# IP Data Direct – Patents, Version 2

## *Technical Reference Guide*

**January 2017**

**Version 2.8**

# TABLE OF CONTENT

## Intended Audience

This guide is primarily intended for the ***software developer*** developing workflow solutions to integrate a custom solution with LexisNexis IP Data Direct – Patents (IPDD), Version 2. Sections of this document are furthermore aimed at ***content managers, product managers*** and ***operations managers*** trying to understand the capabilities of IPDD as well as options for integration into a custom workflow and the means of most effectively placing the system into operation.

## Executive Summary

IP Data Direct – Patents (IPDD), Version 2 represents the latest release of LexisNexis' industry-leading Patent Data delivery platform and provides a significant enhancement from Version 1 both in terms of features as well as ease of use.

This Technical Reference Guide is aimed at the software developer, content manager, product manager and operations manager tasked at integrating IPDD into their in-house solution. The guide provides usage and implementation scenarios with descriptions as well as listing design considerations in the first portion of the document and details on the API capabilities and functionalities with code samples in the second portion of the document.

Further and more detailed information can be obtained from the technical documentation and through the LexisNexis Intellectual Property Sales Team and LexisNexis Univentio Technical Support, contact details for which are provided in Appendix A of this document.

# Overview

This chapter will provide a high-level overview of the IPDD service and will explain the underlying concept of IPDD. In this section we also discuss the service components and how these are linked to form the service.
Section 4 of this guide follows-on to provide an overview of different common usage scenarios and best practises.

## 3.1 Background[1]

The concept underlying IPDD is based upon the idea that a platform for the supply of patent information should provide clients with the means to easily retrieve newly published data and while at the same time maintaining their collections up to date by means of updating publications in the collection as and when a chance occurs. Other design criteria were that IPDD should allow expert users the ability to perform complex calls on the API while novice users should have simple means of accessing the data as well as the ability for different types of applications to access IPDD in a variety of manners. API calls range from inventory and status calls to single element publication retrieval right through to complete batch retrieval functions. Section 0, Technical Service Description provides an in-depth overview of all the available requests. The high-level system architecture is detailed in Figure 1 below.

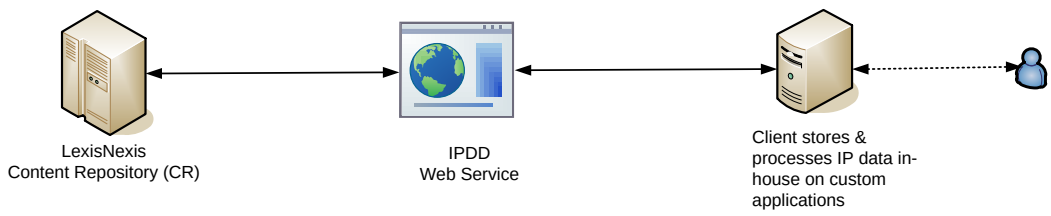



Figure 1: High-Level System Architecture

[1] Please also refer to the IPDD Technical Marketing Guide for additional background information on IPDD. This document can be requested via the LexisNexis IPDD Customer Support

Fundamentally the IPDD service interface can be separated into two parts as follows:

- Part 1: The IPDD service API that is based upon SOAP and REST. This portion of the interface offers the broadest set of functionality but at the same time follows a ‘strict’ approach in that it will not attempt to ‘interpret’ the call placed. This means, for example, that if document US123456B1 is requested but the correct kind code is B2, the service will return zero documents.

- Part 2: The IPDD service also offers a URL interface that has additional logic added to it. This means that the URL interface will, compared to the SOAP and REST API, provide ‘support’ in matching ‘incorrect’ number formats to the LexisNexis format but also for example serve up a WO PDF if a EuroPCT is requested using the EP publication number.

All functionality on the interface(s) can be mixed and matched as required.

### 3.2 Data Flow [2]

When implementing solutions based upon IPDD as a data-delivery platform it is helpful to understand the flow of data through the LexisNexis system since this provides better understanding of the way publications can be retrieved through the API. The data flow is indicated in figure 2 below.

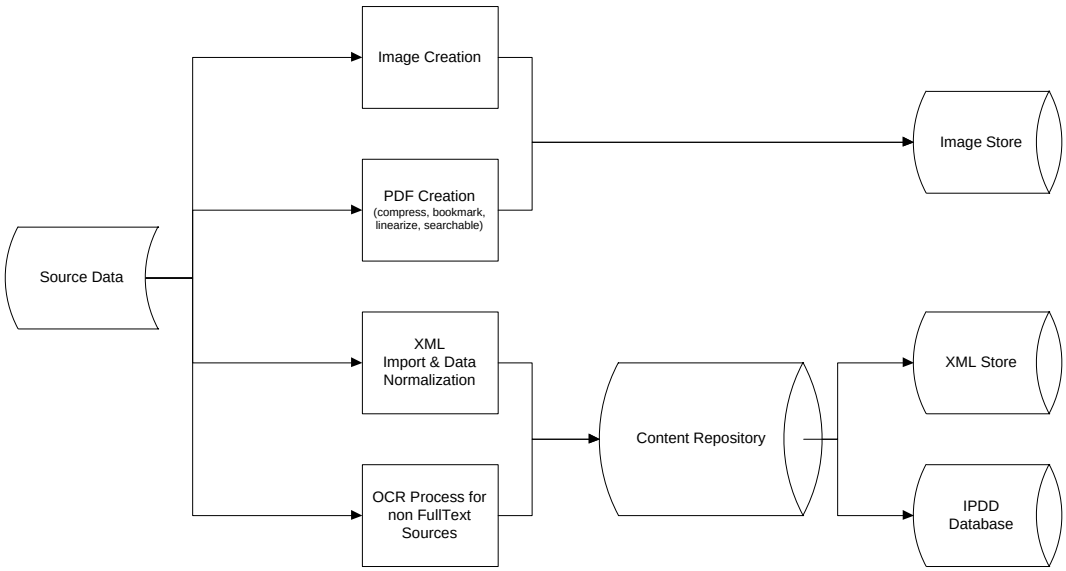



Figure 2: High-Level Data Flow

The basic flow of data as indicated in figure 2 above is that the data for a particular record, once added to the LexisNexis Content Repository is re-published to the LexisNexis XML, PDF and Image stores respectively. The image store contains all of the Image and PDF data where the content repository is the database storage of the XML data. This XML data is extracted to a separate XML store which is the supplier for XML data through IPDD. The IPDD database contains a small subset of the data

[2] Please note that per collection detailed delivery overviews can be obtained from LexisNexis IPDD Customer Support which detail per collection which data elements get delivered in which interval

and account information and is synced with the content repository. Specific attention should be paid to the fact that LexisNexis *merges* data from different sources and uses a concept of source priorities per field to determine which data source takes precedence. This also means that records get completed over time as the sources publish the data over a period of time. See figure 3 for a graphical representation of this process where the data is generated from 3 sources arriving at a different time. The source priority indicates which source is the leading one and which ones are used to add data or to be replaced with data from a source with a higher priority.

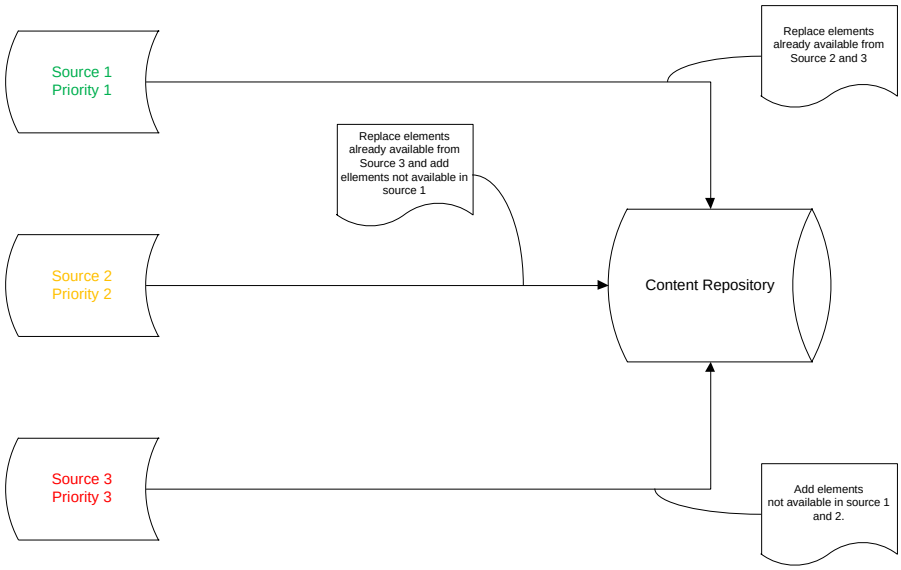



Figure 3: Source Priorities

Once data is available in the stores, the IPDD database is made aware of the availability of the data and will become retrievable through the IPDD API.

### 3.3 Technical environment

The IPDD service is built upon the Microsoft .Net framework version 4.0. It uses Windows Communication Foundation 4.0 (WCF) to expose the service. The data transport is handled via HTTP and encapsulated within the SOAP (Simple Object Access Protocol) or REST (Representational State Transfer).

### 3.4 Accessing the service

When trying to understand the access to the IPDD service it is important to distinguish between the control and the data layer. The control layer can be accessed through either SOAP or REST and by using the corresponding endpoint. Access can be gained by using plain SOAP or REST calls. This however is *not* the preferred way of accessing the Corporate IPDD service since multiple requests are required and the batch data being streamed is typically of large volume. The preferred scenario is using a development environment through which the service is being added as a web service reference. See also Adding Service Reference Examples. The data layer can provide HTTP and FTP as means of data retrieval.

### 3.5 Service date storage

One of the biggest advantages in IPDD is the fact that the IPDD system is stateful on the server side when using either the NewRequest variable or the UpdateRequest variable. This means that when a client requests publications and uses one or both of the above mentioned variables, the request date is stored on the server side. This way IPDD is capable of determining the exact delta between the data that the customer has in his current collection and the changes and additions that need to be delivered as updates for the customer system to remain up to date. Also, the calls

above are the easiest (and recommended) way to maintain synchronization with the LexisNexis Content Repository[3]. The following example shows how this works.

*Example: Requesting new US publications.*

*The client requests US publications and uses the NewRequest variable. The last time newly added US publications where requested was on January 1st 2010. The client gets all of the newly added US publications from January 1st 2010 until present.*

The advantage of this system is that a client does not need to keep track of a date or check if he has all the latest data. A simple check for either new of changed publications is enough to keep his system up to date.

Note: A NewRequest call only delivers newly added publications while the UpdateRequest delivers both newly added and changed publications.

[3] Please refer to the Best Practises section of this document also for best practises on how to handle updates

## 3.6 Request Flow

The following diagram provides an overview on how IPDD can be used to request data. The following chapter will describe some scenarios on how a client can efficiently use IPDD to retrieve the data and remain up to date.

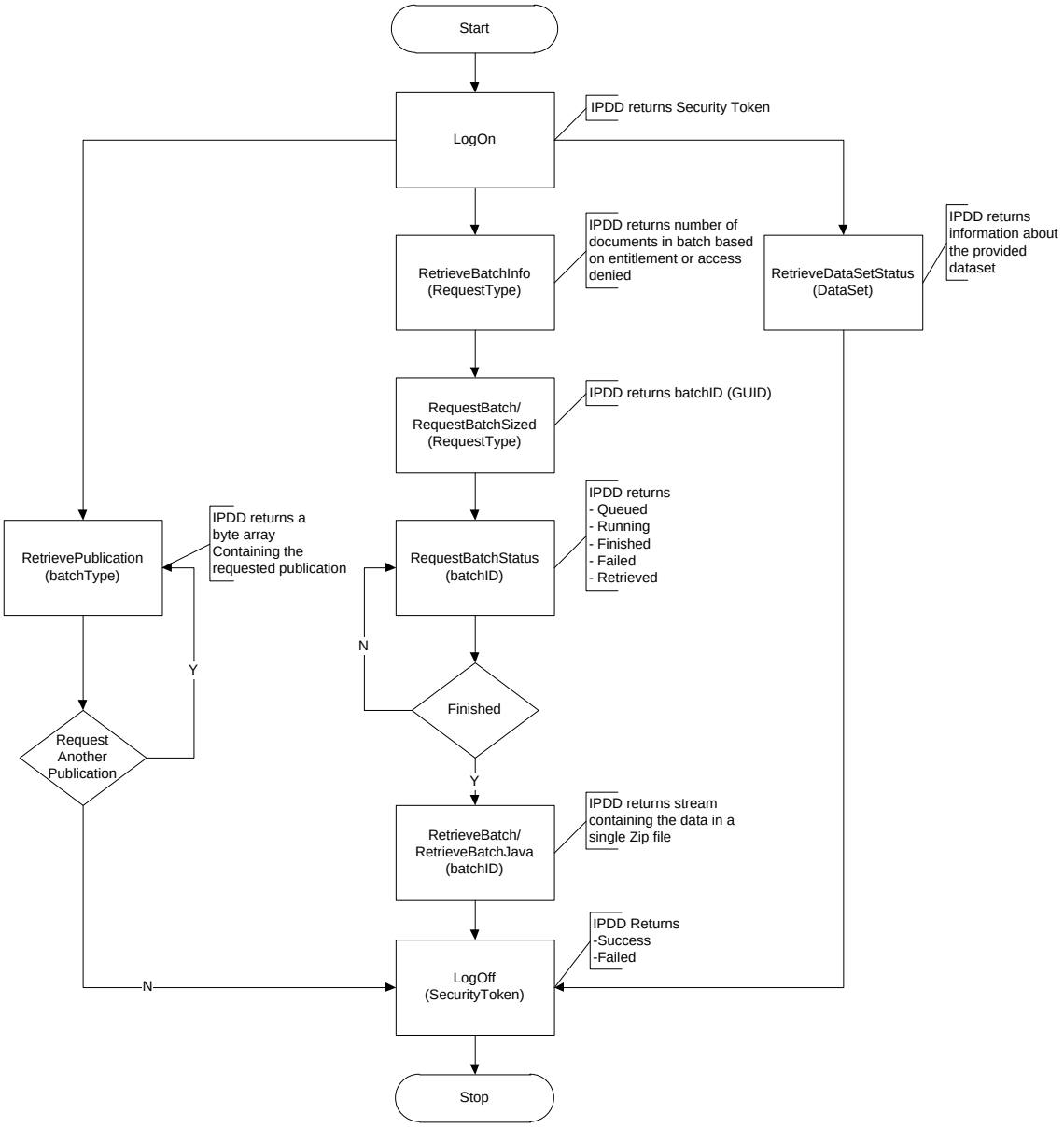



Figure 4: IPDD Request Flow

# "Best Practice" Scenarios for Data Retrieval

There are many possible approaches for retrieving data via IPDD. This chapter will describe of the most common scenarios and will outline a "best practice" approach related to these scenarios. The scenarios will be split into an IPDD service portion and an IPDD Gateway portion.

