precisely

EngageOne Vault

Programming Reference Guide

Version 2021.1.2

Table of Contents

1 - Developing Vault applications

In this section

Overview

The primary Vault functions are divided across three separate services:

- Loader is responsible for automated loading of new data into the vault.
- Server executes database searches and services requests for raw document data.
- Rendering engine converts raw document data into application useful forms.

There are a wide variety of possible custom vault applications you may want to build such as: custom desktop client or extension to an existing desktop client, custom web site or extension to an existing web site, automated extraction and conversion and so on. In almost all cases, the application will communicate directly to the rendering engine. Database searches are relayed by the rendering engine to the server.

Conversion requests are executed inside the rendering engine and the conversion may make additional server requests, fetch resources and fetch raw document data. Multiple rendering engines may be deployed for the same vault and the rendering process is often resource intensive and additional engines provides additional capacity. Additional rendering engines can provide additional redundancy, particularly if some are deployed on separate machines. By default, the services listen on a specific port for all local interfaces:

- Server listens on port 6001 by default
- Loader listens on port 6002 by default
- Rendering engine listens on port 6003 by default

> **Note:** If you want to run multiple services of the same type, you will need to alter the listen port.

Message protocol

Applications interact with the rendering engine using messages. The application begins by opening a connection to the rendering engine. To initiate a request, the application sends a request message and some functions will require a second message containing additional data. When the rendering engine finishes executing the request, it sends a result message back. Some functions will return an additional data message and some functions do not return a result. The application can then send more requests as needed. Finally, the application disconnects from the rendering engine.

Logically, a message can represent several structures:

- A table of data with a certain number of rows and columns. Typically the order of rows is significant.
- A list of key-value pairs (which looks like a two column table) . Typically the order of rows is not significant.

- A binary object such as an image (which looks like a 1x1 table).
- The request and result messages are lists of key-value pairs . The request keys begin with “request.” The result keys begin with “result”.
- Request messages contain the pair: `request.function` and `<function name>`. Functions are in the format: `<component>.<subcommand>` typically (for example, `render.transform`) the pair: `request.blocks <number of blocks>`. Usually 0 but 1 for a few functions any additional parameters the function needs.
- Result messages contain a copy of the request key value pairs for reference. The pair: `result.status` and `<status>`; the status is 0 if the request succeeded, otherwise is a non-zero error number.

  The pair: result.message and `<error message>`; if the request failed, error message is a text description of the fault. The pair: `result.blocks` and `<number of blocks>` which must be either 0 or 1 currently may also contain additional informational key pairs as defined by the function.

For certain applications it is handy to be able to execute multiple requests at once. The pair: `request.sequence` and `<unique identifier>` causes the server to execute the request even if the previous request has not completed the results may be returned to the application in any order. The application uses the sequence number in the result to determine which result is which. If one of the results is very large, it can block smaller results until it has been sent. Note that internally the rendering engine may cache results. Cached results will have the pair: `result.cached` and <number of seconds the data has been in the cache>. Results over a certain size (for example, some images) will not be placed in the cache (it would overwhelm the cache). Results are only kept in the cache for a certain (configurable) time period. Error results are typically kept for a shorter period of time. Requests and results are typically logged by the services. Some functions spawn sub requests that can be logged, can be transmitted upstream to the server.

> **Note:** The services will disconnect clients that have been idle for a certain (configurable) time. If the application experiences a communication error do not retry immediately; either return an error to the caller or delay and retry a certain number of times before returning an error. You will want to avoid an avalanche of requests at the point where service is restored.

Data input and render output options

The data input and render output options supported by Vault are displayed below.

	Output Mode															
Input Mode	GIF	HTML	PDF	Raw	PNG	TIFF	BMP	Stream	Text	Collection	Comments	Doc Info	TIFF G4	XML	Print	Decode
AFP	X		X	X	X	X	X	X	X		X	X	X		X	X
AFPLINE	X		X	X	X	X	X	X	X		X	X	X		X	X
Metacode	X		X	X	X	X	X	X	X			X	X		X	X
DJDE	X		X	X	X	X	X		X			X	X		X	X
DJDELINE	X		X	X	X	X	X	X	X			X	X		X	X
Postscript	X		X	X	X	X	X	X	X			X	X		X	X
PDF			X	X								X				
MPTIFF	X		X	X	X	X	X					X	X		X	
Collection				X						X		X				
HTML		X		X								X				
SplitXML				X								X		X		

Render engine connection

The rendering engine tries to keep messages alive. When sending a request with a second, parameter message, combine them for transmission which helps avoid the Nagle algorithm issue without disabling it (which should be avoided in general).

