28T00:00:00</q1:InitialPeriod>
      </q1:Asset>
    </q1:assets>
  </q1:AddOrUpdateAssets>
</soapenv:Body>
```

- 8) Make sure that address bar at top of request window shows the following address
<https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc/Soap11NoWSA>
- 9) Click submit request button at top-left corner.
- 10) The reply window should display message with body like presented below. Please note that reply message contains the identifier of newly created asset.

```
<s:Body>
  <AddOrUpdateAssetsResponse xmlns="http://schemas.ilevelsolutions.com/services/dataservice/2019/Q1">
    <AddOrUpdateAssetsResult xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <FinishDate>2019-03-08T14:59:58.077</FinishDate>
      <HasModificationConflicts>>false</HasModificationConflicts>
      <RowResults>
        <RowUpdateResult>
          <Id>132292</Id>
          <RowNumber>0</RowNumber>
        </RowUpdateResult>
      </RowResults>
    </AddOrUpdateAssetsResult>
  </AddOrUpdateAssetsResponse>
</s:Body>
</s:Envelope>
```

Perform update of existing Asset

In this example we are performing update of already existing Asset which has the following properties:

- Asset Identifier: 132292
- Asset Name / Excel Ticker: Company A
- Asset Status: Current Investment with identifier = 1

We are going to change the Asset's Reporting Currency property to EUR. Please note that any update request should have Last Modified Date property set to current date and time.

Steps:

- 1) Expand CustomBinding_IDataService2, then expand AddOrUpdateAssets method and double click on Request 1. It will contain a pre-filled example of SOAP UI request.
- 2) Remove <soapenv:Body> tag completely and paste the following one:

```
<soapenv:Body>
  <q1:AddOrUpdateAssets>
    <q1:assets>
      <q1:Asset>
        <q1:Id>132292</q1:Id>
        <q1:Name>Company A</q1:Name>
        <q1:ExcelName>Company A</q1:ExcelName>
        <q1:AssetStatusId>1</q1:AssetStatusId>
        <q1:CurrencyCode>EUR</q1:CurrencyCode>
        <q1:LastModifiedDate>2019-03-30T15:04:30.1922476Z</q1:LastModifiedDate>
      </q1:Asset>
    </q1:assets>
  </q1:AddOrUpdateAssets>
</soapenv:Body>
```

- 3) Make sure that address bar at top of request window shows the following address
<https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc/Soap11NoWSA>
- 4) Click submit request button at top-left corner.
- 5) The reply window should display message with body like presented below . Please note that in case of success the message should present the identifier of updated Asset and the Has Modification Conflicts property should be set to false.

```
<s:Body>
  <AddOrUpdateAssetsResponse xmlns="http://schemas.ilevelsolutions.com/services/dataservice/2019/Q1">
    <AddOrUpdateAssetsResult xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <FinishDate>2019-03-30T15:05:30.1922476Z </FinishDate>
      <HasModificationConflicts>false</HasModificationConflicts>
      <RowResults>
        <RowUpdateResult>
          <Id>132292</Id>
          <RowNumber>0</RowNumber>
        </RowUpdateResult>
      </RowResults>
    </AddOrUpdateAssetsResult>
  </AddOrUpdateAssetsResponse>
</s:Body>
```

Perform Individual iGet for an Asset

In this example we are running a request to get the following:

- Assets: Company A with identifier = 132292
- Data Item: Gross Profit with identifier = 5002
- Scenario: Actual with identifier = 1
- End of Period: Current
- Exchange Rate: Spot
- Period Length: Reporting Period
- As of Date (Reported Date): Current

Steps:

- 1) Expand CustomBinding_IDataService2, then expand iGetBatch method and double click on Request 1. It will contain a pre-filled example of SOAP UI request.
- 2) Remove <soapenv:Body> tag completely and paste the following one:

```
<soapenv:Body>
<iGetBatch xmlns="http://schemas.ilevelsolutions.com/services/dataservice/2019/Q1">
<serviceRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<DataSourceId i:nil="true"/>
<ParametersList>
<BaseRequestParameters i:type="AssetAndFundGetRequestParameters">
<DataItemId>5002</DataItemId>
<DataValueType>Numeric</DataValueType>
<EntitiesPath>
<Path xmlns:d6p1="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
<d6p1:int>132292</d6p1:int>
</Path>
</EntitiesPath>
<RequestIdentifier>-285</RequestIdentifier>
<ScenarioId>1</ScenarioId>
<StandardizedDataId i:nil="true"/>
<CurrencyCode>RC</CurrencyCode>
<DetailId i:nil="true"/>
<EndOfPeriod>
<IsFiscal>false</IsFiscal>
<PeriodsQuantity>0</PeriodsQuantity>
<Type>Current</Type>
<Value i:nil="true"/>
</EndOfPeriod>
<ExchangeRate>
<Type>Spot</Type>
<Value i:nil="true"/>
</ExchangeRate>
<Offset i:nil="true"/>
<Period>
<Quantity>1</Quantity>
<Type>ReportingPeriod</Type>
</Period>
<ReportedDate>
<IsFiscal>false</IsFiscal>
<PeriodsQuantity>0</PeriodsQuantity>
<Type>Current</Type>
<Value i:nil="true"/>
</ReportedDate>
</BaseRequestParameters>
</ParametersList>
</serviceRequest>
</iGetBatch>
</soapenv:Body>
```

- 3) Make sure that address bar at top of request window shows the following address
<https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc/Soap11NoWSA>
- 4) Click submit request button at top-left corner.

iLEVEL Data Migration

Some clients may choose to migrate their entire iLEVEL database into a separate data warehouse. For those who choose to do so, the following section provides a set of methods which can be used to populate data into an external Data Warehouse. On an ongoing basis, clients will be able to continue to load changes to the data using the iLEVEL Web Service spelled out in the next section: [Migrating iLEVEL Data Changes \(Deltas\) to a Data Warehouse](#).

Migrating iLEVEL Data to a Data Warehouse

While Web Service methods themselves can serve to retrieve data from iLEVEL, certain procedures must be followed to retain the associations between object data so that these relationships will be maintained when the data is imported into an external location.

Objects and Associations Available for Migration

The objects available for migration are:

- Assets
- Funds
- Scenarios
- Segments
- Securities
- Data Items
- Investment Transactions

The object associations are:

- Assets to Funds
- Funds to Funds
- Assets to Assets
- Assets to Data Items
- Securities to Assets
- Segments to Assets
- Scenarios to Data Items

Data Available for Migration

Data items contain values that can be migrated along with the listed objects. Data exists by:

- Asset
- Owner
- Scenario
- Detail (Security, Segment)
- Period (calculation period length)
- Period End (reporting period)
- Reported Date

General Migration Rules

The following rules need to be followed during import:

- 1) When storing entities retrieved from iLEVEL Web Service, Id fields for every object should be stored in an external data warehouse. This way you can find what was added/modified/deleted.
- 2) Object data and object-association data returned all at once needs to be cached during the import procedure.
- 3) Data must be chunked by 20,000 requests per batch.

Importing Objects

Assets

To import assets to your data warehouse, you need to do the following:

- 1) Obtain assets using iLEVEL's Web Service and create a dictionary from the returned results using the following method:
- 2) Load iLEVEL Asset IDs from your data warehouse to memory.
- 3) Find the assets ids which exist in your data warehouse but not in obtained dictionary and remove them.