## 4.1 Usage Scenarios

The IPDD system has been designed with the utmost level of flexibility and customization options in mind. The system provides multiple ways to ensure synchronization through inventory calls, multiple means of retrieving data to meet our customers varying needs but also simple means of performing data retrieval. Furthermore, while customers can custom-develop the integration of their workflow with IPDD, LexisNexis also supplies a retrieval client[4] for customers who do not wish to develop their own integration.

There are multiple means of retrieving data via the IPDD service. The data can for example be retrieved based on a date or date range or a variety of other parameters. However, the preferred way is to use the 'New' and 'Update' requests to keep the users system up to date with the latest publications. The advantage is that by using these two request types, IPDD maintains state on behalf of the customer. This means that the user does not need to ensure that he maintains synchronization with the LexisNexis system but IPDD ensures that. The consequence being that the user always receives exactly the documents he has never seen before (*NewRequest*) and that have changed since the last update session (*UpdateRequest*).

The table below lists the different scenarios described as well as the target customer type per scenario.

[4] The IPDD Retrieval Client supports Microsoft Windows only

| | Patent Research Platforms | Generic Research Platforms | Analytics / Patent Lifecycle Management | Specialized Applications |
|---|---|---|---|---|
| **Scenario 1**: New and Changed Publications in a single Request | X | X | X | |
| **Scenario 2**: New and Changed Publications in two separate Requests | X | X | X | |
| **Scenario 3**: Retrieval based upon Publication Date | | X | | |
| **Scenario 4**: Single Publication Retrieval | X | | | X |
| **Scenario 5**: Data Elements Retrieval | | | | X |
| **Scenario 6:** Retrieval based on publication numbers | X | | X | X |

Table 1: “Best Practice” Update Scenarios per Platform Type

All of the scenarios are also available as working sample projects in C# and Java. A zip file with sample code is available for download upon request [5] [6].

[5] Please contact LexisNexis Univentio Customer Support for access to these code samples

[6] Please refer to the Technical description chapter for a technical explanation of each request. It is advised to use the code examples file for a good overview of how to use these scenarios. An example of how to add IPDD as a reference to a code project is shown in Adding Service Reference Examples.

### 4.1.1 Scenario 1: New and Changed Publications as single Request

Who would use this approach?

This approach to IPDD data retrieval would typically be used by a customer who needs to load full authorities on an ongoing basis into an internal database. Examples are Patent Information Providers and Corporate Customers with an internal database[7].

Why would you use this approach?

For any IPDD customer using the platform to retrieve data into an in-house database this approach is the preferred “best practice” approach because:

- The IPDD platform maintains state on behalf of the customer
- Customers get a full delta of their current state and the LexisNexis Content Repository
- Implementation is simple
- Operational complexity is low

Description of Scenario

This scenario is the most efficient way to make sure that a client receives the latest new and updated publications from IPDD. It uses the *UpdateRequest* variable to get all of the new and updated publications in one single request, whereby the request delivers both all new publications (i.e. the publications the customer database has no prior knowledge of) and the changed publications.

[7] Corporate Customers would use a flavour of IPDD, called IPDD Corporate. IPDD Corporate allows corporate users to retrieve subsets of collections using custom filters

## Process Flow

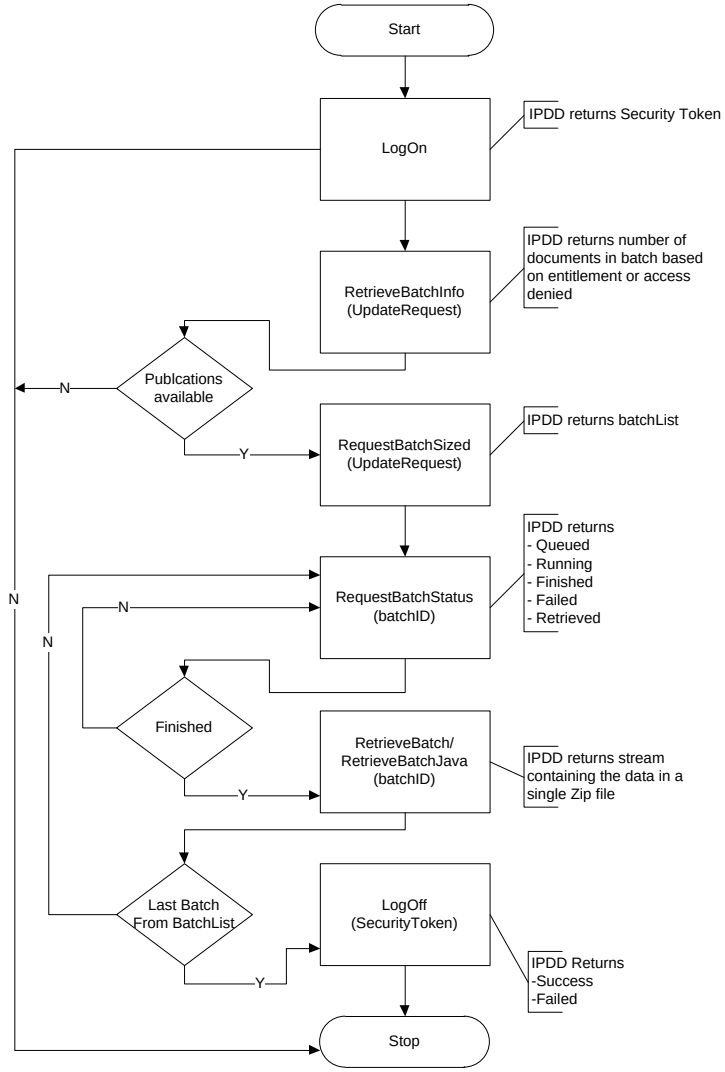



Figure 5: IPDD “Best Practice” Scenario 1 Process Flow

Technical Details

### *Logon*

The first step is to logon to the service and acquire a valid security token, this token is valid for 12 hours after which a new token need to be requested.

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = port.logOn(id, password);
```

### *Creating the UpdateRequest variable*

The *UpdateRequest* variable is used to check for new and updated publications since the last time the *UpdateRequest* was used.

C#

```
//Create the update request variable to get all new and changed publications
UpdateRequest updateRequestVariable;
updateRequestVariable = new UpdateRequest();

//Set the acquired securitytoken
updateRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
updateRequestVariable.DataSet = dataset;
//Set the datatype
updateRequestVariable.DataType = dataType;
```

Java

```
//Create the update request variable to get all new and changed publications
UpdateRequest updateRequestVariable;
updateRequestVariable = new UpdateRequest();
```

```
//Set the acquired securitytoken
updateRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
updateRequestVariable.setDataSet(dataSet);
//Set the datatype
updateRequestVariable.setDataType(dataType);
```

### *Check if new/updated publications are available*

The *RetrieveBatchInfo* call is used to get a count of available publications. Based on this count the client can check if there is data available and how much. The request variable used in this call is the previously set *Updaterequest*.

C#

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.RetrieveBatchInfo(updateRequestVariable);
```

Java

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.retrieveBatchInfo(updateRequestVariable);
```

### *Requesting the publications*

The publications are requested by using the *RequestBatchSized* call. LexisNexis advises to use the sized call since this call supports batches higher than 50.000 publications. This request will return a list of batches that can be retrieved when they are finished.

C#

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches;
listOfBatches = client.RequestBatchSized(updateRequestVariable, 20000);
```

Java

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches = new BatchList();
listOfBatches = client.requestBatchSized(updateRequestVariable, new Long(20000));
```

### *Check for the batch status*

The status will be checked for the first batch in the list, since this is the first one to be processed by IPDD. In this scenario the client checks every 5 minutes for the batch status. If the status changes to *finished* the batch will be retrieved.

C#

```
BatchStatus status;

//Refresh the client to avoid connection problems
using (ServiceClient statusClient = new ServiceClient("basicHttp"))
{
       //check for the batchstatus once every 5 minutes and proceed when the batch is
       finished
       status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
       batch.BatchId);
}


while (status.Status != Status.Finished)
{
      //Check if the status is failed
      if (status.Status == Status.Failed)
             throw new Exception("Batch Failed.");

      //Wait for 5 minutes
      System.Threading.Thread.Sleep(300000);

      //Refresh the client to avoid connection problems
      using (ServiceClient statusClient = new ServiceClient("basicHttp"))
      {
             //Request the batch status
              status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
              batch.BatchId);
      }
}
```

Java

```
BatchStatus status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());


while (status.getStatus() != Status.FINISHED)
{
        //Check if the status is failed
```

```
        if (status.getStatus() == Status.FAILED)
                throw new Exception("Batch Failed.");

        Thread.sleep(300000);

        status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());
}
```

### *Retrieving the batch*

The batch will be retrieved as a data stream. This way the data is retrieved in small parts to avoid a high memory load when retrieving large files. The data is retrieved with the help of the position parameter. The use of this parameter enables the client to restart a broken stream in case of a connection problem.

C#

```
// Allow 1000 retries per batch retrieval when a connection breaks
//
while (retries < 1000 && retry)
{
   tempPosition = 0; // always reset (will be essential in case of a read error and
the tmpPosition is unreliable)

   //Retrieve Batch
   try
   {
      //Use a new client connection to ensure a stable connection
      using (ServiceClient downloadClient = new ServiceClient("basicHttp"))
      {
         Stream data = downloadClient.RetrieveBatch(
            logonInfo.SecurityToken, batch.BatchId, position);

         while ((i = data.Read(buffer, 0, 4096)) != 0)
         {
            file.Write(buffer, 0, i);

            //Store the current stream/byte position
            tempPosition = data.Position;
         }

         //Batch retrieved, set retry to false and close the stream
         data.Close();
         retry = false;
      }
   }
   //Catch IOException which happens when the download stream breaks
```

```
   catch (IOException)
   {
      //Add the current byte position to the already retrieved ones
      position += tempPosition;

      Console.WriteLine("  IOError while downloading {0}, will retry at position {1}",
batch.BatchId, position);
      retries++;
   }
}
```

## Java

```
//Generate a batch filestream
FileOutputStream file = new FileOutputStream(String.format("{0}data-{1}.zip", batchFolder,
batchCount));

//Allow 1000 retries per batch retrieval when a connection breaks
while (retries < 1000 && retry)
        {
        tempPosition = 0; // always reset (will be essential in case of a read error and the tmpPosition
is unreliable)
        //Retrieve batch
        try
        {
                DataHandler result = client.retrieveBatchJava(logonInfo.getSecurityToken(),
                batch.getBatchId(), new Long(0));

                InputStream data = result.getInputStream();

                while ((i = data.read(buffer, 0, 4096)) > 0)
                {
                        file.write(buffer, 0, i);

                        tempPosition += i;
                }

                data.close();
                retry = false;
        }
        catch(IOException ex)
        {
```

```
                position += tempPosition;
                retries++;
        }
}
```

### *Proceeding to the next batch*

Check the next batch in the list to see if the status is finished. If there are no more batches in the list proceed to logging off from IPDD.

### *Logoff*

Logoff from IPDD by calling the *LogOff* request.

C#