It is important to test the connection for read and write-ability and time-out accordingly. For example, with “select” you don't want to hang on a dead network connection, upstream or downstream. Consistently check for socket errors and protocol errors (`result.status`). For requests with potentially large data blocks (such as PDFs) do not assume a message will fit in memory. On web servers, relay the data by chunk. For client apps, be prepared to spill large results to disk.

API sets

The Rendering engine server provides a set of APIs which provides web developer the ability to access the Vault servers (the Repository server or the Rendering Engine), and provides access to databases, accounts and documents stored in the Vault and also provides rendering capabilities so that renderings of documents in the Vault can be requested by client code. Alternatively you can use Rendering Engine protocols which allow you communicate with Rendering Engine servers across your network via TCP/IP sockets.

.NET API

e2NetRender: the latest UNICODE enabled .NET API, refer to **.NET API (e2NetRender)** on page 8 for detailed information on using this API.

Java API

The Java API is a Java class library which encapsulates access to the Vault servers (the Repository server and/or the Rendering Engine), Refer to **Java API** on page 54. This Java API is only compatible with the Vault 7.4 servers (and later).

Rendering engine protocols and message formats

You can build client applications using messages transmitted via TCP/IP sockets on your network. The Rendering Engine provides protocols (referred to as modes) that perform different functions which can be incorporated in your application.

2 - .NET API (e2NetRender)

Vault is shipped with a system rendering API for the .NET environment. This API is compatible with the Microsoft .NET Framework 4.5. It consists of five namespaces, refer to e2NetRender, e2NetRender.render2, e2NetRender.service, e2NetRender.model and e2NetRender.exception.

The sections that follow are intended to provide guidance required for creating your own e2Vault .NET programming project. Note that the sample code is based on VCS.net.

In this section

Programming procedure

1. Add a reference of `e2NetRender.dll`.
2. Import namespaces.
3. Create an object of class `RenderClient2`. It is recommended to use the new solution of creating an object of a Command or a specific command object (for example, `DocumentCommand`) over the old solution.
4. Get messages.
5. Set SSL parameters if SERVERS are working under SSL mode.
6. Connect or disconnect from the server.
7. Use `GetMyVersion()` to get the version number of the package.
8. Use `GetMyMsg()` to get the messages inside the service for every calling if necessary.
9. Use `SetCharsetName()/SetCodePage()` to set a new character encoding. The default character encoding is UTF8.
10. Set host IP and port number (Vault Repository Server and Vault Rendering Engine).
11. Use `connect()/close()` to connect to or disconnect from `SERVER/Render`. However, if you are using a specific command (for example, `DocumentCommand`) you do not need to call the methods of `connect()/close()` because the classes automatically call them. Instead, call the related methods (for example, `DocumentCommand.minimumDocumentList()`).
12. Use `RenderClient2.DatabaseList()` or `Command.dbList()` to get database information of the Vault system.
13. Use `RenderClient2.DatabaseInfo()` or `Command.dbInfo()` to get index information of a specific database.
14. Using the old solution:
    a) Use `RenderClient2.DatabaseSearch()` to search an index for a specific database.
    b) Use `RenderClient2.documentList()` to get document list.
    c) Use `RenderClient2.DatabaseResolve()` to get document file and offset/pointer.
    d) Use `RenderClient2.DocumentData()` to get all information of the document.
15. **Using the new solution, for a heavyweight search:**
    a) Use `Command.accountList()` to search an index in a specific database for account numbers (name and address associated with the account number).
    b) If the index is an account-based index, you'll receive an account number from the search result, otherwise an exception will be thrown.
    c) Use `Command.documentList()` or `DocumentCommand.documentList()` to get a document list based on the index name (general document index) if you have the account number; otherwise the search will based on the INVLINK index if you do not specify an index name.