```
var assets = client.GetObjects(GetObjectTypes.Asset).ToDictionary(a => a.Id, a => (AssetIdentifier)a);
```

```
var deletedAssetIds = existingAssetIds.Where(id => !assets.ContainsKey(id));
DeleteAssets(deletedAssetIds);
```


- 4) Find new assets and add them to your data warehouse.

```
var newAssets = assets.Values.Where(a => !existingAssetIds.Contains(a.Id));
AddAssets(newAssets);
```

- 5) Find and update existing assets.

```
var assetsToUpdate = assets.Values.Where(a => existingAssetIds.Contains(a.Id));
UpdateAssets(assetsToUpdate);
```

- 6) Save the reference to your Assets Dictionary (it will be used later for object associations and data updates).
- 7) Please note that some of the asset properties that can be edited via the iLEVEL Portal are not available via the GetObjects() call. You will need to perform an iGet to retrieve them.

Parameters should be Identifier (varies by asset: assetID), DataItemId (Id of DataItem which is property of asset between 1000 and 1024), OwnerID = 0, DetailID = 0, ScenarioID = 1 (Actual), Period = 1 (RP), PeriodEnd = Current, AsOf = Current

Funds

To import funds to your data warehouse, you need to do the following:

- 1) Obtain funds using iLEVEL's Web Service and create a dictionary from the returned results using the following method:

```
var funds = client.GetObjects(GetObjectTypes.Fund).ToDictionary(f => f.Id, f => (ClientFund)f);
```

- 2) Load iLEVEL Fund IDs from your data warehouse to memory.
- 3) Find the funds ids which exist in your data warehouse but not in obtained dictionary and remove them.

```
var deletedFundIds = existingFundIds.Where(id => !funds.ContainsKey(id));
DeleteFunds(deletedFundIds);
```

- 4) Find new funds and add them to your data warehouse.

```
var newFunds = funds.Values.Where(a => !existingFundIds.Contains(f.Id));
AddFunds(newFunds);
```

- 5) Find and update existing funds.

```
var fundsToUpdate = funds.Values.Where(f => existingFundIds.Contains(f.Id));
UpdateFunds(fundsToUpdate);
```

- 6) Save the reference to the Funds Dictionary (it will be used later for object associations and data updates).
- 7) Please note that some of the fund properties that can be edited via the iLEVEL Portal are not available via the GetObjects() call. You will need to perform an iGet to retrieve them.

Parameters should be: Identifier (varies by fund: fundId), DataItemid (Id of DataItem which is property of fund between 1200 and 1211), OwnerID = 0, DetailID = 0, ScenarioID = 1 (Actual), Period = 1 (RP), PeriodEnd = Current, AsOf = Current

Scenarios

To import scenarios to your data warehouse, you need to do the following:

- 1) Obtain scenarios using iLEVEL's Web Service and create a dictionary from the returned results using the following method:

```
var scenarios = client.GetScenarios().ToDictionary(v => v.Name, v => v);
```

- 2) Load iLEVEL Scenario IDs from your data warehouse to memory.
- 3) Find the Scenario IDs which exist in your data warehouse but not in obtained dictionary and remove them.

```
var deletedScenarioIds = existingScenarioIds.Where(id => !scenarios.ContainsKey(id));
DeleteScenarios(deletedScenarioIds);
```

- 4) Find new scenarios and add them to your data warehouse.

```
var newScenarios = scenarios.Values.Where(v => !existingScenarioIds.Contains(v.Id));
AddScenarios(newScenarios);
```

- 5) Find and update existing scenarios.

```
var scenariosToUpdate = scenarios.Values.Where(v => existingScenarioIds.Contains(v.Id));
UpdateScenarios(scenariosToUpdate);
```

- 6) Save the reference to your Scenarios Dictionary (it will be used later for object associations and data updates).

Segments

To import segments to your data warehouse, you need to do the following:

- 1) Obtain root segments using iLEVEL's Web Service and create a dictionary from the returned results using the following method:

```
var roots = client.GetObjects(GetObjectTypes.SegmentInfo).ToDictionary(f => f.Id, f => (SegmentInfo)f);
```

Roots have information about associations to assets and if rollups are allowed.

- 2) Obtain the whole segment tree using the following method:

```
var segments = client.GetObjects(GetObjectTypes.SegmentNode).ToDictionary(f => f.Id, f => (SegmentNode)f);
```

SegmentNode will have Id, Name and ParentId. This is enough to reconstruct whole tree.

SegmentNode object has properties which should help you build tree if needed.

- 3) Load iLEVEL segments ids from your data warehouse to memory.
- 4) Find the segments ids which exist in your data warehouse but not in obtained dictionary and remove them.

```
var deletedSegmentIds = existingSegmentIds.Where(id => !segments.ContainsKey(id));
DeleteSegments(deletedSegmentIds);
```

- 5) Find new segments and add them to your data warehouse.

```
var newSegments = segments.Values.Where(a => !existingSegmentIds.Contains(f.Id));
AddSegments(newSegments);
```

- 6) Find and update existing segments.

```
var segmentsToUpdate = segments.Values.Where(f => existingSegmentIds.Contains(f.Id));
UpdateSegments(segmentsToUpdate);
```

- 7) Save the reference to the Segments Dictionary (it will be used later for object associations and data updates).
- 8) You can extend the SegmentInfo object to have property AssetIds which will have allowed Asset IDs for this segment and its children.

Securities

To import securities to your data warehouse, you need to do the following:

- 1) Obtain securities using iLEVEL's Web Service and create a dictionary from the returned results using the following method:
- 2) Load iLEVEL Security IDs from your data warehouse to memory.
- 3) Find the Security IDs which exist in your data warehouse but not in obtained dictionary and remove them.

```
var deletedSecurityIds = existingSecurityIds.Where(id => !securities.ContainsKey(id));
DeleteSecurities(deletedSecurityIds);
```

- 4) Find new securities and add them to your data warehouse.

```
var newSecurities = securities.Values.Where(s => !existingSecurityIds.Contains(e.Id));
AddSecurities(newSecurities);
```

- 5) Find and update existing securities.