```
using (ServiceClient logoffClient = new ServiceClient("basicHttp"))
{
      //Logoff from IPDD
      logoffClient.LogOff(logonInfo.SecurityToken);
}
```

JAVA

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

### 4.1.2 Scenario 2: Retrieve the new and changed publications separately

Who would use this approach?

This approach to IPDD data retrieval would typically be used by a customer who needs to load full authorities on an ongoing basis into an internal database, but needs to spread updates over a period of time for example to allow for indexing (unlike Scenario 1 where all data is loaded in a single go). Examples are Patent Information Providers and Corporate Customers with an internal database[8].

Why would you use this approach?

For any IPDD customer using the platform to retrieve data into an in-house database this approach is the preferred "best practice" approach because:

- The IPDD platform maintains state on behalf of the customer
- Customers get a full delta of their current state and the LexisNexis Content Repository
- Implementation is simple
- Operational complexity is low
- The new publications can be delivered before the updates in two separate processing steps

Description of Scenario

This scenario is the most efficient way to make sure that a client receives the latest new and updated publications from IPDD. It uses the *UpdateRequest* variable to get all of the new and updated publications in one single request, whereby the request delivers both all new publications (i.e. the publications the customer database has no prior knowledge of) and the changed publications.

This scenario allows a customer to make use of the same advantages that scenario 1 provides while splitting the delivery into two distinct steps. The new publications and changed publications are retrieved via *NewRequest* and *ChangedRequest* variable respectively. First all of the new publications are requested by using the

[8] Corporate Customers would use a flavour of IPDD, called IPDD Corporate. IPDD Corporate allows corporate users to retrieve subsets of collections using custom filters

*NewRequest* variable and after processing and retrieval the updated publications are requested and retrieved by using the *UpdateRequest* variable.

## Process Flow

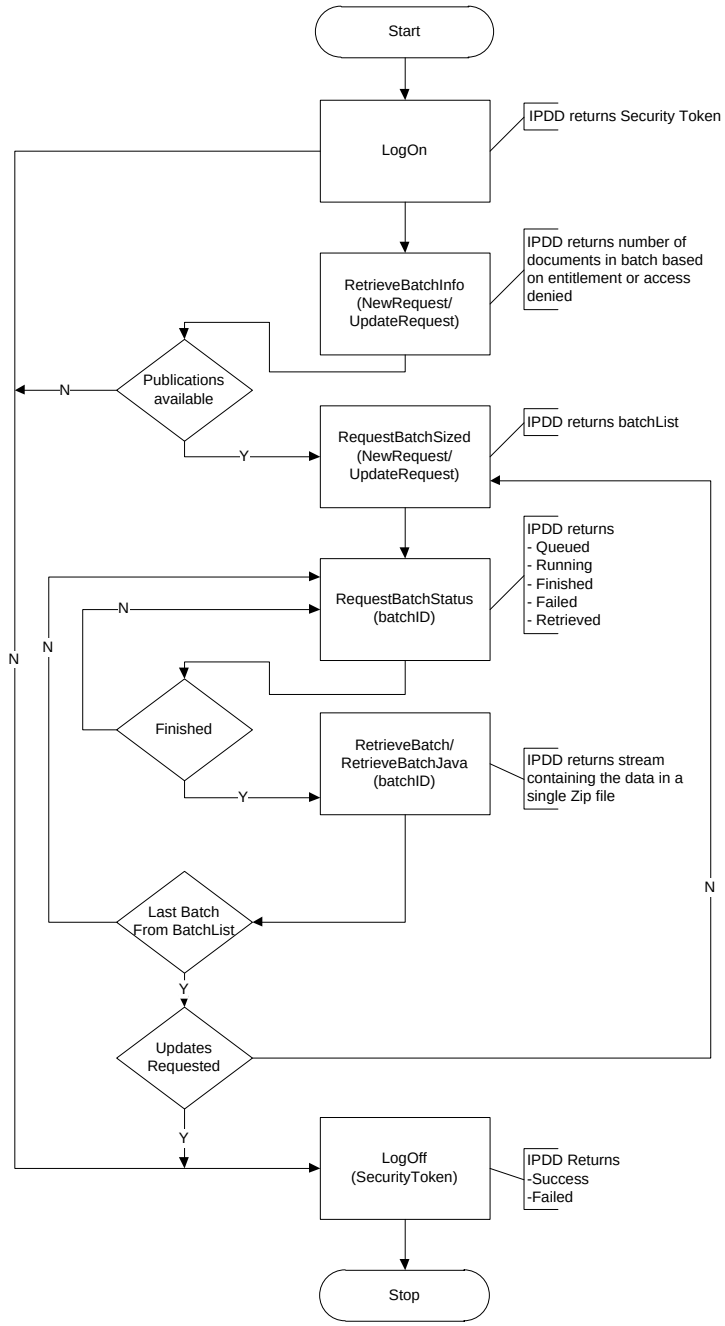



Figure 6: IPDD “Best Practice” Scenario 2 Process Flow

### 4.1.3 Scenario 3: Retrieve Publications based on Publication Date

Who would use this approach?

This approach to IPDD data retrieval would typically be used by a customer who needs to load full authorities on an ongoing basis into an internal database, and wants to do this by using the publication date of the authority. Examples are Patent Information Providers and Corporate Customers with an internal database

Why would you use this approach?'

For any IPDD customer using the platform to retrieve data into an in-house database who wants to keep track of the data internally by using the publication date publication.

- The IPDD platform delivers the data based on the publication date
- No state is being maintained, this needs to be implemented on the client side
- Implementation is simple
- Operational complexity is moderate to high
- Only newly added publications are being retrieved

Description of Scenario

This scenario enables a customer to specify from which publication date the data should be retrieved. This has to be done on a regular basis (for example once a week) to ensure that the data keeps up to date. The customer keeps track of the publications dates already retrieved to avoid the data is being processed which already has been loaded into the internal database.

Note: In this scenario the customer has no means of maintaining state and thus understanding what documents might have changed!

## Process Flow

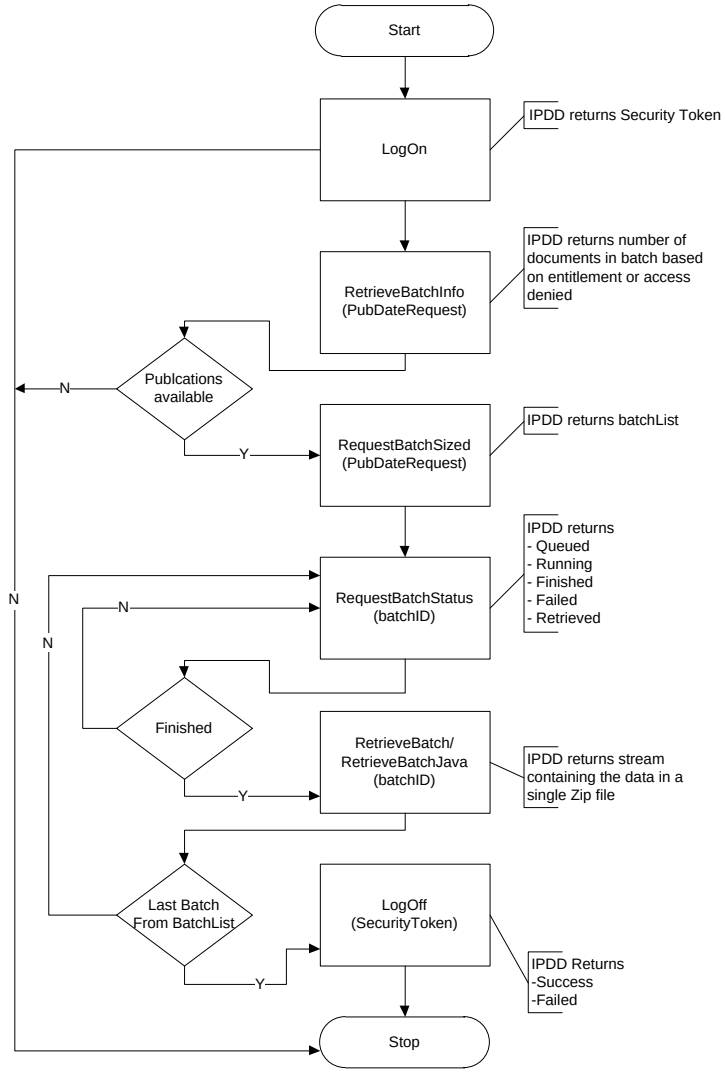



Figure 7: IPDD “Best Practice” Scenario 3 Process Flow

Technical details

### *Logon*

The first step is to logon to the service and acquire a valid security token, this token is valid for 12 hours after which a new token need to be requested.

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = port.logOn(id, password);
```

### *Generate the PubDateRequest variable*

The PubDateRequest variable is set to get data from the provided dataset published on the provided date.

C#

```
//Create the publication request variable for getting publications based on
publication date
PubDateRequest PubDateRequestVariable;
PubDateRequestVariable = new PubDateRequest();

//Set the aquired securitytoken
PubDateRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
PubDateRequestVariable.DataSet = dataset;
//Set the datatype
PubDateRequestVariable.DataType = dataType;
//Set the publciationdate and convert the date to UTC
PubDateRequestVariable.PublicationDate = pubDate.ToUniversalTime();
```

Java

```
//Create the publication request variable for getting publications based on publication date
PubDateRequest pubDateRequestVariable;
pubDateRequestVariable = new PubDateRequest();

//Set the aquired securitytoken
pubDateRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
pubDateRequestVariable.setDataSet(dataSet);
//Set the datatype
pubDateRequestVariable.setDataType(dataType);
//set the publicationDate
pubDateRequestVariable.setPublicationDate(date);
```

### *Check if publications are available*

The *RetrieveBatchInfo* call is used to get a count of available publications. Based on this count the client can check if there is data available and how much. The request variable used in this call is the previously set *PubDateRequest*.

C#

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.RetrieveBatchInfo(updateRequestVariable);
```

*Java*

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.retrieveBatchInfo(pubDateRequestVariable);
```

### *Request the publications*

The publications are requested by using the *RequestBatchSized* call, it is advised to use the sized call since this call supports batches higher than 50.000 publications. This request will return a list of batches that can be retrieved when they are finished.

C#

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches;
listOfBatches = client.RequestBatchSized(updateRequestVariable, 20000);
```

Java

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches = new BatchList();
listOfBatches = client.requestBatchSized(pubDateRequestVariable, new Long(20000));
```

### *Check for the batch status*

The status will be checked for the first batch in the list, since this is the first one to be processed by IPDD. In this scenario the client checks every 5 minutes for the batch status. If the status changes to *finished* the batch will be retrieved.

C#

```
BatchStatus status;

//Refresh the client to avoid connection problems
using (ServiceClient statusClient = new ServiceClient("basicHttp"))
{
       //check for the batchstatus once every 5 minutes and proceed when the batch is
       finished
       status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
       batch.BatchId);
}


while (status.Status != Status.Finished)
{
      //Check if the status is failed
      if (status.Status == Status.Failed)
             throw new Exception("Batch Failed.");

      //Wait for 5 minutes
      System.Threading.Thread.Sleep(300000);

      //Refresh the client to avoid connection problems
      using (ServiceClient statusClient = new ServiceClient("basicHttp"))
      {
             //Request the batch status
              status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
              batch.BatchId);
      }
}
```

Java

```
BatchStatus status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());


while (status.getStatus() != Status.FINISHED)
{
//Check if the status is failed
if (status.getStatus() == Status.FAILED)
throw new Exception("Batch Failed.");


Thread.sleep(300000);


status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());
}
```

### *Retrieve the batch*

The batch will be retrieved as a data stream, this way the data is retrieved in small parts to avoid a high memory load when retrieving large files. The data is retrieved with the help of the position parameter. The use of this parameter enables the client to restart a broken stream in case of a connection problem.

C#

```
//Generate a batch filestream
file = new FileStream(string.Format("{0}data-{1}.zip", batchFolder, batchCount),
FileMode.Create);

//Allow 1000 retries per batch retrieval when a connection breaks
while (retries < 1000 && retry)
{
	//Retrieve Batch
	try
	{
	//Use a new client connection to ensure a stable connection
	using (ServiceClient downloadClient = new ServiceClient("basicHttp"))
	{
		Stream data = downloadClient.RetrieveBatch(logonInfo.SecurityToken,
		batch.BatchId, position);

		while ((i = data.Read(buffer, 0, 4096)) != 0)
		{
			downloaded += i;
			file.Write(buffer, 0, i);

			//Store the current stream/byte position
			tempPosition = data.Position;
		}

		//Batch retrieved, set retry to false and close the stream
		data.Close();
		retry = false;
	}
	}
	//Catch IOException which happens when the download stream breaks
	catch (IOException)
	{
		//Add the current byte position to the already retrieved ones
		position += tempPosition;
		retries++;
	}
}
```

Java

```
//Generate a batch filestream
FileOutputStream file = new FileOutputStream(String.format("{0}data-{1}.zip", batchFolder,
batchCount));

//Allow 1000 retries per batch retrieval when a connection breaks
while (retries < 1000 && retry)
	{
	//Retrieve batch
	try
	{
		DataHandler result = client.retrieveBatchJava(logonInfo.getSecurityToken(),
		batch.getBatchId(), new Long(0));

		InputStream data = result.getInputStream();

		while ((i = data.read(buffer, 0, 4096)) > 0)
		{
			file.write(buffer, 0, i);

			tempPosition += i;
		}

		data.close();

		retry = false;
	}
	catch(IOException ex)
	{
		position += tempPosition;
		retries++;
	}
}
```

### *Proceed to the next batch*

Check the next batch in the list to see if the status is finished. If there are no more batches in the list proceed to logging off from IPDD.

### *Logoff*

Logoff from IPDD by calling the *LogOff* request.

C#

```
using (ServiceClient logoffClient = new ServiceClient("basicHttp"))
{
      //Logoff from IPDD
      logoffClient.LogOff(logonInfo.SecurityToken);
}
```

JAVA

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

### 4.1.4 Scenario 4 Single Publication Retrieval

Who would use this approach?

This approach to IPDD data retrieval would typically be used by a customer who wants to retrieve a single publication to be used inside a customer application for viewing when doing research. This way a customer does not have to store all of the data on site but only a subset and retrieve the complete publication when needed.

Why would you use this approach?

The advantage of single publication retrieval is the real time delivery of the data. When a customer request a single publication the data is being generated on the fly from the IPDD storage and not processed as a batch delivery. This makes the single publication retrieval an efficient way to view publications directly from IPDD and not use an internal storage mechanism to store all the data.

Description of Scenario

The customer retrieves a single publication for viewing inside an application or to do research on.

Note: This can be applied to both XML and PDF documents.

## Process Flow

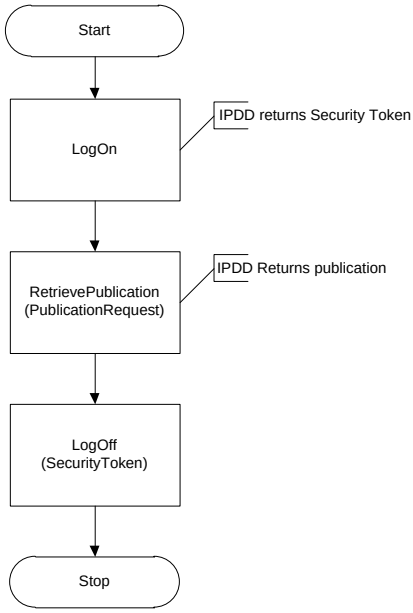



Figure 8: IPDD “Best Practice” Scenario 4 Process Flow

## Technical Details

### *Logon*

The first step is to logon to the service and acquire a valid security token, this token is valid for 12 hours after which a new token need to be requested.