16. **Using the new solution, for a lightweight search:**
    a) Use `Command.minimumAccountList()` to search an index in a specified database for account number.
    b) If the index is an account-based index, you get account number from the search result, otherwise throwing an exception.
    c) Use `Command.minimumDocumentList()` or `DocumentCommand.minimumDocumentList()` to get a document list based on the index name if you have the account number; or the search will based on INVLINK index if you don't specify an index name.
    d) Use `Command.documentInfo` or `DocumentInfoCommand.documentInfo()` to get all of the information of the document if you know the document file name and document file offset (file pointer).
17. Use `RenderClient2.RenderTransform()` to get document pages data through memory.
18. Use `RenderClient2.GetDocumentPagesByFile()` to get document pages data through a file.

    The sections that follow provide detailed information about the above steps.

Install CERTIFICATE in local or web machine via .NET API

1. Transform CERTIFICATE file into PFX or P12 file:

```
openssl pkcs12 -in test.crt -inkey test.key -export -out test.pfx
```

2. Transform PFX file into PEM file (PEM is the format that combines the CERTTIFICATE and PRIVATE KEY information.):

```
openssl pkcs12 -in test.pfx -out test.pem
```

3. Install the PFX format certificate (Windows XP):
    a) Choose **Start > Control Panel > Internet Options** and select the **Content** tab.
    b) Click **Certificates** and click **Trusted Root Certification Authorities** and click **Import**. The "Certificate Import Wizard" will launch.
    c) Click **Next** and use **Browse...** to select a PFX format certificate.
    d) Click **Next** and select **Place all certificate in the following store**.
    e) Click **Browse...** and select **Trusted Root Certification Authorities** and click **Next**.
    f) Click **Finish** then **OK** from the "Import was successful" dialog.
    g) You will now be able to see the CERTIFICATE in the list.

Add a reference of e2NetRender.dll

1. Choose the project that will be using `e2NetRender.dll`
2. Right click **References** inside the window of **Solution Explorer**
3. Choose **Add Reference...**
4. Choose the tab of **.NET**
5. Click "**Browse...**"
6. Choose the directory where `e2NetRender.dll` is installed and choose `e2NetRender.dll`
7. Click **OK** return the project.
   e2NetRender is now visible within your project's References of the Solution Explorer.

Import namespaces

using `e2NetRender`

using `e2NetRender.render2`

Create an object of class RenderClient2

```
string exmsg="";//will be used later
string msg="";//will be used later
string host="127.0.0.1";//will be used later
int    port=6003;//will be used later
e2NetRender.render2.RenderClient2 myClient=new
e2NetRender.render2.RenderClient2();
```

Get version number

Use `RenderClient2.GetMyVersion()` to get the version number of the package.

Get messages

Use `RenderClient2.GetMyMsg()` to retrieve RenderClient messages for every calling, as necessary

Set new character encoding

Use `RenderClient2.SetCharsetName()/SetCodePage()` to set new character encoding. The default character encoding is UTF8

Set Rendering Engine IP address and port number

This applies to the Vault Repository server and Rendering Engine.

```
myClient.SetHost(host);
myClient.SetPort(port);
```

Set up `RenderClient2` to use SSL servers

```
 RenderClient2 client=new RenderClient2();
  //set other parameters
  //...
  client.usesslsocket=1;                              #default value is 0
  client.targethostname = targethostname;             #server host name
  client.certificatefilename = certfilename;          #certificate file
name (PEM format)
  client.clientsslprotocol = Sslprotocols.Tls;      #set up Ssl protocol

  client.connect();
  //....
```

Use OPENSSL to test certificate or key files

1. Create certificate/key files:

Refer to "notes-SelfSigned-Certificate.txt"

2. Create PFX file:

```
openssl pkcs12 -in test.crt -inkey test.key -export -out test.pfx
```

Note: You can leave the password empty.

3. Create PEM file:

```
openssl pkcs12 -in test.pfx -out test.pem"
```

Note: The PFX password is the one entered above command. You have to enter a password for the PEM file.