```
var securitiesToUpdate = securities.Values.Where(a => existingSecurityIds.Contains(e.Id));
UpdateSecurities(securitiesToUpdate);
```

- 6) Save the reference to your Securities Dictionary (it will be used later for object associations and data updates).

- 7) Please note that some of the Security properties that can be edited via the iLEVEL Portal are not available via the GetObjects() call. You will need to perform an iGet in order to retrieve them.

Parameters should be Identifier (varies by security: securityID), DataItemId (Id of DataItem which is property of security between 1300 and 1318), OwnerID = 0, DetailID = 0, SecurityID = 1 (Actual), Period = 1 (RP), PeriodEnd = Current, AsOf = Current

Data Items

To import data items to your data warehouse, you need to do the following:

- 1) Obtain data items using iLEVEL's Web Service and create a dictionary from the returned results using the following method:
- 2) Load iLEVEL data item ids from your data warehouse to memory.
- 3) Find the data items ids which exist in your data warehouse but not in obtained dictionary and remove them.

```
var dataItems = client.GetDataItems(new DataItemsSearchCriteria { GetGlobalDataItemsOnly = false }).ToDictionary(d => d.Name, d
=> (DataItemObjectEx)d);
```

- 4) Find new data items and add them to your data warehouse.

```
var newDataItems = dataItems.Values.Where(s => !existingDataItemIds.Contains(d.Id));
AddDataItems(newDataItems);
```

- 5) Find and update existing data items.

```
var dataItemsToUpdate = dataItems.Values.Where(a => existingDataItemIds.Contains(d.Id));
UpdateDataItems(dataItemsToUpdate);
```

- 6) Save the reference to your DataItems Dictionary (it will be used later for object associations and data updates).
- 7) You can extend the DataItem object to introduce properties for the AssetIds and ScenarioIds that are allowed for this DataItem.

Cash Transactions

To import cash transactions (investment transactions) to your data warehouse, you need to do the following:

- 1) Obtain investment transactions using iLEVEL's Web Service and create a dictionary from the returned results using the following method:

```
var invTransactions = Enumerable.Empty<InvestmentTransaction>();
foreach (var fund in fundDictionary.Values)
{
    var fundInvestmentTransactions = client.GetInvestmentTransactions(
```

```
new InvestmentTransactionsSearchCriteria { OwnerIds = new List<int>{new[] { fund.Id }, }};
invTransactions = invTransactions.Concat(fundInvestmentTransactions);
}
investmentTransactions = invTransactions.ToDictionary(x => x.Id, x => x);
```

- 2) Load iLEVEL investment transactions ids from your data warehouse to memory.
- 3) Find the investment transactions ids which exist in your data warehouse but not in obtained dictionary and remove them.

```
var deletedInvestmentTransactionIds = existingInvestmentTransactionIds.Where(
    id => !investmentTransactions.ContainsKey(id));
DeleteInvestmentTransactions(deletedInvestmentTransactionIds);
```

- 4) Find new investment transactions and add them to your data warehouse.

```
var newInvestmentTransactions = investmentTransactions.Values.Where(
    t => !existingInvestmentTransactionIds.Contains(t.Id));
AddInvestmentTransactions(newInvestmentTransactions);
```

- 5) Find and update existing investment transactions.

```
var investmentTransactionsToUpdate = investmentTransactions.Values.Where(
    t => existingInvestmentTransactionIds.Contains(t.Id));
UpdateInvestmentTransactions(investmentTransactionsToUpdate);
```

- 6) Save the reference to your Investment Transactions Dictionary.

Importing Object Associations

Certain objects in the iLEVEL Platform have associations to other objects. When importing objects and data into a data warehouse, associations must be maintained to assure the meaning and integrity of the data. The sections below describe how to import these associations.

Entities Associations

Relationships mapping is retrieved by calling the `GetObjectRelationships()` method, so the following logic could be used to import entities associations:

- 1) Load existing entities associations.
- 2) Call `GetObjectRelationships` method.

```
var objectrelationships = service.GetObjectRelationships();
var fundtoasset = objectrelationships.Where(r => r.TypeId == ObjectTypes.FundToAsset).ToList();
var fundtofund = objectrelationships.Where(r => r.TypeId == ObjectTypes.FundToFund).ToList();
var assettoasset = objectrelationships.Where(r => r.TypeId == ObjectTypes.AssetToAsset).ToList();
```

- 3) Import entities association using similar logic as in “Import Objects” section.

Assets to Data Items Associations

Dataltem has information about associated assets. This information is stored in the AssetIdsString. If this property is null or an empty string, or if Dataltem.IsGlobal set to true, the data item is associated to all assets. It is a good idea to extend the Dataltem object so AssetIds will be loaded into HashSet<int>. The following logic could be used to import asset to dataitem associations:

- 1) Load existing asset to dataitems associations.
- 2) For each asset in assets, get which data items it is associated to and create AssetToDataltem association.

```
foreach (var asset in assets.Values)
    var assetTodataltems = dataltems.Where(d => d.AssetIds == null || d.AssetIds.Contains(asset.Id))
        .Select(d => new {AssetId = asset.Id, DataltemId = d.Id});
```

- 3) Import asset to data items association using similar logic as in “Import Objects” section.

Securities to Assets Associations

Security to Asset mapping is retrieved together with SecurityObject via GetObjects(GetObjectTypes.Security) call:

- 1) Load existing security to asset associations.
- 2) Import asset to security association using similar logic as in “Import Objects” section.

Scenarios to Dataltems Associations

Dataltem has information about associated scenarios. This information is stored in ScenarioIdsString. It is a good idea to extend the Dataltem object so ScenarioIds will be loaded into HashSet<int>. The following logic could be used to import asset to dataitem associations:

- 1) Load existing scenario to data item associations.
- 2) For each scenario (obtained via **Importing Scenarios, Owners, Securities and Dataltems**), get the data items associated to it and create ScenarioTodataltem association.