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = client.logOn(id, password);
```

### *Retrieve single Publication*

Create a publication request type and set the properties. In this scenario a US publication is being requested with number 6,000,000 and kind A.

C#

```
//Create the single publication request variable
PublicationRequest PubRequestVariable;
PubRequestVariable = new PublicationRequest();

//Set the aquired securitytoken
PubRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
PubRequestVariable.DataSet = dataset;
//Set the datatype
PubRequestVariable.DataType = dataType;
//Set the Authority
PubRequestVariable.Authority = "US";
//Set the document number
PubRequestVariable.Number = "6000000";
//Set the kind code
PubRequestVariable.Kind = "A";

byte[] publication = client.RetrievePublication(PubRequestVariable);
```

JAVA

```
//Create the single publication request variable
PublicationRequest pubRequestVariable;
pubRequestVariable = new PublicationRequest();

//Set the aquired securitytoken
pubRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
pubRequestVariable.setDataSet(dataSet);
//Set the datatype
pubRequestVariable.setDataType(dataType);
//Set the authority
pubRequestVariable.setAuthority("US");
//Set the document number
pubRequestVariable.setNumber("6000000");
//Set the kind code
pubRequestVariable.setKind("A");
```

byte[] publication = client.retrievePublication(pubRequestVariable);

### *Logoff*

Logoff from IPDD by calling the *LogOff* request.

C#

```
using (ServiceClient logoffClient = new ServiceClient("basicHttp"))
{
      //Logoff from IPDD
      logoffClient.LogOff(logonInfo.SecurityToken);
}
```

JAVA

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

### 4.1.5 Scenario 5: Data Elements Retrieval

Who would use this approach?

This approach to IPDD data retrieval would typically be used in a specialized application for which only a part of the publication is needed. A customer for example performs researches on family and English title data inside a publication and therefore only needs the family and English title sections.

Why would you use this approach?

The single data element retrieval only delivers the data needed for a customer which reduces the size of the data needed to download and minimizes the storage needed for the customer to store the data.

Description of Scenario

The customer retrieves a single publication and enables only the family section. The IPDD service returns the family section of the requested publication.

Process Flow

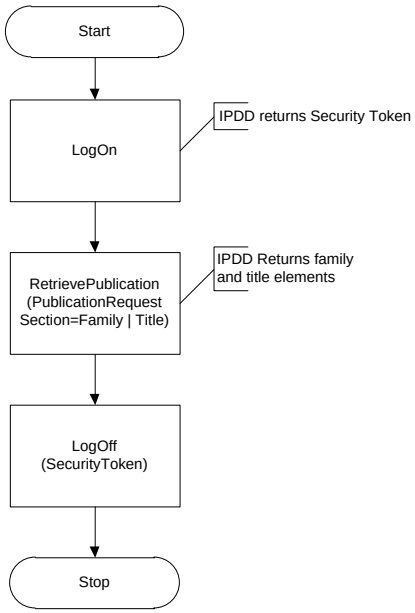



Figure 9: IPDD “Best Practice” Scenario 5 Process Flow

Technical Details

### *Logon*

The first step is to logon to the service and acquire a valid security token, this token is valid for 12 hours after which a new token need to be requested.

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = client.logOn(id, password);
```

### *Retrieve single data element*

Retrieve a single data element from US publication 6,000,000. In this scenario the Family data is being retrieved by setting the publication section property.

C#

```
//Create the single publication request variable
PublicationRequest PubRequestVariable;
PubRequestVariable = new PublicationRequest();

//Set the aquired securitytoken
PubRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
PubRequestVariable.DataSet = dataset;
//Set the datatype
PubRequestVariable.DataType = dataType;
//Set the Authority
PubRequestVariable.Authority = "US";
//Set the document number
PubRequestVariable.Number = "6000000";
//Set the kind code
PubRequestVariable.Kind = "A";

//Set the publication section
PublicationSettings pubSettings = new PublicationSettings();
```

```
pubSettings.PublicationSection = PublicationSection.Family | PublicationSection.Title;

//Set the title language to English
LanguageSettings langSettings = new LanguageSettings();

langSettings.TitleLanguage = Language.English;

pubSettings.LanguageSettings = langSettings;
PubRequestVariable.PublicationSettings = pubSettings;

byte[] publication = client.RetrievePublication(PubRequestVariable);
```

JAVA

```
//Create the single publication request variable
PublicationRequest pubRequestVariable;
pubRequestVariable = new PublicationRequest();

//Set the aquired securitytoken
pubRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
pubRequestVariable.setDataSet(dataSet);
//Set the datatype
pubRequestVariable.setDataType(dataType);
//Set the authority
pubRequestVariable.setAuthority("US");
//Set the document number
pubRequestVariable.setNumber("6000000");
//Set the kind code
pubRequestVariable.setKind("A");

//Set the publication section
PublicationSettings pubSettings = new PublicationSettings();

pubSettings.getPublicationSection().add("Family");
pubSettings.getPublicationSection().add("Title");

byte[] publication = client.retrievePublication(pubRequestVariable);
```

### *Logoff*

Logoff from IPDD by calling the *LogOff* request.

C#

```
using (ServiceClient logoffClient = new ServiceClient("basicHttp"))
{
      //Logoff from IPDD
      logoffClient.LogOff(logonInfo.SecurityToken);
}
```

JAVA

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

### 4.1.6 Scenario 6: Retrieval based on Publication numbers

Who would use this approach?

This approach to IPDD data retrieval would typically be used by a customer who needs to load full authorities based on a list of publication numbers already available. Examples are Patent Information Providers and Corporate Customers with an internal database[9].

Why would you use this approach?

A customer can retrieve a specific list of data based on publication numbers. The advantage is that this list could be generated from a previously internal search on a smaller dataset. After this search a customer could retrieve the full set based on this search result which avoids the need of having a large storage of all the data internally.

Description of Scenario

This scenario uses a list of publication numbers and asks the IPDD service to get the data based on this list. If this data has been processed the result will be retrieved by the customer.

[9] Corporate Customers would use a flavour of IPDD, called IPDD Corporate. IPDD Corporate allows corporate users to retrieve subsets of collections using custom filters

## Process Flow

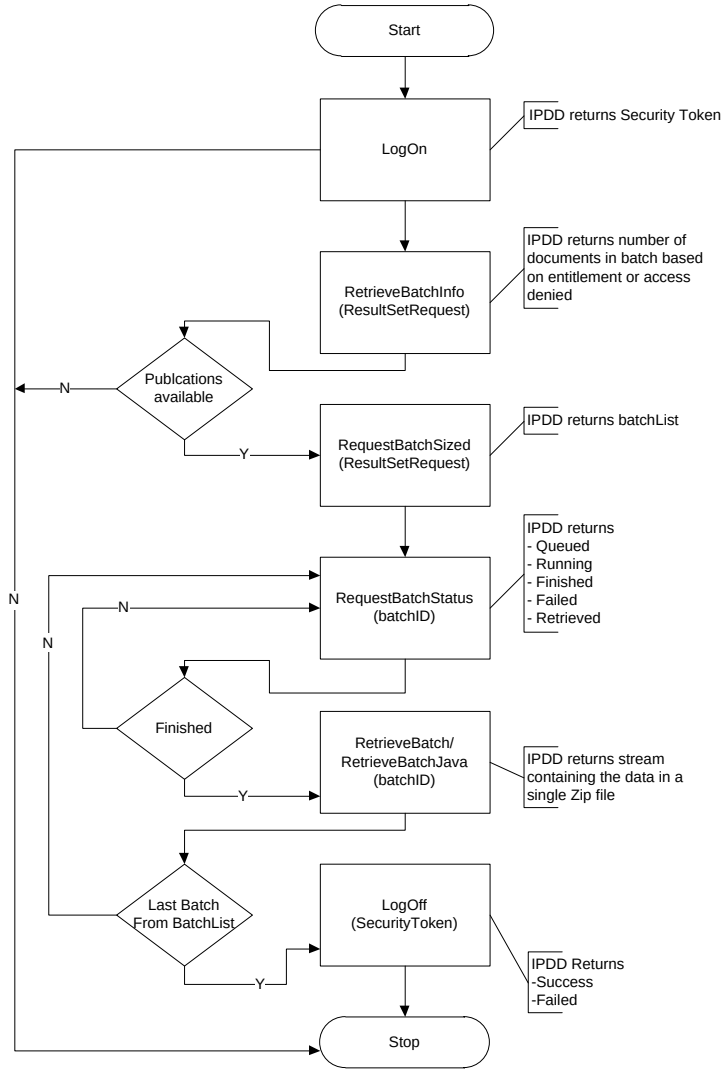



Figure 10: IPDD “Best Practice” Scenario 6 Process Flow

Technical details

### *Logon*

The first step is to logon to the service and acquire a valid security token, this token is valid for 12 hours after which a new token need to be requested.

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = port.logOn(id, password);
```

### *Generate the ResultSetRequest variable*

The ResultSetRequest variable is set to get data from the provided dataset based on the provided publication numbers.

C#

```
//Create the result set request variable to retrieve a set of publications
ResultSetRequest SetRequestVariable;
SetRequestVariable = new ResultSetRequest();

//Set the acquired securitytoken
SetRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
SetRequestVariable.DataSet = dataset;
//Set the datatype
SetRequestVariable.DataType = dataType;

//Set the publication list variable
List<Publication> publications = new List<Publication>();
Publication publication;

//create a list of 100 US publications
for (int i = 6000000; i <= 600100; i++)
{
       publication = new Publication();
```

```
        publication.Authority = "US";
        publication.Number = i.ToString();
        publication.Kind = "A";

        publications.Add(publication);
}

//Add the publications list to the SetRequest variable
SetRequestVariable.Publications = publications.ToArray();
```

Java

```
//Create the result set request variable to retrieve a set of publications
ResultSetRequest SetRequestVariable;
SetRequestVariable = new ResultSetRequest();

//Set the aquired securitytoken
SetRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
SetRequestVariable.setDataSet(dataSet);
//Set the datatype
SetRequestVariable.setDataType(dataType);

//Set the publication list variable
ArrayOfPublicationType publications = new ArrayOfPublicationType();
Publication publication;

//create a list of 100 US publications
for (int i = 6000000; i <= 6000100; i++)
{
        publication = new Publication();

        publication.setAuthority("US");
        publication.setNumber(String.valueOf(i));
        publication.setKind("A");

        publications.getPublicationType().add(publication);
}

//Add the publications list to the SetRequest variable
SetRequestVariable.setPublications(publications);
```

### *Check if publications are available*

The *RetrieveBatchInfo* call is used to get a count of available publications. Based on this count the client can check if there is data available and how much. The request variable used in this call is the previously set *ResultSetRequest*.

C#

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.RetrieveBatchInfo(SetRequestVariable);
```

*Java*

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.retrieveBatchInfo(SetRequestVariable);
```

### *Request the publications*

The publications are requested by using the *RequestBatchSized* call, it is advised to use the sized call since this call supports batches higher than 50.000 publications. This request will return a list of batches that can be retrieved when they are finished.

C#

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches;
listOfBatches = client.RequestBatchSized(SetRequestVariable, 20000);
```

Java

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches = new BatchList();
listOfBatches = client.requestBatchSized(SetRequestVariable, new Long(20000));
```

### *Check for the batch status*

The status will be checked for the first batch in the list, since this is the first one to be processed by IPDD. In this scenario the client checks every 5 minutes for the batch status. If the status changes to *finished* the batch will be retrieved.

C#

```
BatchStatus status;

//Refresh the client to avoid connection problems
using (ServiceClient statusClient = new ServiceClient("basicHttp"))
{
       //check for the batchstatus once every 5 minutes and proceed when the batch is
       finished
       status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
       batch.BatchId);
}


while (status.Status != Status.Finished)
{
      //Check if the status is failed
      if (status.Status == Status.Failed)
             throw new Exception("Batch Failed.");

      //Wait for 5 minutes
      System.Threading.Thread.Sleep(300000);

      //Refresh the client to avoid connection problems
      using (ServiceClient statusClient = new ServiceClient("basicHttp"))
      {
             //Request the batch status
              status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken,
              batch.BatchId);
      }
}
```

Java

```
BatchStatus status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());


while (status.getStatus() != Status.FINISHED)
{
        //Check if the status is failed
        if (status.getStatus() == Status.FAILED)
                throw new Exception("Batch Failed.");


        Thread.sleep(300000);
```

```
        status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());
}
```

### *Retrieve the batch*

The batch will be retrieved as a data stream, this way the data is retrieved in small parts to avoid a high memory load when retrieving large files. The data is retrieved with the help of the position parameter. The use of this parameter enables the client to restart a broken stream in case of a connection problem.

C#

```
//Generate a batch filestream
file = new FileStream(string.Format("{0}data-{1}.zip", batchFolder, batchCount),
FileMode.Create);

//Allow 1000 retries per batch retrieval when a connection breaks
while (retries < 1000 && retry)
{
        //Retrieve Batch
        try
        {
        //Use a new client connection to ensure a stable connection
        using (ServiceClient downloadClient = new ServiceClient("basicHttp"))
        {
                Stream data = downloadClient.RetrieveBatch(logonInfo.SecurityToken,
                batch.BatchId, position);

                while ((i = data.Read(buffer, 0, 4096)) != 0)
                {
                        downloaded += i;
                        file.Write(buffer, 0, i);

                        //Store the current stream/byte position
                        tempPosition = data.Position;
                }

                //Batch retrieved, set retry to false and close the stream
                data.Close();
                retry = false;
        }
        }
        //Catch IOException which happens when the download stream breaks
        catch (IOException)
        {
                //Add the current byte position to the already retrieved ones
                position += tempPosition;
                retries++;
        }
}
```

Java

```
//Generate a batch filestream
FileOutputStream file = new FileOutputStream(String.format("{0}data-{1}.zip", batchFolder,
batchCount));

//Allow 1000 retries per batch retrieval when a connection breaks
while (retries < 1000 && retry)
	{
	//Retrieve batch
	try
	{
		DataHandler result = client.retrieveBatchJava(logonInfo.getSecurityToken(),
		batch.getBatchId(), new Long(0));

		InputStream data = result.getInputStream();

		while ((i = data.read(buffer, 0, 4096)) > 0)
		{
			file.write(buffer, 0, i);

			tempPosition += i;
		}

		data.close();

		retry = false;
	}
	catch(IOException ex)
	{
		position += tempPosition;
		retries++;
	}
}
```

### *Proceed to the next batch*

Check the next batch in the list to see if the status is finished. If there are no more batches in the list proceed to logging off from IPDD.

### *Logoff*

Logoff from IPDD by calling the *LogOff* request.

C#