4. Run server:

```
openssl s_server -port 8080 -cert test.crt -key test.key
```

Or

```
openssl s_server -port 8080 -cert test.pem
```

5. Run client:

```
openssl s_client -host servermachine -port 8080
```

Or

```
openssl s_cleint -host servermachine -port 8080 -cert test.crt -key
test.key
```

Or

```
openssl s_client -host servermachine -port 8080 -cert test.pem
```

Login user name and password

If the Vault Rendering Engine you are connecting to has authentication enabled, then you will need to setup the user name and password for the connection. The password may be encrypted using the Vault password encryption model.

```
RenderClient2 client=new RenderClient2();
client.Username="myuser";
client.Password="mypassword";
```

The settings should be before calling `client.connect()`

Connect or disconnect from the server

Use `RenderClient2.connect()/close()` to connect to or disconnect from the server:

```
myClient.connect();
//call RenderClient2.DatabaseList();
//...
//call RenderClient2.RenderTransform()
myClient.close();
```

RenderClient2.DatabaseList() get database information

```
        try
   {
       myClient.connect();
    e2DBList dbl=myClient.DatabaseList();
    myClient.close();

    if(dbl!=null)
    {
      for(int i=0;i<dbl.Size();i++)
     {
      e2Database db=(e2Database)dbl.Get(i);

      string name=db.name;
      string desc=db.desc;

      //use name/desc to do something
      //...
```

```
     }
    }
    else
    {
     msg="failed to get db list : "+myClient.GetMyMsg();
     //do something if failed
     //...
    }
   }
   catch(e2Exception ex)
   {
    exmsg=ex.Message;
    msg=myClient.GetMyMsg();
    //do something
    //...
   }
   catch(Exception ex)
   {
    exmsg=ex.Message;
    //do something
    //...
   }
   finally
   {
    //free resources
    myClient.close();
   }Use RenderClient2.DatabaseInfo() to get index information of a
specific database
         string db="default"; //the db is from above call

         myClient.connect();
         e2IndexList idxl=myClient.GetIndexList(db);
         myClient.close();

         string msg="";
         if(idxl==null)
         {
             msg="failed to get indexlist : "+myClient.GetMyMsg();
             //display message
             return;
         }

         for(int i=0;i<idxl.Size();i++)
         {
             e2Index idx=(e2Index)idxl.Get(i);

             int indexno=idx.indexno;
             string name=idx.name;
             string attr=idx.attributes;
             string =idx.;
             string desc=idx.desc;

             //use index info to do something
```

```
            . . .
        }
```

RenderClient2.DatabaseInfo() search index for database

```
          try
{
 string db="default"; //the db is from above call
 myClient.connect();
 e2IndexList idxl=myClient.DatabaseInfo(db);
 myClient.close();

 if(idxl!=null)
 {
   for(int i=0;i<idxl.Size();i++)
  {
   e2Index idx=(e2Index)idxl.Get(i);

   int indexno=idx.indexno;
   string name=idx.name;
   string attr=idx.attributes;
   string =idx.;
   string desc=idx.desc;

   //use index info to do something
   //...
  }
 }
 else
 {
   msg="failed to get indexlist : "+myClient.GetMyMsg();
  //do something
  //...
 }
}
catch(e2Exception ex)
{
 exmsg=ex.Message;
 msg=myClient.GetMyMsg();
 //do something
 //...
}
catch(Exception ex)
{
 exmsg=ex.Message;
 //do something
 //...
}
finally
{
 //free resources
 myClient.close();
}
```