```
foreach (var scenario in scenarios)
    var scenarioTodataltems = dataltems.Where(d => d.ScenarioIds.Contains(scenario.Id))
        .Select(d => new {ScenarioId = scenario.Id, DataltemId = d.Id});
```

- 3) Import scenario to data item association using similar logic as in “Import Objects” section.

Segments to Assets Associations

Segments to Asset mapping is retrieved together with SegmentInfo via GetObjects(GetObjectTypes.SegmentInfo) call:

- 1) Load existing segment to asset associations. SegmentInfo has the property AssetIDsString, which contains the list of assets associated to this root segments (and all children). It also has the property IsAllAssets which indicates if a segment associated to all assets.
- 2) Import asset to security association using similar logic as in “Import Objects” section.

Importing Data

Objects in the application (like the ones mentioned in the previous sections) contain associated data. The next step in importing information into your data warehouse is to import the data associated with imported objects.

Please note you need to import data in “chunks” (a constraint on the amount of data requested). In our case, a chunk maxes out at 20,000 iGets in one chunk. The following parameters need to be specified for each iGet:

Asset or Fund, DataItemId, Owner (funds only), Scenario, Detail (Security, Segment), Period (calculation period length), Period End (reporting period) and Reported Date.

Preparation

This section explains the preparation steps needed in order to finish your process of creating iGets. In short, you need to define the specific items from which you wish to build your iGet batches.

- 1) You need to define the assets for which data needs to be imported. Filter the assets dictionary from “**Import Assets**” so it will display only necessary assets (list: **assets**).
- 2) Define which data items you want to import (list: **dataitems**).
- 3) Decide if you going to import Asset level data (in this case you need to pass OwnerID = 0) or Owner level data (Owner ID = FundID or OtherOwnerID). For each asset, filter Owners to return only the owners and Funds for which you want to get data (list: **owners**).
- 4) Filter scenarios to retrieve the list of scenarios for which you want to import data (list: **scenarios**).
- 5) Detail – Filter securities to return the securities for which you want to import data. If DetailsID = 0, then asset-level data will be imported (list: **details**).
- 6) Define the period types for which you need to import data. For example, (ReportingPeriod, Monthly, Quarterly, LTM, YTD, etc.) the iLEVEL system calculates all available periods (depending on the reporting calendar type of the asset), but you may not need all of them to be imported in your system. Create a list of PeriodType values (list: **periods**).

Define the Period End range for which you want to import data. For example, from Jan 2012 to Oct 2012. The date can be specified in multiple ways to retrieve same data. For example, CQ1.2012, MAR.2012 and 03/31/2012 all will return the same data.

- a. **Best practice** is to use calendar dates (i.e. 03/31/2012) whenever possible.
 - For companies with quarterly reporting calendars, requests must be made for the last day in the quarter.
 - For monthly reporting companies, use the last day in month.
 - For 13-period reporting companies for the month where we have two periods requests should be made for the 15th and the last day of that month. (list: **endddates**)
- 7) Define which Reported Dates you want to use. If you need only the latest data, use Latest Approved or Current. (list: **reporteddates**)
- 8) Choose the currency that you want your monetary values to get converted to.
 - a. **Best practice** would be to pick one ("USD" or "EUR"), and if you need a conversion to other currencies, use the rates from GetCurrencyRates method (**currency**).
- 9) Choose FxRate – the date for which your currency conversion will be applied.
 - a. **Best practice** would be to use last period end date you have (**fxtype**).

Building iGets

Once we prepare all the parameters above, we can start building iGets. Remember that we need to process iGets in chunks by 20,000 iGets or less.

The following logic (several nested for each) can be used to import data (assumes that all preparation steps were completed). Iterate through selected combinations (as defined in the "Preparation" section). In this way you can build a batch of iGets (20,000 maximum).

- 1) For each **asset** from your previously defined (list) **assets**: select an asset and go to the next step.
- 2) For each **scenario** from your previously defined (list) **scenarios**: select a scenario and go to the next step.
- 3) For each **dataitem** in **dataitems**:
 - a. Check if **dataitem.AssetIds** has **asset.id**.
 - b. If no, take next **dataitem**; if yes, continue to c.
 - c. Check if **dataitem.Scenarioids** has **scenario.id**.

- d. If no, take next **dataitem**.
 - e. If yes, continue to next step.
- 4) Get **datavaluetype** from your previously defined (list) **dataitem** using **dataitems** dictionary.
- 5) For each **owner** from your previously defined (list) **asset.Owners**: select an Owner and go to the next step.
- 6) For each **detail** from your previously defined (list) **details**: select a detail and go to the next step.
 - a. If **detail** is segment, check if this **segment** is associated with asset using **rootsegment.AssetIds**.
 - b. If no, pick next **detail**.
 - c. If yes, continue to the next step.
- 7) For each **period** from your previously defined (list) **periods**. select a period and go to the next step. Don't forget to specify the multiplier for the period in the Quantity field. This is necessary for those periods where the number is clearly not indicated in the title (like "Month" or "Quarter"). For the periods that don't need or already have the number (like "ReportingPeriod" or "L3M") leave the field empty.
- 8) For each **periodend** from your previously defined (list) **periodends**: select a periodend and go to the next step.
- 9) For each **reporteddate** from your previously defined (list) **reporteddates**: select a reporteddate and go to the next step.
- 10) Construct your **iGet(asset, dataitem, datavaluetype, owner, scenario, detail, period, periodend, reporteddate, currency, fxtype)**.
- 11) Assign a sequential number (increasing every time) to the **RequestIdentifier** field of the iGet.
- 12) Add your newly created **iGet** to the internal list of requests.
- 13) Check if **requests.Count == 20000** make an **iGetBatch** webserver call (can be asynchronous).
- 14) Clear internal list of **requests**.
- 15) Continue with next **iGet** creation until all loops are finished.

Interpreting iGetBatch Results

- 1) You should keep original **iGets** until the **iGetBatch** returns result.
- 2) For each **datavalue** in the **iGetBatch** results:
 - a. Find the matching **iGet** in **iGets** by using the **RequestIdentifier** field which you assigned in the “Building iGets” section - step 11.
 - b. Check if the result is an Error by validating the **datavalue.Error** property.
 - c. Check if the result is null by validating the **datavalue.NoDataAvailable** property.
- 3) Put all iGet parameters and **datavalue.Value** into an internal data structure and add to **responses** list.
- 4) Once you are done with processing your results, call a Stored Procedure to add/update records in your data warehouse.

Migrating iLEVEL Data Changes (Deltas) to a Data Warehouse