```
using (ServiceClient logoffClient = new ServiceClient("basicHttp"))
{
      //Logoff from IPDD
      logoffClient.LogOff(logonInfo.SecurityToken);
}
```

JAVA

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

### 4.1.7 Operations Planning

Since IPDD gets its data from different sources containing different and sometimes overlapping data elements it is necessary to design workflow operations in line with business requirements but also the nature of the data delivery. LexisNexis Univentio supports customers in their workflow design process and provides relevant documentation per collection to indicate which data elements arrive in which order such that customers can take the required design decision. The following paragraphs will provide a description on how to handle specific issues when planning for the implementation of IPDD.

Availability of data elements

The data sources that make up an authority differ per authority and may overlap in which case ‘source priorities’ determine the order in which the sources are added to the IPDD database or not. Data sources may refer to either data by issuing authorities and/or data elements manufactured by LexisNexis Univentio.
Consequently each data source could either add new elements or update an existing data element. LexisNexis provides so-called ‘delivery overviews’ per authority which detail the manner in which the major data categories are updated and / or completed over time (see Figure 10, following page).
Based on this information customers can design their workflow and loading schedule to match their specific requirements and constraints. Figure 10 is an example of such a delivery overview which is provided to customers upon request.

LexisNexis

## United States of America - Data Elements and their availability online

### Summary

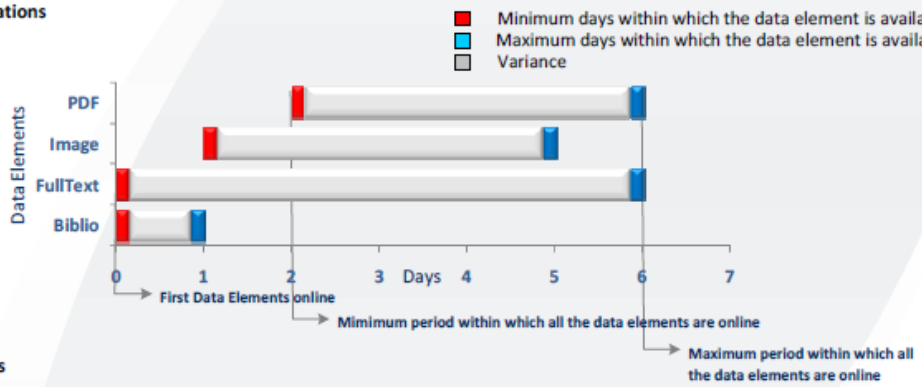
This document summarizes the order/ sequence in which the primary data elements of united states are processed and when they are available online. LexisNexis performed a detailed analysis on the united states past data and summarizes the following,

- For Application kind - the bibliographic information and full text is seen online first followed by Images and PDFs
- For Grant kind - the bibliographic information and full text is seen online first followed by Images and PDFs

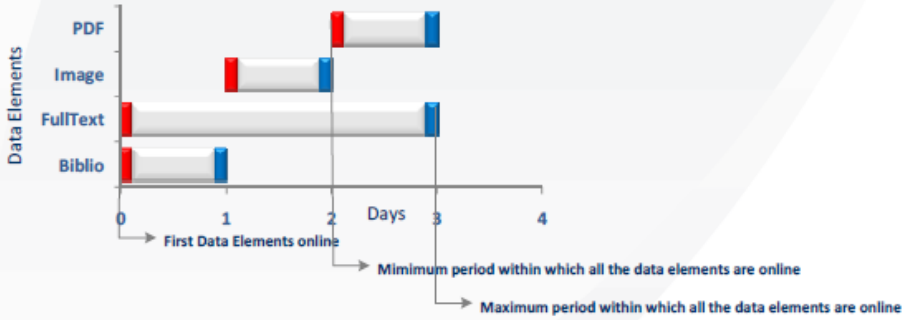
### Data Elements and their availability online

The below charts show for applications and grants the minimum and maximum number of days required for a data element to be online from the publication date. For example, In case of US Applications the PDFs takes a minimum of 2 days and maximum 6 days to be online from the publication date. The variances include the arrival delay of the sources.

**Applications**




**Grants**




| Data Element | Application | | Grant | | Arrival Frequency at LexisNexis | Day of Arrival at LexisNexis |
|---|---|---|---|---|---|---|
| | Min Days | Max Days | Min Days | Max Days | | |
| Biblio | 0 | 1 | 0 | 1 | Twice a week | Tuesday/ Thursday |
| Full Text | 0 | 6 | 0 | 3 | Twice a week | Tuesday/ Thursday |
| Image | 1 | 5 | 1 | 2 | Twice a week | Tuesday/ Thursday |
| PDF | 2 | 6 | 2 | 3 | Twice a week | Tuesday/ Thursday |

Note:

1. Biblio contains meta information including Abstracts
2. Full text means claims and description in this context
3. IPDDv1 will have an extra one day delay

Version 1.0 8/13/2009

Figure 10: US Delivery Overview

Based on this overview a customer could decide to get all the US Applications Bibliographic XML data on the day of publication and subsequently wait up to 3 days to get the updated US publications containing the full text data. In situations where a customer only wants to load records that contain both US bibliographic information **and** full text data in his database, it would be advisable to wait at least 3 days after the publication date to ensure that all elements have been delivered or alternatively perform inventory calls or post-processing to ensure that data is only loaded once the key criteria are met. The Delivery Overviews, however, provide the underlying information required to assist the workflow and operations design process.

Data load

Each authority could generate a high number of updates in addition to newly added data. If an authority for example has an average update count of 50,000 publications per week the customer should ensure that his workflow processes can handle this volume of updates. If this is not the case and volume needs to be reduced, customers have the option of a) separating the addition of new publications from updates and b) lower the amount of update calls issued. This will lead to less timely data but lower the overall number of updates.

Update frequency

The vast majority of authorities publish on a weekly or bi-weekly basis. A customer could choose to get this data at the same frequency if timeliness is a key design criteria or alternatively execute a monthly retrieval in the case that the collection does not need to be kept up to date within a few days or hours of a change occurring. However, it is advisable to perform regular and consistent updates.

### 4.1.8 Technical Considerations

IPDD is a highly versatile and flexible platform. This has its advantages but also makes it potentially challenging for the novice to develop a workflow solution that successfully interacts with IPDD. This chapter has been added with the intent of listing some “do’s” and “don’ts” which are important for the developer when developing against IPDD.

Refresh token within 12 hours

A security token retrieved from the *LogOn* call is only valid for 12 hours. The *SecurityInformation* return value contains a field called Expiration. This field is the exact time after which a token is not valid anymore. A client can use this value to check if a token is still valid and if not, request a new one.

Refresh service connection

The service client object keeps the underlying connection alive and is part of a connection pool on the server side. When issuing a request that potentially takes a long time to process, it is advisable to refresh the service client object to get a new connection. This way connection errors are reduced since an older connection might drop out of the pool if not begin used for a while.

Use the UTC time zone

When using date objects in the request variables it is recommended to set the time zone to UTC. This way the date used matches the actual date used by the service.

Use the BatchInformation call to check for new data

With the help of the *BatchInformation* call a client can check if there new data has become available. Should, for example, new data have become available this could be determined by comparing the latest insert data with the date of the last IPDD data retrieval.

## 4.2 Gateway scenarios

The IPDD gateway has been designed to allow the retrieval of single publications via an URL based interface. A client can easily retrieve a PDF or XML document or a Clipped Image on the fly without having to do a service call to IPDD. The following scenarios provide examples of how the gateway can be used.

### 4.2.1 Scenario 1: Retrieve a single PDF

The US publication 7100000 with kind code B1 is retrieved via the following URL:

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=us,7100000,b1

After the POST the PDF data is downloaded and can be stored on disk or opened in a PDF reader.

### 4.2.2 Scenario 2: Retrieve all images publications from one number

All off the images from US publication 7100000 with kind code B1 are retrieved via the following URL:

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<passord>&images=us,6000000,a

After the POST the Zip file containing all of the images is downloaded and can be stored on disk or opened in a Zip client.

# Technical description service

This chapter provides a technical description of all the requests available in IPDD. Each request will be described along with code snippets in C# and JAVA. The first paragraphs will provide an explanation on how IPDD handles faults and how request parameters are being created. After that each request will be described.

## 5.1 Fault Messages

Each request will return a fault message if a problem occurs during the processing of the request. IPDD uses custom fault messages to indicate the nature of the fault and provide the client with the ability to handle the fault correctly. These so called SOAP faults can be handled when using the SOAP service. If REST is being used the fault message will also have a corresponding HTTP status code which can be used to handle the fault.

The following table shows the fault messages generated by IPDD along with the corresponding HTTP status code.

| Name | Description | Code |
|---|---|---|
| InvalidLogOn | Unknown Login provided. | 401 |
| InvalidPassword | Unknown Password provided. | 401 |
| InvalidSecurityToken | Invalid or expired security token provided. | 401 |
| InvalidDataSet | The provided security token does not allow access to the requested dataset. | 406 |
| InvalidRequestType | The provided request type is not allowed. | 410 |
| InvalidBatchId | Invalid or expired batch id provided. | 409 |
| BatchRunning | The requested batch is still being processed. | 404 |
| BatchFailed | The requested batch has failed. | 410 |
| DataUnavailable | The requested data is unavailable. | 404 |
| ImageDataUnavailable | The requested image data for this publication is not available. | 404 |
| NotAllowedRequest | The provided security token does not allow usage of this request. | 405 |

| | | |
|---|---|---|
| NotAllowedData | The provided security token does not allow access to the requested data. | 403 |
| ServiceError | General service error, please contact customer support. | 400 |
| InvalidDateRange | Invalid date range, 1 to 31 days allowed | 412 |
| LargeBatch | The requested batch has too many publications to be processed, please use the sized batch request. | 413 |
| InvalidBatchSize | The provided batchsize is either too large or too small, please refer to the technical documentation for the allowed range. | 414 |
| InvalidSearch | The provided search terms are either invalid or produce too many results. | 417 |

## 5.2 Request variables

IPDD requests have different variables that a client can use to request data. Some requests use a custom IPDD variable named *RequestType*. This is a base variable type from which several other specialized request variables derive. Each of these variables can be used to do a specific request of data, like for example the *PubDateRequest* variable which is used to get all publications from a certain publication date. Each request described in the next paragraphs will have a list of which request types can be used.

The following table provides an overview of all the request types available and the associated property fields available per request type. An explanation of the IPDD specific fields and return types can be found at IPDD custom return and field types.

| **Name** | **Fields** | **Description** |
|---|---|---|
| PubDateRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup<br>PublicationDate<br>PublicationSettings | Request a batch containing all of the publications based on the provided publication date. |
| PubDateRangeRequest | SecurityToken<br>DataSet | Request a batch containing all of the publications based on the provided |

| | | |
|---|---|---|
| | DataType<br>KindGroup<br>PublicationDateFrom<br>PublicationDateToo<br>PublicationSettings | publication date range. |
| NewRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup | Request a batch containing all of the newly added publications since the last new request. |
| AddedOnDateRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup<br>AddedOnDate<br>PublicationSettings | Request a batch containing all of the publications, which have been added on the provided added on date. |
| AddedOnDateRangeRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup<br>AddedOnDateFrom<br>AddedOnDateToo<br>PublicationSettings | Request a batch containing all of the added publications based on the provided added on date range. |
| UpdateRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup | Request a batch containing all of the updated and new publications since the last update and new request. |
| ChangeDateRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup<br>ChangeDate<br>PublicationSettings | Request a batch containing all of the publication changed on the provided date. |
| ChangeDateRangeRequest | SecurityToken<br>DataSet<br>DataType<br>KindGroup<br>ChangeDateFrom<br>ChangeDateToo | Request a batch containing all off the changed publications based on the provided changed date range. |

| | | |
|---|---|---|
| | PublicationSettings | |
| PublicationRequest | SecurityToken<br>Dataset<br>DataType<br>Authority<br>Number<br>Kind<br>PublicationSettings | Request a single publication in Xml format or the PDF /Clip based on the publication number information and data type. |
| ResultSetRequest | SecurityToken<br>DataSet<br>DataType<br>Publications<br>PublicationSettings | Request a dataset based on the publication information provided in the publications field. This request type can be used for requesting a result set but also for retrieving a specific dataset based on a publication list. |
| HistoryRequest | SecurityToken<br>DataSet<br>DataType<br>RequestDateFrom<br>RequestDateTo<br>MaxCount<br>Status | Request a history list in XML format of previously requested batches.The result can be filtered by using the date properties from and to and by settings the status property. |
| SearchRequest | SecurityToken<br>DataSet<br>DataType<br>SearchTerms<br>Authorities<br>OutputTemplate<br>MaxRows | Request a resultset in XML format based on the searchterms. The resultset output can be set by using the outputtemplate property. |
| BatchSearchRequest | SecurityToken<br>DataSet<br>DataType<br>SearchTerms<br>Authorities | Request a batch containing data based on the provided search terms. |

## 5.3 Logon

The Login and Password are provided through this request, which returns a *SecurityInformation* type containing a unique token. This token will be used to access other methods and will be valid for 12 hours or until the Logoff request is being used.

| Input | Output | Faults |
|---|---|---|
| Login (string)<br>Password (string) | Security Information (IPDD Type) | InvalidLogin,<br>InvalidPassword,<br>ServiceException |

C#

```
//Logon
SecurityInformation logonInfo;
logonInfo = client.LogOn(id, passWord);
```

Java

```
// Logon
SecurityInformation logonInfo = port.logOn(id, password);
```

## 5.4 LogOff

The logoff request invalidates the provided security token, which makes the token invalid to use for further requests.

| Input | Output | Faults |
|---|---|---|
| Security Token (guid) | True/False (boolean) | InvalidSecurityToken,<br>ServiceException |

C#

```
//Logoff from IPDD
logoffClient.LogOff(logonInfo.SecurityToken);
```

Java

```
//Logoff from IPDD
client.logOff(logonInfo.getSecurityToken());
```

## 5.5 RetrieveDataSetStatus

The *RetrieveDataSetStatus* request is used to retrieve date information related to the dataset. This way a client can check the latest change date on a dataset or the last time a client retrieved new publication from that dataset.

| Input | Output | Faults |
|---|---|---|
| Security Token (guid)<br>DataSet (IPDD Type) | DataSet Status (IPDD Type) | InvalidSecurityToken,<br>InvalidDataSet,<br>ServiceException |

C#

```
DataSetStatus status = client.RetrieveDataSetStatus(logonInfo.SecurityToken, dataset);
```

Java