RenderClient2.DatabaseSearch() search index for database

```
   {
    string dbname="default";

    e2SearchParameters2 param2=new e2SearchParameters2();
    //search solution 1:
    int    indexno=1;//from RenderClient2.DatabaseInfo()
    string index="wjctul";//from RenderClient2.DatabaseInfo()
    e2IndexType indextype=e2IndexType.NONE;
    indextype=e2Index.getIndexType(index);

    param2.searchmode=e2SearchMode.GENERIC;
    param2.dbname=dbname;
    param2.SetIndex(indexno, index);
    param2.prefix="";
    param2.first="";
    param2.maxresult=e2Consts.MAX_SEARCH;
    //search solution 2:
    //string account="12345";//when search INVLINK index, you should
know ACCOUNT and DATE
    //string date="2008/01/01";
    //
    //param2.searchmode=e2SearchMode.INVLINK;
    //param2.dbname=dbname;
    //param2.SetAccount_Date(account, date);
    //param2.prefix="";
    //param2.first="";
    //param2.maxresult=e2Consts.MAX_SEARCH;

    //search solution 3:
    //param2.searchmode=e2SearchMode.GUID;
    //param2.dbname=dbname;
    //param2.prefix="";
    //param2.first="";
    //param2.maxresult=e2Consts.MAX_SEARCH;

    //search solution 4:
    //param2.searchmode=e2SearchMode.IGUID;
    //param2.dbname=dbname;
    //param2.prefix="";
    //param2.first="";
    //param2.maxresult=e2Consts.MAX_SEARCH;

    //after calling RenderClient2.DatabaseSearch():
    //moredata=0, no more data.
    //moredata=1, means more data,
   //then set param.first="last result of MATCHED" to call the function
 again to get more.
    int moredata=0;
```

```
myClient.connect();
e2SearchList2 sl=myClient.DatabaseSearch(param2, out moredata);
//while(moredata==1)
//{
// param2.first=((e2SearchData2)sl.Get(sl.Size()-1)).matched;
// e2SearchList2 sl0=myClient.DatabaseSearch(param2, out moredata);

// //using result to do something
// //...
//}
myClient.close();

msg="";
if(sl==null)
{
 msg="failed to get searchlist : "+myClient.GetMyMsg();
 //display message
 return;
}

//get title
e2SearchTitle2 title=sl.title;
int extsize=0;
string[] exttitle=title.exttitle;
if(title.exttitle!=null)
{
 extsize=title.exttitle.Length;
}
//get search data
for(int i=0;i<sl.Size();i++)
{
 //
 e2SearchData2 sd=(e2SearchData2)sl.Get(i);
 if(indextype==e2IndexType.CUSTOMER_RECORD)
 {
  string match=sd.matched;
  string acc=sd.account;
  string dat=sd.date;
  string type=sd.format;
  string file=sd.file;
  string offset=sd.pointer;
  int pages=sd.pages;
  //
  //these parameters can be used to get document pages
  //
 }
 else if(indextype==e2IndexType.DOCUMENT_RECORD)
 {
  string match=sd.matched;
  string acc=sd.account;
  string offset=sd.pointer;
  //
```

```
      //need to use ACCOUNT to call
      //RenderClient2.Resolve()/RenderClient2.DocumentData()
      //to get more detail information of the document.
      //
     }
     else
     {
      //unknown index type
      msg="unknown index type [ "+index+" ]";
      return;
     }

     //get extended data
     string[] extdata=sd.extdata;
     extsize=0;
     if(sd.extdata!=null)
      extsize=sd.extdata.Length;

     //use data to do something
     //...
   }
  }
  catch(e2Exception ex)
  {
   exmsg=ex.Message;
   msg=myClient.GetMyMsg();
   //do something
   //...
  }
  catch(Exception ex)
  {
   exmsg=ex.Message;
   //do something
   //...
  }
  finally
  {
   //free resources
   myClient.close();
  }
```

RenderClient2.DatabaseResolve() for document file and offset pointer

```
try
   {
    string db="default";//db name is from GetDBList()
    string account="12345";//account number from
RenderClient2.DatabaseSearch() or you already know it
    string docdate="2008/01/01";
    string docfile="";
    string docoffset="";

    myClient.connect();
   bool bret=myClient.DatabaseResolve(db, account, docdate, out docfile,
 out docoffset);
    myClient.close();

    //
    if(!bret)
    {
     msg="failed to get document info : "+myClient.GetMyMsg();
     //display message
     return;
    }
   }
   catch(e2Exception ex)
   {
    exmsg=ex.Message;
    msg=myClient.GetMyMsg();
    //do something
    //...
   }
   catch(Exception ex)
   {
    exmsg=ex.Message;
    //do something
    //...
   }
   finally
   {
    //free resources
    myClient.close();
   }
```

RenderClient2.DocumentData() get document information document

```
try
    {
    string docfile="doc_file_name";//from RenderClient2.DatabaseResolve()

    string docoffset="000000000000";//from RenderClient2.DatabaseResolve()

     e2DocumentDataList ddl=null;

     myClient.connect();
     ddl=myClient.DocumentData(docfile, docoffset);
     myClient.close();

     //
     if(ddl!=null)
     {
      //string account=
      //string docfile=
      //string docoffset=
      string docdate=ddl.GetDocDate();
      string doctype=ddl.GetDocType();
      string address=ddl.GetDocAddress();
      string name=ddl.GetDocName();
      int    totalpages=ddl.GetDocPages();
     }
     else
     {
      msg="failed to get document information : "+myClient.GetMyMsg();
      //display message
      return;
     }
    }
    catch(e2Exception ex)
    {
     exmsg=ex.Message;
     msg=myClient.GetMyMsg();
     //do something
     //...
    }
    catch(Exception ex)
    {
     exmsg=ex.Message;
     //do something
     //...
    }
    finally
    {
```

```
    //free resources
    myClient.close();
   }
```