After migrating data into an external warehouse, users can continue to load changes to the data using the iLEVEL Web Service methods below. The iLEVEL Web Service provides a set of methods that can be used to populate only data changes (deltas) into an external data warehouse. New/updated and deleted data can be obtained via this set of methods.

Methods to obtain changes:

List<int> GetUpdatedObjects(UpdatedObjectTypes objectType, DateTime startDate, DateTime endDate);

Gets Ids for new and updated Assets, Funds, DataItems, Investments, InvestmentTransactions, Securities, Segments. startDate and endDate must be within 30 days.

List<int> GetDeletedObjects(DeletedObjectTypes objectType, DateTime startDate, DateTime endDate);

Gets Ids of Assets, Funds, DataItems, Investments, InvestmentTransactions, Securities, Segments that were deleted. startDate and endDate must be within 30 days.

List<NamedEntity> GetObjectsByIds(GetObjectTypes objectType, IEnumerable<int> objectIds);

Retrieves Assets, Funds, DataItems, Investments, Securities, Segments by supplied type and list of Ids.

List<InvestmentTransaction> GetInvestmentTransactions(InvestmentTransactionsSearchCriteria filter);

Retrieves InvestmentTransactions by supplied list of Ids.

List<NamedEntity> GetObjectRelationshipsByIds(List<int> ids);

Retrieves direct ObjectRelationships by list of Ids.

**List<int> GetUpdatedData(
 DateTime startDate,
 DateTime endDate,
 IEnumerable<int> entityIds,
 IEnumerable<int> dataItemIds,
 IEnumerable<int> scenarioIds,
 IEnumerable<PeriodTypes> periods,
 DateTime? StartPeriodEndDate,
 DateTime? EndPeriodEndDate);**

Gets the Ids of the records in StandardizedData that were changed. These Ids can be used in calls to iGetBatch to get values. Filter parameters works as OR for values inside filter and AND between filters. For example (where entityIds.Contains(AssetId) AND dataItemIds.Contains(DataItemId)). **startDate** and **endDate** must be within 30 days. This method can return up to 1M Ids. Calls to iGetBatch with these Ids should be batched by no more than 20K.

Now iGetBatch can return all parameters of StandardizedData record.

DataServiceRequest.IncludeStandardizedDataInfo flag can be used when you need to get the parameters returned. This is needed when you get the Id of the record which was new in StandardizedData. Once the record is created in StandardizedData, these parameters do not change (essentially these are the parameters of an iGet); only the value can change.

In the case where **DataServiceRequest.IncludeStandardizedDataInfo** is specified, the iGetBatch returns **List<DataValueEx>**. DataValueEx has SDParameters property that are parameters from StandardizedData.

Note: for performance reasons, use the **DataServiceRequest.IncludeStandardizedDataInfo** flag only for the **new** items. Including this flag for every iGetBatch request will result in a much higher traffic from server to client. For updated ones the flag should not be included because the only thing that could change is the value.

List<int> GetDeletedData(DateTime startDate, DateTime endDate);

Gets the list of Ids from StandardizedData that were deleted. **startDate** and **endDate** must be within 30 days. StandardizedData records with these Ids can be deleted from the Data Warehouse.

Note: There is no point in using the **DataServiceRequest.IncludeStandardizedDataInfo** flag for the Ids that were deleted from StandardizedData as these records no longer exist. The received ids are enough to remove the corresponding entries from Data Warehouse.

Simple algorithm to retrieve changes:

- 1) For initial population, use the method defined in “Migrating iLEVEL Data to a Data Warehouse” section.
- 2) For each objectType, like Assets, Funds, DataItems, Investments, InvestmentTransactions, Securites and Segments call **var ids = GetUpdatedObjects(objectType, dateOfLastSync, DateTime.Now)**. It will return a list of IDs, stored in **ids**.
- 3) To get detailed information about updated objects do the following:
 - a. For all objectTypes except InvestmentTransactions call **GetObjectsByIds(objectType, ids)**. You will get **objects** with fields.
 - b. For InvestmentTransactions object type call **GetInvestmentTransactions(new InvestmentTransactionsSearchCriteria { TransactionIds = ids })** and you will get **InvestmentTransactions** with fields.

- 4) For **ObjectTypes: FundToAsset, FundToFund** and **AssetToAsset** call **GetObjectRelationshipsByIds(ids)**. You will get **ObjectRelationships** with fields.
- 5) Update your database with the results from step 4. If the record does not exist in your database, it means it is new and you need to add it.
- 6) Call **GetDeletedObjects(objectType, dateOfLastSync, DateTime.Now)**. It will return a list of IDs that were deleted.
- 7) Delete (or mark as deleted) records in your database.
- 8) Call **GetUpdatedData(dateOfLastSync, DateTime.Now, entityIds, dataitemIds, scenarioIds, periods, startPeriodEndDate, endPeriodEndDate)** –this returns new and updated record **ids**. For selection of **entityIds, dataitemIds, scenarioIds, periods, startPeriodEndDate, endPeriodEndDate** refer to the “Migrating iLEVEL Data to a Data Warehouse” section.
- 9) Check **ids** against your database and split them for **newIds** and **updatedIds**.
- 10) For **newIds** call **iGetBatch** specifying (set to true) [DataServiceRequest.IncludeStandardizedDataInfo](#) flag.
- 11) Add records returned by **iGetBatch** to your database.
- 12) For **updatedIds** call **iGetBatch** without specifying [DataServiceRequest.IncludeStandardizedDataInfo](#) flag.
- 13) Update values returned by **iGetBatch** in your database.
- 14) Call **GetDeletedData(dateOfLastSync, DateTime.Now)** - it will return a list of IDs that were deleted from **StandardizedData**.
- 15) Delete (or mark as deleted) records in your database.

Setting Custom Properties for Assets and Funds

The iLEVEL Web Service provides two methods that can be used to set data items on the Asset or Fund objects. These data items must be created by the user (i.e., non-default data items) and must be in the “Company Attributes” or “Fund Attributes” category. This is useful when you want to associate (for example) an id from an external system with an Asset or Fund in the iLEVEL Platform. System-defined data items from “Company Attributes” and “Fund Attributes” will not be updated.

Dictionary<int, DataValue> GetObjectCustomAttributes(ObjectWithCustomAttributes objectType, int Id);

Returns custom Data Items for Asset or Fund.

List<DataValue> SetObjectCustomAttributes(ObjectWithCustomAttributes objectType, int Id, Dictionary<int, object> values);

Updates custom Data Items for Asset or Fund.

Example: The user creates a new Data Item named “Asset Identifier” in the data item category “Company Attributes”. The user will be able to retrieve and update this custom attribute.

Example: The user creates a new Data Item named “Fund Identifier” and puts it into the data item category “Fund Attributes”. The user will be able to retrieve and update this custom attribute.

System-defined attributes: The default system-defined attributes for assets like Company Name” or “Excel Ticker” need to be updated using the entity management calls.

Entity Management

The iLEVEL Web Service provides a set of methods which can be used to retrieve full information, as well as create or update some of the entities. Entities available to create and update include Asset, Fund, InvestmentTransactions, Fund-Asset associations, Fund-Fund associations, Asset-Asset associations.