```
DataSetStatus status = client.retrieveDataSetStatus(logonInfo.getSecurityToken(), dataset);
```

## 5.6 RetrieveBatchInfo

The *RetrieveBatchInfo* request is being used to retrieve information about a batch a client wants to request based on the provided request type. The return type contains a count of documents that will be available in the batch and can be extended with a list of publications when specified in the request type. When using the *PublicationRequest* type a client can get a list of all publication stages from a single publication by using an asterix as kind code. Based on this information a client can decide to actually request the data or to adjust the request to get more/less data and/or postpone the retrieval. Note that the maximum allowed list size is 100,000 publications, see the *RetrieveBatchInfoLarge* call is a larger set is required.

| Input | Output | Faults |
|---|---|---|
| Request Type (IPDD Type) | Batch Information (IPDD Type) | InvalidSecurityToken,<br>InvalidDataSet,<br>InvalidRequestType,<br>NotAllowedRequest,<br>NotAllowedData,<br>InvalidDateRange,<br>ServiceException |

C#

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.RetrieveBatchInfo(updateRequestVariable);
```

Java

```
//Retrieve batch info
BatchInformation batchInfo;
batchInfo = client.retrieveBatchInfo(updateRequestVariable);
```

## 5.7 RetrieveBatchInfoLarge

The *RetrieveBatchInfoLarge* call is used to retrieve a result set of publications based on the provided request type. This call is an extension on *RetrieveBatchInfo* with the difference that this call returns a stream containing a zip file. This zip file contains the publications divided in xml files with a maximum of 50,000 per file. The maximum allowed count of publications is 1,000,000.

| Input | Output | Faults |
|---|---|---|
| Request Type (IPDD Type) | Zipfile (Stream) | InvalidSecurityToken,<br>InvalidDataSet,<br>InvalidRequestType,<br>NotAllowedRequest,<br>NotAllowedData,<br>LargeBatch,<br>ServiceException |

C#

```
Stream data = downloadClient.RetrieveBatchInfoLarge(updateRequestVariable);
while ((i = data.Read(buffer, 0, 4096)) != 0)
{
     file.Write(buffer, 0, i);
}

data.Close();
```

Java

```
DataHandler result = client.retrieveBatchJava(logonInfo.getSecurityToken(), batch.getBatchId(), new
Long(0));
InputStream data = result.getInputStream();
while ((i = data.read(buffer, 0, 4096)) > 0)
{
          file.write(buffer, 0, i);
}
data.close();
```

## 5.8 RequestBatch

The RequestBatch request requests the IPDD service to start generating a batch containing the data specified in the provided request type. The request returns a batch id, which will be used to download the batch after it has been processed or to retrieve the current status of the batch. Note that this request has a maximum of 50,000 publications, the RequestBatchSized call is required to get larger publication sets.

| Input | Output | Faults |
|---|---|---|
| Request Type (IPDD Type) | Batch Id (guid) | InvalidSecurityToken,<br>InvalidDataSet,<br>InvalidRequestType,<br>DataUnavailable,<br>NotAllowedRequest,<br>NotAllowedData,<br>InvalidDateRange,<br>LargeBatch,<br>ServiceException |

C#

```
//Request batch below 50,000 publications
Guid batchId = client.RequestBatch(updateRequestVariable);
```

Java

```
//Request batch below 50,000 publications
String batchId = client.requestBatch(updateRequestVariable);
```

## 5.9 RequestBatchSized

The RequestBatchSized request requests the IPDD service to start generating a batch. The batch will be divided into smaller batches if the publication count is larger than the provided batch size. A batch size between 20000 and 50000 publications can be specified. The request returns a list with batch ids and count which can be used to get the status of each batch and download when finished.

| Input | Output | Faults |
|---|---|---|
| Request Type (IPDD Type)<br>BatchSize (20000 – 50000) | Batch List (IPDD Type) | InvalidSecurityToken,<br>InvalidDataSet,<br>InvalidRequestType,<br>DataUnavailable,<br>NotAllowedRequest,<br>NotAllowedData,<br>InvalidDateRange,<br>ServiceException |

C#

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches = new BatchList();

listOfBatches = client.RequestBatchSized(updateRequestVariable, 20000);
```

Java

```
//RequestBatch with a maximum of 20000 publications per batch
BatchList listOfBatches = new BatchList();
listOfBatches = client.requestBatchSized(updateRequestVariable, new Long(20000));
```

## 5.10 RetrieveBatchStatus

The *RetrieveBatchStatus* request is used to retrieve the status of a batch. The request return type contains a field status that is either running, failed or finished. Based on this status a client can proceed to retrieve the batch in the case that the process hand been completed, hold (running) or perform additional action (failed). When a batch has been processed, the field size is populated with batch size in bytes. This way a client knows the size of the batch file.

| Input | Output | Faults |
|---|---|---|
| Security Token (guid)<br>Batch Id (guid) | Batch Status (IPDD Type) | InvalidSecurityToken,<br>InvalidBatchId,<br>ServiceException |

C#

```
//Request the batch status
status = statusClient.RetrieveBatchStatus(logonInfo.SecurityToken, batch.BatchId);
```

Java

```
BatchStatus status = client.retrieveBatchStatus(logonInfo.getSecurityToken(), batch.getBatchId());
```

## 5.11 RetrieveBatch/RetrieveBatchJava

The *RetrieveBatch* request returns a data stream containing a zip file with the publications belonging to the provided batch id. The *RetrieveBatchJava* call does exactly the same, but is made especially for Java since the default call does not create a datahandler in Java because of an interoperability problem between Java and WCF. If the stream breaks while retrieving the data the field position can be used to pick up the stream from the last processed position.

| Input | Output | Faults |
|---|---|---|
| Security Token (guid)<br>Batch Id (guid)<br>Position (long) | Zipfile (stream) | InvalidSecurityToken,<br>InvalidBatchId,<br>BatchRunning,<br>BatchFailed,<br>ServiceException |

C#

```
//Retrieve the batch by using a stream
Stream data = downloadClient.RetrieveBatch(logonInfo.SecurityToken, batch.BatchId, 0);

while ((i = data.Read(buffer, 0, 4096)) != 0)
{
      file.Write(buffer, 0, i);
}

data.Close();
```

Java

```
DataHandler result = client.retrieveBatchJava(logonInfo.getSecurityToken(), batch.getBatchId(), new
Long(0));
InputStream data = result.getInputStream();
while ((i = data.read(buffer, 0, 4096)) > 0)
{
         file.write(buffer, 0, i);
}
data.close();
```

## 5.12 RetrievePublication

The *RetrievePublication* request is used to get a single publication based on the provided *PublicationRequest* parameter. An XML request will provide an XML file, a PDF request will return a PDF file, a Clip request will return a PNG file, and a Images request will provide a zip file with PNG images.

| Input | Output | Faults |
|---|---|---|
| PublicationRequest (IPDD Type) | Byte array | InvalidSecurityToken,<br>InvalidDataSet,<br>InvalidRequestType,<br>DataUnavailable,<br>ImageDataUnavailable,<br>NotAllowedRequest,<br>NotAllowedData,<br>ServiceException |

C#

```
byte[] publication = client.RetrievePublication(PublicationRequestVariable);
```

Java

```
byte[] publication = client.retrievePublication(PublicationRequestVariable);
```

## 5.13 ResultSetRequest

The *ResultSetRequest* is used to obtain a set of publications based on the provided *ResultSetRequest* parameter. The request returns an XML document. This set does not contain the full publication but only a result set of publications which can be used in for example an overview of publications available.

| Input | Output | Faults |
|---|---|---|
| ResultSetRequest (IPDD Type) | Byte array | InvalidSecurityToken,<br>InvalidDataSet,<br>DataUnavailable,<br>NotAllowedRequest,<br>NotAllowedData,<br>ServiceException |

C#

```
//Create the result set request variable to retreive a set of publications
ResultSetRequest SetRequestVariable;
SetRequestVariable = new ResultSetRequest();

//Set the acquired securitytoken
SetRequestVariable.SecurityToken = logonInfo.SecurityToken;
//Set the dataset
SetRequestVariable.DataSet = dataset;
//Set the datatype
SetRequestVariable.DataType = dataType;

//Set the publication list variable
List<Publication> publications = new List<Publication>();
Publication publication;

//create a list of 100 US publications
for (int i = 6000001; i <= 600100; i++)
{
publication = new Publication();

publication.Authority = "US";
publication.Number = i.ToString();
publication.Kind = "A";

publications.Add(publication);
}

//Add the publications list to the SetRequest variable
SetRequestVariable.Publications = publications.ToArray();

//Request the result set
```

```
byte[] resultSet = client.RequestResultSet(SetRequestVariable);
```

Java

```
//Create the result set request variable to retrieve a set of publications
ResultSetRequest SetRequestVariable;
SetRequestVariable = new ResultSetRequest();

//Set the aquired securitytoken
SetRequestVariable.setSecurityToken(logonInfo.getSecurityToken());
//Set the dataset
SetRequestVariable.setDataSet(dataSet);
//Set the datatype
SetRequestVariable.setDataType(dataType);

//Set the publication list variable
ArrayOfPublicationType publications = new ArrayOfPublicationType();
Publication publication;

//create a list of 100 US publications
for (int i = 6000001; i <= 6000100; i++)
{
publication = new Publication();

publication.setAuthority("US");
publication.setNumber(String.valueOf(i));
publication.setKind("A");

publications.getPublicationType().add(publication);
}

//Add the publications list to the SetRequest variable
SetRequestVariable.setPublications(publications);

//Request the result set
byte[] resultSet = client.requestResultSet(SetRequestVariable);
```

## 5.14 HistoryRequest

The *HistoryRequest* is used to provide an overview of previously requested batches based on the provided parameters. This information can be used to keep track of requested data and can also be used to retrieve batches requested in the past based on the provided batch id. The result is a byte array containing an XML document. The maximum number of allowed batches in a list is 200.

| Input | Output | Faults |
|---|---|---|
| HistoryRequest (IPDD Type) | Byte array | InvalidSecurityToken,<br>DataUnavailable,<br>LargeBatch,<br>ServiceException |

C#

```
//Create the historyrequest variable
HistoryRequest historyRequest = new HistoryRequest();

//Set the acquired securityToken
historyRequest.SecurityToken = logonInfo.SecurityToken;

//Set the dataset
historyRequest.DataSet = DataSet.US;

//Set the datatype
historyRequest.DataType = DataType.Xml;

//Set the from date
historyRequest.RequestDateFrom = new DateTime(2010, 1, 1);

//Set the to date
historyRequest.RequestDateTo = new DateTime(2010, 2, 1);

//Set the statues to retrieved
historyRequest.Status = Status.Retrieved;

//Set the maximum allowed result
historyRequest.MaxCount = 100;

using (ServiceClient historyClient = new ServiceClient("basicHttp"))
{
       byte[] historyList = historyClient.RequestHistory(historyRequest);
}
```

JAVA

```
//Create the historyrequest variable
HistoryRequest historyRequest = new HistoryRequest();

//Set the acquired securityToken
historyRequest.setSecurityToken(logonInfo.getSecurityToken());

//Set the dataset
historyRequest.setDataSet(dataSet.US);

//Set the datatype
historyRequest.setDataType(dataType.XML);

//Set the from date
historyRequest.setRequestDateFrom(date);

//Set the to date
historyRequest.setRequestDateTo(date);

//Set the statues to retrieved
historyRequest.setStatus(Scenarios.Status.RETRIEVED);

//Set the maximum allowed result
historyRequest.setMaxCount(100);

byte[] historyList = client.RequestHistory(historyRequest);
```

## 5.15 RequestSearch

The RequestSearch is used to perform a search on one or multiple datasets, the result of this search is an Xml file containing the result set of the performed search. This result set also contains search statistics on how many documents have been searched.

| Input | Output | Faults |
|---|---|---|
| SearchRequest(IPDD Type) | Byte array | InvalidSecurityToken, InvalidSearch, DataUnavailable, LargeBatch, InvalidDataSet, NotAllowedRequest, ServiceException |

C#

```
//Create the searchRequest variable
SearchRequest searchRequest = new SearchRequest();

//Set the acquired securityToken
searchRequest.SecurityToken = logonInfo.SecurityToken;

//Add the US authority
searchRequest.Authorities = new Collection();
searchRequest.Authorities.Add("US");

//Set the search terms
searchRequest.SearchTerms = “CPC(A61!) AND DATE(>2014-01)”;

//Set the maximum allowed result
searchRequest.MaxRows = 100;

using (ServiceClient searchClient = new ServiceClient("basicHttp"))
{
   byte[] searchResult = searchClient.RequestSearch(searchRequest);
}
```

JAVA

```
//Create the historyrequest variable
SearchRequest searchRequest = new SearchRequest();

//Set the acquired securityToken
searchRequest.setSecurityToken(logonInfo.getSecurityToken());

//Add the US authority
Collection auth = new Collection();
auth.getString().add("US");
searchRequest.setAuthorities(auth);
```

```
//Set the search terms
searchRequest.setSearchTerms("CPC(A61!) AND DATE(>2014-01)");

//Set maximum allowed result
searchRequest.setMaxRows(100);
byte[] searchResult = client.requestSearch(searchRequest);
```

# Technical description gateway

The IPDD Gateway is a separate implementation on top of IPDD that provides access to publication data via a URL interface. The functionality can be divided into two groups, one is the retrieval of a list which contains the publication data based on the provided parameters and the other is the actual retrieval of publication data based on the provided parameters. The following chapters will provide an overview of the current functionality of the IPDD Gateway.

## 6.1 Fault messages

The IPDD gateway returns a message when the request made could not be processed. The following table gives an overview of these messages.

| **Message** | **Explanation** |
|---|---|
| Error code: 21<br>(No valid Authority provided) | The authority parameter has an invalid authority or the client does not have access to this authority. |
| Error code: 23 | Invalid query string or service error. |
| Error code: 24<br>(Invalid Username provided) | Invalid username provided. |
| Error code: 25<br>(PDF file not found) | Image data not found. |
| Error code: 26<br>(Publication not found) | Publication not found. |
| Error code: 28<br>(Invalid date, use 'yyyymmdd') | The provided publication date is not correctly formatted. |
| Error code: 29<br>(Invalid file format, use xml/txt/url) | The provided list file format is not correct. |
| Error code: 30<br>(Zip set to false but more than one patent found) | The zip enable option is set to false but more than one publication was found in the request. If the result is more than one the data needs to be zipped. |
| Error code: 31<br>(Invalid date range, 1 to 31 days allowed) | A date range with more than 31 or less than 1 day was provided. |
| Error code: 32 | An invalid email address was provided. |

| | |
|---|---|
| (Invalid email address provided) | |
| Error code 33:<br>(Invalid Password provided) | Invalid password provided. |
| Error code 34:<br>(Decryption of key failed) | Decryption of the provided key failed. |
| Error code 35:<br>(Invalid key data) | Invalid data inside the encrypted key. |

## 6.2 List retrieval

The list retrieval functionality enables the client to retrieve a list of publication numbers in several formats and based on a single publication number, publication date or publication date range. This list can then be used for the actual publication request or to have an overview of the actual available publications on the IPDD Gateway, currently the list will only provide an overview of the available PDF files on the IPDD gateway.

### 6.2.1 List request options

*Publication number*

A list of all the available publication stages will be provided when the authority and publication number is set.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000

*Publication number and kind*

A list with one publication based on the provided authority, number and kind code.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&kind=A

*Publication Date*

A list containing all of the publications published by the provided authority on the given publication date.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&pubdate=20080101

A list with all of the publication published by the provided authority in the given date range, note that the maximum date range is between 1 and 31 days.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&pubdatestart=20080101&pubdateend=20080115

### 6.2.2 List output options

The list result can be acquired in different formats, where the default is XML. By using the parameter format a client can specify which format to use. Each format will be described along with an example layout.

*Xml*

The Xml format will return an Xml file containing the authority, document number, kind code and publication date.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=xml

List:

```
<?xml version="1.0" encoding="utf-8"?>
<lexisnexis-publicationlist date-produced="20081127" schema-version="1.1" produced-by="LexisNexis-Univentio" lang="eng">
        <document-id>
                <country>US</country>
                <doc-number>6000000</doc-number>
                <kind>A</kind>
                <date>19991207</date>
        </document-id>
</lexisnexis-publicationlist>
```

*Txt*

The Txt format will return a Txt file containing the authority, document number and kind code in a comma separated format.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=txt

List:

*US,6000000,A*

*URL*

The URL format will return a Txt file containing IPDD Gateway publication request URL's which can be used to retrieve a publication.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=url

List:

*http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A*

## 6.3 Publication retrieval

The publication retrieval enables a client to retrieve publication data based on the provided parameters. A client can retrieve a single PDF for a publication or retrieve the images from this publication. The following paragraphs will describe all the possibilities.

### 6.3.1 Single Publication

A default single publication request is formatted like this:

**PDF:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A

Returns: PDF file

**Clipped Image:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&clip=US,6000000,A

Returns: PNG file

**Images:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&images=US,6000000,A

Returns: Zip file

A client provides the login name and password and submits the publication request in the publication parameter. The publication parameter consists of the authority, number and complete kind code separated by commas.

### 6.3.2 Kind code wildcard

If a client replaces the kind code with an asterisk, the IPDD Gateway will return all of the available publication stages belonging to the specified authority and publication number.

A default wildcard publication request is formatted like this:

**PDF:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,*

Returns: PDF file if result is one, zip file if more results are found.

**Clipped Image:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&clip=US,6000000,*

Returns: Returns PNG file if result is one, zip file if more results are found.

**Images:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&images=US,6000000,*

Returns: Zip file

A client provides his login name and password and puts the publication request in the publication parameter. The publication parameter consists of the authority, number and an asterisk for the kind code separated by commas.

### 6.3.3 Kind code semi wildcard

The IPDD gateway will return a subset of available stages when the number from the kind code is being replaced by an asterisk. This way a client can retrieve a publication when he knows the letter of the kind code but not the number.

A default semi wildcard publication request is formatted like this:

**PDF:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A*

Returns: PDF file if result is one, zip file if more results are found.

**Clipped Image:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&clip=US,6000000,A*

Returns: Returns PNG file if result is one, zip file if more results are found.

**Images:**

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&images=US,6000000,A*

Returns: Zip file

A client provides his login name and password and puts the publication request in the publication parameter. The publication parameter consists of the authority, number and an asterisk for the kind code separated by commas.

## 6.4 Advanced options

The IPDD Gateway has several advanced options that can be used in conjunction with the publication retrieval and list retrieval. The options are being activated through extra parameters in the IPDD Gateway request URL. The following options are currently available.

### 6.4.1 List options

*Encrypt request*

A list can be supplemented with an encrypted request which can be used to retrieve a publication via the key parameter (see *Request with encryption*).

The key value will be added to different output formats in the following way.

*XML*

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=xml&generatekey=true

List:

```
<?xml version="1.0" encoding="utf-8"?>
<lexisnexis-publicationlist date-produced="20081127" schema-version="1.1" produced-
by="LexisNexis-Univentio" lang="eng">
        <document-id>
                <country>US</country>
                <doc-number>6000000</doc-number>
                <kind>A</kind>
                <date>19991207</date>
                <key><key></key>
        </document-id>
</lexisnexis-publicationlist>
```

*TXT*

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=txt&generatekey=true

List:

*US,6000000,A,<key>*

*URL*

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&authority=US&number=6000000&format=url&generatekey=true

List:

*http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?key=<key>*

### 6.4.2 Publication options

*Request with encryption*

An encrypted key retrieved through a list request can be used to retrieve a PDF publication with the use of the *key* parameter.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?key=<key>

*Override Zip*

The default zip functionality can be overridden by using the zip Boolean parameter. The default functionality is to return a zip file when multiple publications are being found and return a PDF/PNG file when one publication is being found. The publication data will always be in a single zip file when the zip parameter is set to true.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A&zip=true

*Email Encrypted link*

Via the email parameter an encrypted publication retrieval link can be emailed. The request will be encrypted and emailed to the provided email address.

URL:
http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A&email=john@doe.com

*Add remark*

A remark can be provided when requesting a publication, this remark will be stored in the IPDD database and can be used later on to get an overview based on the provided remark.

URL:

http://ipdatadirect.lexisnexis.com/downloadpdf.aspx?lg=<login>&pw=<password>&pdf=US,6000000,A&remarks=testing123

# Appendix

## 7.1 Contacting LexisNexis IPDD Support

Support for IP DataDirect – Patents can be obtained by contacting LexisNexis IPDD Customer Support as follows:

E: customersupport@univentio.com
T: +31 88 639 00 00

Opening Hours: Mon - Fri; 0800-1700 CE

## 7.2 Glossary

| Name | Description |
|---|---|
| MTOM | MTOM is the W3C Message Transmission Optimization Mechanism, a method of efficiently sending binary data to and from Web services. |
| REST | REST is a term to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer. |
| HTTP | The Hypertext Transfer Protocol (HTTP) is an Application Layer protocol for distributed, collaborative, hypermedia information systems. |
| SOAP | SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. |
| IDE | An integrated development environment (IDE) also known as integrated design environment or integrated debugging environment is a software application that provides comprehensive facilities to computer programmers for software development. |
| PNG | Portable Network Graphics (PNG) is a bitmapped image format that employs lossless data compression. |
| PDF | Portable Document Format (PDF) is a file format created by Adobe Systems in 1993 for document exchange. Adobe PDF is used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system. |
| WCF | The Windows Communication Foundation (or WCF) is an application programming interface in the .NET Framework for building connected, service-oriented applications. |

## 7.3 Endpoints

Several service endpoints are available to provide the best interoperability.

*BasicHttp*

*http://ipdatadirect.lexisnexis.com/service.svc*

The *BasicHttp* endpoint is the default endpoint used to connect to the service. It is SOAP 1.1 enabled and has MTOM (*Message Transmission Optimization Mechanism*) enabled. It also uses a streaming transfer mode meaning that the data is being streamed from the server instead of being buffered first.

*BasicHttpTxt*

*http://ipdatadirect.lexisnexis.com/service.svc/basichttptxt*

The *BasicHttpTxt* endpoint has the same functionality as the *BasicHttp* with the exception that MTOM is not enabled. This endpoint is best being used for SOAP 1.1 interoperability when the client has no support for MTOM.

*Rest*

*http://ipdatadirect.lexisnexis.com/service.svc/rest*

The Rest endpoint is used for posting REST style messages to the service instead of using SOAP. This way a client can access the service directly without implementing a SOAP client. See the chapter Technical description service for some REST examples.

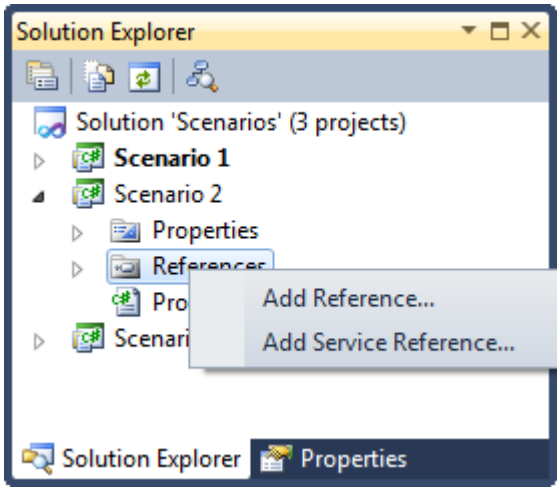
## 7.4 Adding Service Reference Examples

An easy way to access IPDD is to add as a web service reference in an IDE. This reference can then easily be accessed in the code. The following chapters will give an example in Visual Studio 2010 and in NetBeans 6.8.

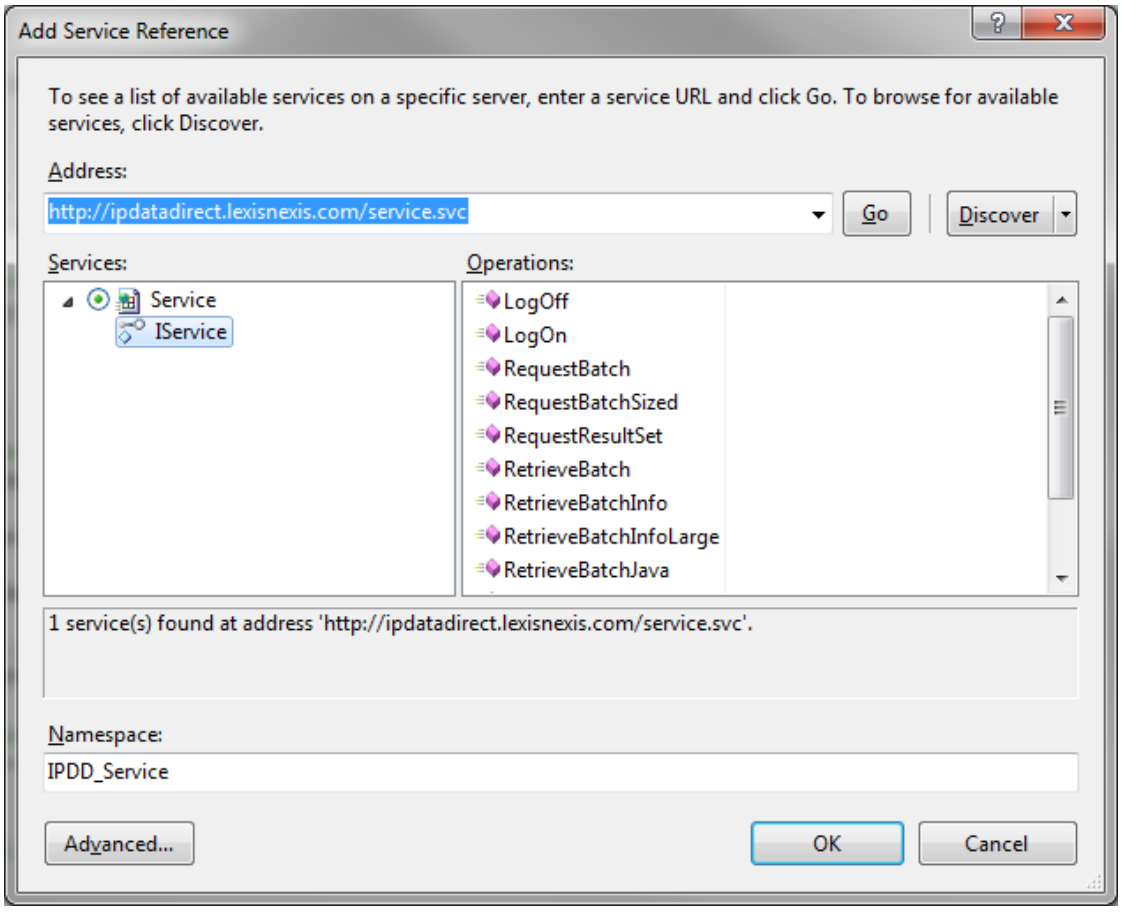
### 7.4.1 Visual Studio 2010

The section will describe how to add IPDD as a reference to Visual Studio 2010.

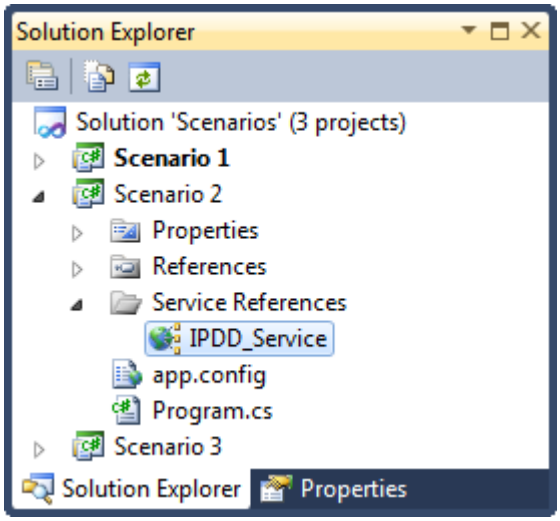
- Launch Visual Studio 2010
- Create or open a project
- Right click on the references folder in the solution explorer and select *Add Service Reference*



- Enter the IPDD service URL in the Address Box en click the Go button set the Namespace as IPDD_Service.



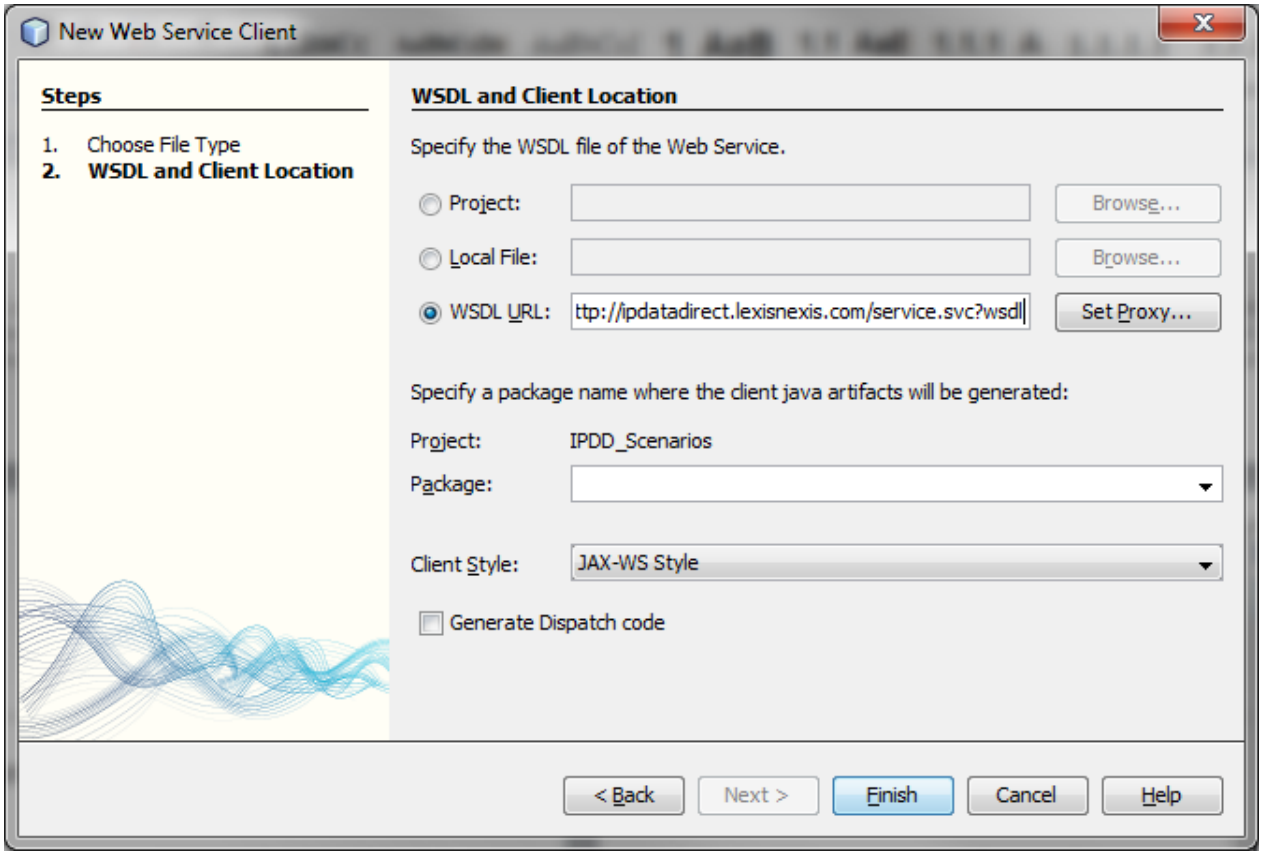

- The service reference is now available in the project under folder *Service References* and accessible in the code from the namespace IPDD_Service

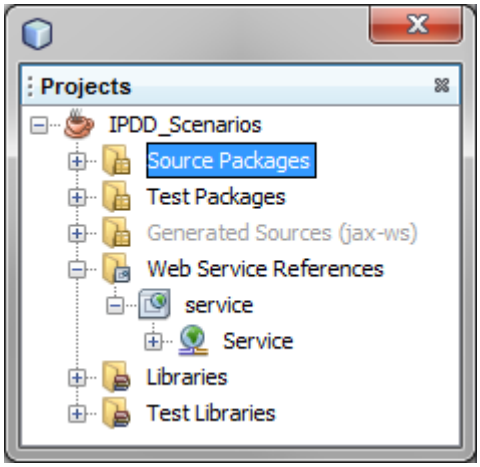


### 7.4.2 NetBeans 6.8

This section will describe how to add IPDD as a reference to NetBeans 6.8.

- Launch Netbeans 6.8
- Create or open a project
- Right click on the Source packages folder in the projects tree and select New > Web Service Client.
- Enter the IPDD service URL in WSDL URL box and click the finish button.






- The service reference is now available in the project under the folder Web Service References and accessible in the code under the namespace IPDD_Service.

## 7.5 IPDD custom return and field types

IPDD uses several custom variables as return or request type variable. The following tables describe each of those variables.

| Name | Type | Description |
|---|---|---|
| DataSet | Enum | Authority code or custom type used to identify the dataset. |
| DataType | Enum | Data format of the requested publication<br>• **Xml**<br>• **Pdf**<br>• **Clip**<br>• **Images** |
| KindGroup | Enum | Group of publications to request<br>• **All**<br>• **Application**<br>• **Grant**<br>• **Other** |
| ListFormat | Enum | Format of the publication list<br>• **Xml**<br>• **Txt**<br>• **Unknown** |
| DataSetStatus | IPDD Object | Collection of date objects giving the status of the provided dataset.<br>• **LatestPublicationDate**<br>• **LatestInsertDate**<br>• **LatestChangeDate**<br>• **RetrievedInsertDate**<br>• **RetrievedChangeDate** |
| BatchInformation | IPDD Object | Provides information about a possible batch based on the provided request type.<br>• **Count**<br>• **List**<br>• **TotalSearched**<br>• **TotalFound** |
| BatchList | IPDD Object | List of Batches generated by the RequestBatchSized call.<br>• **BatchId** *(guid to identify the batch)* |

| | | |
|---|---|---|
| | | • **BatchSize** *(number of publications to get processed)* |
| BatchStatus | IPDD Object | Status of the batch being processed.<br>• **Status** *(current processing status of the batch)*<br>• **Size** *(size of the batch in bytes)*<br>• **Count** *(number of publications in the batch)*<br>• **FilePath** *(FTP Location and filename of the batch)* |
| SecurityInformation | IPDD Object | Information to login to IPDD.<br>• **SecurityToken**<br>• **Expiration** |
| PublicationSettings | IPDD Object | Used with the PublicationRequest type to choose specific elements in a single publication.<br>• **FamilyType**<br>• **PublicationSection**<br>• **LanguageSettings** |
| PublicationSection | Enum | Sections inside a publication which can be selected separately or combined. |
| FamilyType Enum | Enum | Type of family information to be selected.<br>• **Both**<br>• **Main**<br>• **Extended**<br>• **Simple**<br>• **All** |
| LanguageSettings | IPDD Object | Used to select a section in a specified language.<br>• **TitleLanguage**<br>• **AbstractLanguage**<br>• **DescriptionLanguage**<br>• **ClaimsLanguage** |
| Status | Enum | Current status of the batch.<br>• **All** *(all statuses, only to use with the history call to get everything)*<br>• **Finished** *(batch has been finished but not retrieved)*<br>• **Running** *(batch is being processed)*<br>• **Failed** *(batch has failed)*<br>• **Queued** *(batch is on the queue waiting to get processed)*<br>• **Retrieved** *(batch is finished and has been retrieved)* |

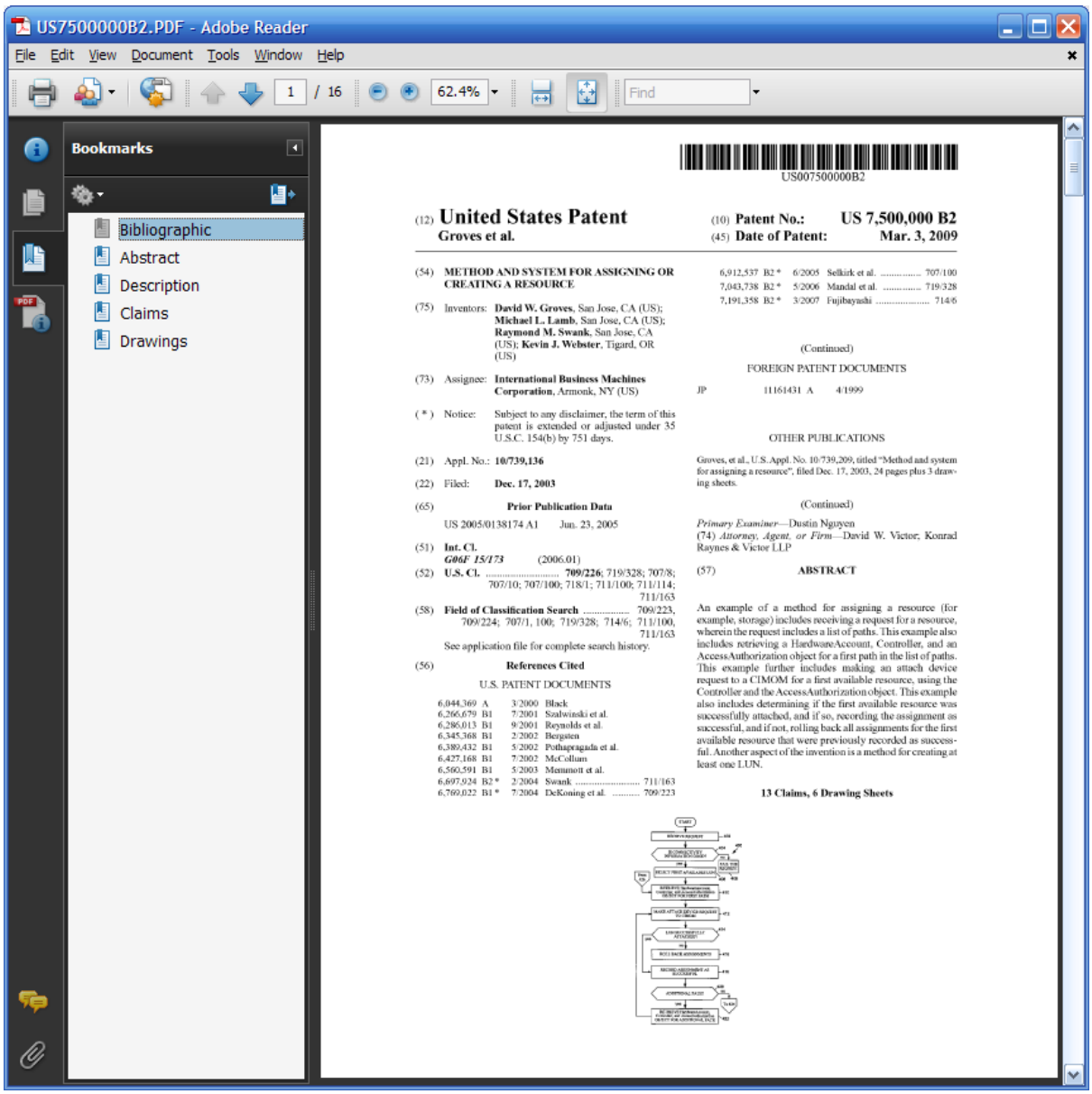
## 7.6 Image Types

IPDD has three image types which can be retrieved by using the IPDD Service or the IPDD gateway. Below is an example and description of each of the image types.

### 7.6.1 PDF

The PDF type is the complete publication in one single PDF file. The PDF file is bookmarked in most cases also full text searchable.

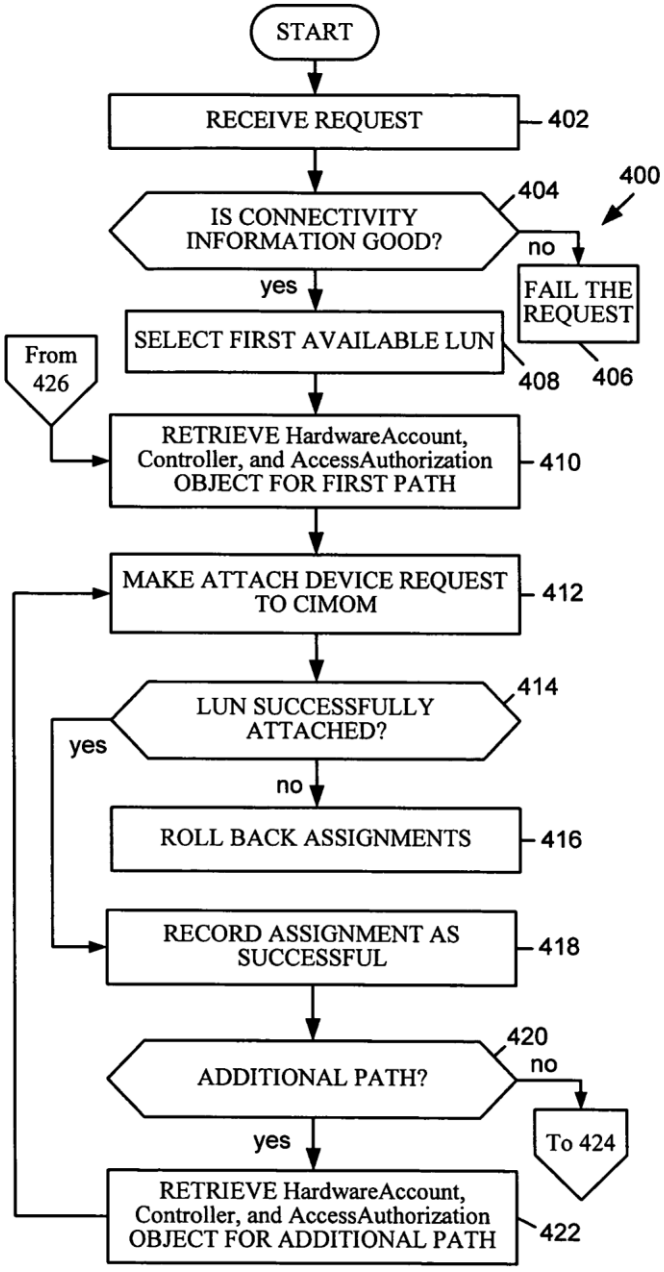
Below is an example of a PDF from publication US7500000B2 with the bookmark information on the left side.



### 7.6.2 Clip

The Clip is one Image in PNG format which is the first image found inside a publication. Normally this would be a front page image giving an overview of the publication.

Below is an example form publication US7500000B2 which shows the first drawing from the front page showed in PDF image example.



### 7.6.3 Images

The images type contains all of the images found inside one publication published in one zip file. This collection contains the clip and other images like mathematical formulas or chemical structures. This zip file also has an xml file in which the publication information along with the image information is being described.

Below is an example of a zip file containing all of the images available in publication US7500000B1.