RenderClient2.RenderTransform() get document pages data

```
try
   {
    e2RenderParameters param=new e2RenderParameters();

    param.parametertype=e2DocParameterType.NORMAL;
    param.dbname="default";//db name from RenderClient2.DatabaseList()

    //get below parameters from
DatabaseSearch()/DatabaseResolve()/DocumentData()
    string account="12345";
    string docdate="2007/01/01";
    string doctype="afp";
    string docfile="filename";
    string docoffset="00000C00";
    param.SetNormalParameters(account, docdate, doctype, docfile,
docoffset);

    param.startpage=1;
    param.totalpages=1;

    param.SetOutputType(0);//0=gif,2=pdf, 3=raw, 4=png, 5=tiff, 9=text
    param.resolution=800;//512, 640, 800, 1024, 1280
    param.orientation=0;//0=0, 1=90, 2=180, 3=270

    //param.transformmode=e2TransformMode.mem; //default mode is memory
 mode

    //
    myClient.connect();
    e2RenderPages pages=myClient.RenderTransform(param);
    myClient.close();

    if(pages==null)
    {
    msg="failed to get document pages by memory : "+myClient.GetMyMsg();

     //display message
     return ;
    }

    //use pages data to do something
    byte[] pagesbytes=pages.pagesdatabytes;
```

```
    int    pagessize=pages.pagesdatasize;
    //...
   }
   catch(e2Exception ex)
   {
    exmsg=ex.Message;
    msg=myClient.GetMyMsg();
    //do something
    //...
   }
   catch(Exception ex)
   {
    exmsg=ex.Message;
    //do something
    //...
   }
   finally
   {
    //free resources
    myClient.close();
   }
```

RenderClient2.RenderTransformByFile() get document pages data

```
try
   {
    e2RenderParameters param=new e2RenderParameters();

    param.parametertype=e2DocParameterType.NORMAL;

    param.dbname="default";//db name from RenderClient2.DatabaseList()

   //account/date/type/file/offset are from RenderClient2.DocumentData()

    string account="12345";
    string docdate="2008/01/01";
    string doctype="Afp";
    string docfile="docfilename";
    string docoffset="000000000000";
    param.SetNormalParameters(account, docdate, docfile, doctype,
docoffset);

    param.startpage=1;
    param.totalpages=1;

    param.SetOutputType(0);//0=gif,2=pdf, 3=raw, 4=png, 5=tiff, 9=text
    param.resolution=800;//512, 640, 800, 1024, 1280
```

```
    param.orientation=0;//0=0, 1=90, 2=180, 3=270

    param.transformmode=e2TransformMode.file;
    string outputfilename="test.gif";
    param.outputfilename=outputfilename;

    //
    myClient.connect();
    int filesize=myClient.RenderTransformByFile(param);
    myClient.close();

    if(filesize<=0)
    {
     msg="failed to get document pages by a file [ "+outputfilename+" ]
 : "+myClient.GetMyMsg();
     //display message
     return ;
    }

    //use the pagesdata inside the file to do something
    //....
   }
   catch (VaultConnectionException e)
             {
                 System.Console.WriteLine("a VaultConnectionException :
" + e.Message);
                 return;
             }
             catch (VaultServerErrorException e)
             {
                 System.Console.WriteLine("a VaultServerErrorException :
 " + e.Message);
                 return;
             }

   {
    exmsg=ex.Message;
    msg=myClient.GetMyMsg();
    //do something
    //...
   }
   catch(Exception ex)
   {
    exmsg=ex.Message;
    //do something
    //...
   }
   finally
   {
    //free resources
    myClient.close();
   }
```