List<Asset> GetAssets(List<int> assetIds, List<int> fundIds, List<KeyValuePair<int, int>> dataItems);

Retrieves list of Asset objects by list of assets and/or funds and/or by list type data item values and/or list of ids.

Assets have the same properties as are available on iLEVEL Portal UI. Usually you need to use this method when you need to make changes to asset.

UpdateResult AddOrUpdateAssets(List<Asset> assets, bool overwrite);

Adds or updates assets. A flag overwrite is used for conflict resolution. If it is set to true, the records will be updated even if the server has a newer version that was sent from client. The LastModifiedDate property of the Asset is used to determine modification conflicts. The result of add/update will be returned for each row.

Asset fields:

Name	Description	Updatable	Required
ID	ID of the record	No	Yes
Name	Name of the asset	Yes	Yes
ExcelName	Excel Ticker (short name)	Yes	Yes
CalendarType	Calendar of the asset	No	Yes
AssetStatusId	Asset Status	Yes	Yes
CurrencyCode	Default currency for the asset	No	Yes
Description	Description	Yes	No
GeographyId	Geography ID (from List type Data Item Geography)	Yes	No
IndustryId	Industry ID (from list type Data Item Industry)	Yes	No
ParentAssetId	Id of the parent asset	Yes	No
ShortDescription	Short asset description	Yes	No
AcquisitionDate	Acquisition date of the asset	Yes	No
ExitDate	Exit Date	Yes	No
AcquisitionAsOf	Acquisition As Of date (reporting period based) to use in iGets/iPuts	Yes	No
ExitAsOf	Exit As Of date (reporting period based) to use in iGets/iPuts	Yes	No
FiscalYearEnd	Sets a fiscal calendar using the same means as used in iLEVEL bulk upload tables in Excel	No	Yes
InitialPeriod	Very first reporting period in the system for this asset	No	Yes
LastModifiedDate	When the record was last modified	No	Yes

Fiscal Year End fields (fund class has the same structure):

Name	Range of values	Description	Required
Month	[1, 12]	Calendar month of Fiscal Year End (FYE).	Yes
MonthToCalendarQuarter	{Q1, Q2, Q3, Q4}	Calendar quarter, which Fiscal Year End (FYE) month maps to.	Required only if CalendarType is set to Quarterly
Year	{Previous, Current}	Calendar year of Fiscal Year End (FYE).	Yes

List<Fund> GetFunds(string nameFilter, IEnumerable<int> ids);

Retrieves list of Fund objects by nameFilter or by list of supplied Ids. Funds have the same properties as are available on the iLEVEL Portal UI. Usually you need to use this method when you need to make changes to fund.

Fund fields:

Name	Description	Updatable	Required
ID	ID of the record	No	Yes
Name	Name of the fund	Yes	No
ExcelName	Excel Ticker (short name)	Yes	Yes
StatusId	Fund Status	Yes	Yes
CalendarType	Type of the fiscal calendar of the fund	No	Yes
CurrencyCode	Default currency for the fund	No	Yes
Description	Description	Yes	No
Vintage	Vintage year of the fund	Yes	Yes
TotalCommittedCapital	Total capital committed to date	Yes	No
InitialCloseDate	Initial Close Data	Yes	No
FinalCloseDate	Final Close Date	Yes	No
FiscalYearEnd	Sets a fiscal calendar using the same means as used in iLEVEL bulk upload tables in Excel (the same structure as used for assets)	No	Yes
InitialPeriod	Very first reporting period in the system for this fund	No	Yes
LastModifiedDate	When the record was last modified	No	Yes

UpdateResult AddOrUpdateFunds(List<Fund> funds, bool overwrite);

Adds or updates funds. A flag overwrite is used for conflict resolution. If it is set to true, the records will be updated even if the server has a newer version that was sent from the client. The LastModifiedDate property of the Fund is used to determine modification conflicts. The result of add/update will be returned for each row.

List<InvestmentTransaction> GetInvestmentTransactions(InvestmentTransactionsSearchCriteria filter);

Retrieves list of InvestmentTransaction objects by set of filters. InvestmentTransactions have the same properties as are available on the iLEVEL Portal UI. Usually you need to use this method when you need to make changes to InvestmentTransaction.

UpdateResult AddOrUpdateInvestmentTransactions(List<InvestmentTransaction> invTrans, bool overwrite);

Adds or updates InvestmentTransactions. A flag overwrite is used for conflict resolution. If it is set to true, the records will be updated even if the server has a newer version that was sent from client. The LastModifiedDate property of the InvestmentTransaction is used to determine modification conflicts. The result of add/update will be returned for each row.

List<ObjectRelationship> GetObjectRelationships();

Retrieves a list of [ObjectRelationship](#) objects. [ObjectRelationship](#) objects have the same properties as are available on the iLEVEL Portal UI. Usually you need to use this method when you need to make changes to [ObjectRelationship](#).

List<ObjectRelationship> GetObjectRelationshipsByIds(List<int> ids);

Retrieves a list of [ObjectRelationship](#) objects by given list of ids. [ObjectRelationship](#) objects have the same properties as are available on the iLEVEL Portal UI. Usually you need to use this method when you need to make changes to [ObjectRelationship](#).

UpdateResult AddOrUpdateInvestments(List<Investment> investments, bool overwrite);

Adds or updates [Investment](#). A flag overwrite is used for conflict resolution. If it is set to true, the records will be updated even if the server has a newer version that was sent from the client. The LastModifiedDate property of the [Investment](#) is used to determine modification conflicts. The result of add/update will be returned for each row.

**List<Security> GetSecurities(List<int> assetIds_, List<int> fundIds_,
List<KeyValuePair<int, int>> dataItems_,
IEnumerable<int> secIds_);**

Retrieves a list of [Security](#) objects by list of assets and/or funds and/or by list type data item values and/or list of ids. [Security](#) objects have the same properties as are available on the iLEVEL Portal UI. Usually you need to use this method when you need to make changes to [Security](#).

UpdateResult AddOrUpdateSecurities(List<Security> securities, bool overwrite);

Adds or updates [Security](#). A flag overwrite is used for conflict resolution. If it is set to true, the records will be updated even if the server has a newer version that was sent from the client. The LastModifiedDate property of the [Security](#) is used to determine modification conflicts. The result of add/update will be returned for each row.

Web Service Methods for Custom Currency Feeds

Custom currency management is now part of the iLEVEL Data Service. This allows you to upload currency conversion rates or use your own currency conversion rates instead of using the rates provided by iLEVEL. If enabled to use custom currencies, you will be able to get/add/update rates via Web Service.

The Custom Currency Web Services are secured in the same way as for the Data Web Service, via username and password. There are two additional Web Service methods that facilitate the storage and retrieval of currency rates.

GetCurrencyRates

Use this method to retrieve currency rates using the following parameters:

GetCurrencyRates(DateTime startDate, DateTime endDate, string currencyFrom, string currencyTo)

All parameters must be specified in order to retrieve data. The currencyFrom and currencyTo fields are currency codes in ISO. If you are not enabled for Custom Currency Rates, the Web Service will throw an exception.

The service will return the following object. The same object will be used to input currency rates.

```
public class CurrencyRate
{
    public DateTime Date;
    public decimal Bid;
    public decimal Ask;
    public decimal Mid;
    public string currencyFrom;
    public string currencyTo
}
```

Currency rate fields:

Field Name	Description
Date	Date for which currency conversion rate was set
Bid	Bid rate for currency conversion
Ask	Ask rate for currency conversion
Mid	This is a calculated field. $(\text{Bid} + \text{Ask})/2$. If Bid or Ask is not provided, then Mid will be equal to the provided field. If supplied for Add/Update currency, it will be ignored.
currencyFrom	3-letter currency ISO code from which the conversion rate was stored (optional)
currencyTo	3-letter currency ISO code to which the conversion rate was stored (optional)

Example: Get All Rates for the Past Month from EUR to USD

```
var client = new DataServiceClient();

//set credentials
client.ClientCredentials.UserName.UserName = "username";
client.ClientCredentials.UserName.Password = "password";

var existingRates = client.GetCurrencyRates(DateTime.Now.AddMonths(-1), DateTime.Now, "EUR", "USD")
```

Example: Get All Rates for the Past Month for all Currencies

```
var client = new DataServiceClient();

//set credentials
client.ClientCredentials.UserName.UserName = "username";
client.ClientCredentials.UserName.Password = "password";

var existingRates = client.GetCurrencyRates(DateTime.Now.AddMonths(-1), DateTime.Now, null, null)
```

UpdateCurrencyRates

Use this method to add/update currency rates. This method will take a list of **CurrencyRateInfo** and will add/update current currency rates.

UpdateCurrencyRates(List<CurrencyRateInfo> rates_)

If you are not enabled for Custom Currency Rates, the service will throw an exception. The **Date**, **currencyFrom**, and **currencyTo** are required fields. One of the fields for **Bid** or **Ask** is required. When adding/updating, if the exchange rate already exists for a given day and currency codes, it will be overwritten; otherwise, it will be added.

Example: Update Currency Rates

```
var client = new DataServiceClient();

//set credentials
client.ClientCredentials.UserName.UserName = "username";
client.ClientCredentials.UserName.Password = "password";

var newCurrencyRates = new List<CurrencyRateInfo>();

//Fill in new rates...

client.UpdateCurrencyRates(newCurrencyRates);
```

Assumptions/Requirements

- Client should be enabled for Custom Currency in order to get/set Custom Currencies.
- Once a client is enabled for Custom Currency, the client won't be able to use the currency rates provided by iLEVEL. This means the client becomes fully responsible for the population of currency rates.
- Currently up to 10,000 records can be added/updated/retrieved at once.

Note: If the currency rate is missing for a day, but exists for one of the previous days, the Web Service will not supply the missing date. That is, the currency rate must be provided for every date which will be used in calculations. If the currency conversion rate is missing and the iGet/Formula requires a currency conversion rate, a "No Data Available" message will be returned.

SUPPORT



Frequently Asked Questions (FAQ)

General Questions

Question	Answer
How do I get credentials to access Web Services?	<p>For having capability to access the web services in a sandbox environment, a new user needs to be created that has the Enable Data Service capability. To create a new user, please follow the following steps:</p> <ol style="list-style-type: none"> 1. Access the User Management page under the Tools section of the Web Portal 2. Create a new user by clicking the Add New User button located on the upper-right-hand corner of the page <p>Make sure the new API user persona has the following capabilities:</p> <ul style="list-style-type: none"> • Access Data • Enable Data Service • Access Unapproved Data • Edit Entities • Manage Entities • Edit Cash Entries, Enable iPuts, Enable iPut As of Date and Delete Data are only for clients with the iPut feature enabled
What application is best to use Web Services?	The client can use any application with Web Services, but it is recommended to use Visual Studio or SoapUI.
What programming languages are supported through the Web Services?	All programming languages are supported through Web Services. However, sample “applications” (e.g. code) are documented only for .NET (iGet), .NET(iPuts), Java, Python, cURL languages.
How long are the API versions supported?	<p>We commit to support the previous version of the current API, but we strongly encourage you to migrate to a new version of API after each major release.</p> <p>As an example, version "2018Q3" will be supported until the next major release.</p>
Is it possible to do a backup of the whole iLEVEL Database?	<p>API calls do allow you/your team to pull data from iLEVEL into your API console and directly into your data warehouse, should you choose to load directly. For more information about this capability, please refer to the Migrating into a Data Warehouse section of the guide.</p> <p>There is no single API call that allows you to perform a whole backup of your iLEVEL database; Data export is possible, though using specific requests (iGets) to retrieve the data. Once the initial load of all data occurs, you can use one API call (<i>GetUpdatedData</i>) to load all additional changes to the data into your database.</p>
How often are the API calls changing?	It is very rare for API calls to change either their method or their signature. The iGets and iPuts are stable over the past 5 years and the iGet/iPut documentation displays the history of the change over the past few years for clients to review.

What levels of security are in place for API calls?	Security measures include using the API channel protected using regular https (SSL). Additionally, every client who will access either the iGet or iPut feature will have a separate API user created with specific permissions. This API user will have a different username and password to ensure authentication steps for access to API.
Can clients use a different in-house integration tool instead of SoapUI?	Yes, clients may use another service other than SoapUI as long as it is in a programming language that can be supported by the system. However, we strongly recommend SoapUI so we can easily resolve issues that may come up with API calls.

Sandbox and Production

Question	Answer
What is the URL for SandBox Web Services when using to deploy code? (i.e. the WSDL)	<p>For the US: https://sandservices.ilevelsolutions.com/DataService/Service/2019/Q1/DataService.svc?singlewsdl</p> <p>For the EU: https://sandservices.ilevelsolutions.eu/DataService/Service/2019/Q1/DataService.svc?singlewsdl</p> <p>Note: this is not the URL for the portal view of Sandbox. The URL for the portal view of the Sandbox is as follows: https://sand.iLEVELsolutions.com/ for the US and https://sand.iLEVELsolutions.eu/ for the EU. The WSDL will change every release to reflect the current version as can be reflected in the year and quarter after the "Service" backslash.</p>
How do I enable data access in the Sandbox?	To enable data access in the Sandbox, the Web Services internal team creates access to the Sandbox. The team will create a copy of your Production environment in the Sandbox and enable the "Web Services API" setting for you to be able to use API calls in order to enter or retrieve data.

Why can't I use my Sandbox API credentials to load data into Production?

The first thing to address is making sure your production environment has been enabled with the appropriate settings by the Web Services team at iLEVEL.

Typically, once they have reviewed your code in the sandbox environment and approved it, they will ensure your production environment is set up. If they have yet to approve your code, the production environment will not be set up yet with those capabilities.

The second possible reason is that you cannot use your existing sandbox user credentials in the production environment. Just like when a user wants to access web services within the sandbox environment, a new API user needs to be created in production that has the Enable Data Service capabilities.

Data Entry

Question	Answer
What API calls do I use to load Cash Transactions versus Data Items?	<p>In iLEVEL there are two different types of data: (1) Periodic Data Items (2) Cash Transactions.</p> <p>"iPuts" via the API are only for loading in Periodic Data Items whereas cash transactions are loaded in differently. If you have the iPut feature enabled, you can use the iPutData call to load data. If the feature is not enabled, use the "AddOrUpdateDataItems" API call.</p> <p>Cash Transactions are loaded in API through the "AddOrUpdateInvestmentTransactions" call.</p>
What are data limits for iPuts using the API?	<p>We recommend limiting entity creation to batches of 200 entities and iPut batches 20,000 (as well as iGet batches).</p> <p>It is also recommended to combine several individual API requests into one large batch, but we don't recommend sending many batches of small numbers of API requests.</p>
What is the maximum amount of investment/cash transactions allowed to be put in at one time?	<p>The maximum amount of investment transactions allowed to be put in bulk is 20,000. If that number is passed, an exception will be thrown.</p> <p>However, we recommend splitting up transactions in batches of 10,000 to avoid time out errors.</p>
How do I bulk delete transactions?	<p>In order to delete transactions, you should first get their Investment Transaction ID using GetInvestmentTransactions (this call supports filtering) and then pass those to DeleteObjects.</p>

Is there documentation or are there sample scripts of the Cash Transactions API calls?

Yes, you can find sample scripts in the [Get All Cash Transactions](#) and [Perform Cash Transaction Entries](#) sections of this guide.

You can leverage the AddorUpdateInvestmentTransactions to create cash transaction entries.

Can I use ETL jobs to enable data entry?

The iLEVEL system supports programmatic data entry or retrieval using the iLEVEL API calls. If you (the client) would like to leverage ETL files, you will have to parse the data from the ETL files into your own internal data warehouse. Once parsed, you will be able to leverage the iLEVEL API to write the appropriate calls to enable data entry into the iLEVEL system.

Can I load data from the API directly into CSV or Excel?

The API, by default, does not generate a CSV file. Instead, it uses SOAP, which is much a better and easier alternative for integrating with external systems since it's already structured.

If you would rather another option, you are able to load the data into a CSV file with additional development effort on your end. You can structure the CSV file the way in which you want based on the data you are pulling back through the API calls. For example, if you would like to create a structure with specific data sets or specific entities, you are able to only populate those values in the CSV.

Why can't I (as a data service user) use iPut AsOf dates?

As of now, 'iPut As of Date' capability cannot be granted for Data-Service users.

Instead, the capability can be granted by the iLEVEL team running a script in the back-end.

Why aren't I able to use/see iPuts in the API?

Only some clients, who purchase the API package which includes iPuts, are specifically granted the ability to load data via iPuts. Please ensure the API user's capabilities include the iPut capacity via the User Management Web Portal Page.

Additionally, a member of the internal team will have to run the "enable iPut script" for you.

Data Retrieval

Question

Answer

What output does the data provide?

The data is provided in XML output but can also be changed to the output as you choose (e.g. CSV file).

How do I access my Qval data?

Retrieving data through Qval is handled through a different service. If you are interested in pursuing this data package, reach out to your Account Management representative.

Currency Rates

Question	Answer
How does one change the type of currency rates used?	<p>It is not possible to change the type of used rates from the iLEVEL Web Portal.</p> <p>Rather, you should reach out to the internal iLEVEL team to enable the setting for which clients can load custom currency rates and populate custom rates through the UpdateCurrencyRates call.</p>
Can I submit FX rates weekly? Or monthly?	<p>Since the behavior introduced by these options is a bit complex, you are only able to select the submission frequency upon set-up and will not be able to change once the environment has been established. The currency rate's behavior can be customized for a particular client need. This probably will require some client-specific modifications.</p>

Issues/Errors

Question	Answer
When there are issues with transmission failure, what's the process for re-sending/addressing?	<p>Typically, iPut errors can occur for several reasons including loading data to a data item that has not been created or loading data to a data item that is strictly a calculated account. These are the same types of validations which occur in the Excel add-in if a user attempts to load an iPut with errors relating to the parameters of input.</p>
What happens when the Web Service application throws an error? What steps should I take?	<p>If there is an error, you should send a note to the dedicated email distribution list with the username and the "request" (i.e. the code which caused the error) in the email. A member of the team will contact you shortly.</p>
Why am I no longer able to access and use my data in my sandbox environment? Why have my previously established capabilities been turned off?	<p>With respect to the connectivity issues in Sandbox environment, they are specifically due to</p> <ol style="list-style-type: none"> 1. Weekly refresh from Production to Sandbox 2. Enabling the appropriate settings. <p>The weekly refresh copies over your production environment to Sandbox which can disable settings and users, if there is a difference between Sandbox and Production.</p> <p>If the issues continue to surface, please contact us here and we will investigate the issue with priority.</p>
What happens if a user submits a batch of iGets (or iPuts) with 100 items of which 2 iGets fail?	<p>If this were to occur, the entire batch does not fail. Instead, only the 2 items that failed will not submit their validations. All other iGets will be submitted to the database. The only instance where this may not be true is if there is an application error such as a connection timeout or database timeout, but these instances are rare.</p>

Glossary of Terms

Asset – a general term used to identify an investment. An asset can be an investment made in a company or in real estate, for example.

Cache – a component that transparently stores data so that future requests for that data can be served faster.

Chunk – a segmentation of data. In our case, chunks consist of a maximum of 20,000 iGets to ensure performance standards.

Data – information that is applied to any object or object-to-object association.

Data Item – a categorization of a particular type of data, sometimes also called an account. Some data items span industries and/or investments (for example “Total Revenue”). Some data items are unique to particular assets or investments (for example, “Shipping Tonnage” might be unique to assets in the Shipping industry). Most data items in the iLEVEL Platform can accept manual entry, but many accept data in the form of formulas or other types of non-manual calculations. See the “Calculated Data” section.

Fund – a collection of assets or investments. General partners use funds as investment vehicles by investors known as Limited Partners (LPs).

iGet – iLEVEL’s proprietary formula that is used to grab data from the iLEVEL data tables. iGets can be invoked manually through the iLEVEL Excel Add-in, or through iLEVEL’s Web Services offering.

Memory – the physical devices used to store programs (sequences of instructions) or data (e.g., program state information) on a temporary or permanent basis

Owners – any entity that has an investment or relationship in a particular asset.

Reporting Period – a period of time in which data is reported for an investment (asset, fund, etc.).

Scenario – a customizable categorization of a collection of data. Some common scenarios are “Actual”, “Budget”, and “Forecast”.

Segment – a hierarchical categorization of data that can be created using the iLEVEL Platform. Segment data can be rolled up through the hierarchy to the asset.

iPut – iLEVEL’s proprietary formula that is used to put data into the iLEVEL data tables. iPuts can be invoked manually through the iLEVEL Excel Add-in, or through iLEVEL’s Web Services offering