How to use the new solution

The steps of the environment settings are the same as those previously listed. The only difference of using the new solution is that you need to create different objects and call different methods. These are more convenient and easy to use.

```
string host="127.0.0.1";
 int port=6003;
Command cmd = null;
string dbname="";
string index="";
string searchkey="";
string account=""
string date="";
int pageno=1;
int pagesize=20;
string file = "";
string pointer = "";
try
{
   // set up login username / password if required
      string user="myuser";
      string password="mypassword";
   //create an object of Command
   cmd = new Command(host, port;);
   cmd.Username=user;
   cmd.Password=password;
  //connect to server
   cmd.connect();
      //
   //call methods
   //
   //get database list from server side
   System.Console.WriteLine("call Command.dbList() :");
   List<e2Database> dblist = cmd.dbList();
   dblist.ForEach(
     delegate(e2Database db)
     {
      System.Console.WriteLine(db.ToString());
     }
   );
   //get database information for a specific database
   System.Console.WriteLine("call Command.dbInfo() :");
   dbname = "trymeafp";
   List<e2Index> dbinfo = cmd.dbInfo(dbname);
   dbinfo.ForEach(
     delegate(e2Index idx)
     {
       System.Console.WriteLine(idx.ToString());
```

```
      }
  );
    //lightweight search account-numbers
    System.Console.WriteLine("call Command.minimumAccountList() :");
    dbname = "trymeafp";
    index = "name";
    searchkey = "A";
    pageno = 1;
    pagesize = 20;
    List<MinimumAccount> minacctlist=cmd.minimumAccountList(dbname,index,
   searchkey, pageno, pagesize);minacctlist.ForEach(
     delegate(MinimumAccount minacct)
     {
        System.Console.WriteLine(minacct.ToString());
     }
    );
    //heavyweight search account-numbers
    System.Console.WriteLine("call Command.accountList() :");
    dbname = "trymeafp";
    index = "name";
    searchkey = "A";
    pageno = 1;
    pagesize = 20;
     List<Account> acctlist = cmd.accountList(dbname, index, searchkey,
       pageno, pagesize);acctlist.ForEach(
      delegate(Account acct)
      {
        System.Console.WriteLine(acct.ToString());
      }
     );
    //lightweight search documents based on non-INVLINK index
    System.Console.WriteLine("call Command.minimumDocumentList() based
on  GUID index :");
    dbname = "trymeafp";
    index = "GUID";
    searchkey = "";
    account = null;
    pageno = 1;
    pagesize = 20;
    List<MinimumDocument> mindoclist = cmd.minimumDocumentList(dbname,
index, searchkey, account, pageno, pagesize);
    mindoclist.ForEach(
      delegate(MinimumDocument mindoc)
      {
        System.Console.WriteLine(mindoc.ToString());
      }
    );
    //lightweight search documents based on INVLINK inde
   System.Console.WriteLine("call Command.minimumDocumentList() based
on INVLINK index :");
   dbname = "trymeafp";
   account = "12345678";
   date = "20";
```

```
   pageno = 1;
   pagesize = 20;
  List<e2DocumentEx> docexlist = cmd.minimumDocumentList(dbname, account,
 date, pageno, pagesize);
   docexlist.ForEach(
      delegate(e2DocumentEx docex)
      {
         System.Console.WriteLine(docex.ToString());
      }
   );
    //heavyweight search documents based on INVLINK index
   System.Console.WriteLine("call Command.documentList() based on INVLINK
 index :");
    dbname = "trymeafp";
    account = "12345678";
    date = "20";
    pageno = 1;
    pagesize = 20;
    List<Document> docinvlist = cmd.documentList(dbname, account, date,
 pageno, pagesize); docinvlist.ForEach(
      delegate(Document docinv)
      {
        System.Console.WriteLine(docinv.ToString());
      }
   );
    //get document attributes
    System.Console.WriteLine("call Command.documentInfo() :");
    dbname = "trymeafp";
    account = "12345678";
    date = "20";
    pageno = 1;
    pagesize = 20;
    DocumentInfo docinfo = cmd.documentInfo(dbname, file, pointer);
    if (docinfo != null)
    {
    System.Console.WriteLine(docinfo.ToString());
    }
    else
    {
    System.Console.WriteLine("null object of DocumentInfo :
"+cmd.GetMyMsg());
    }
}
catch (VaultAuthException e)
{
 System.Console.WriteLine("a VaultAuthException : " + e.Message);
 return;
}
catch (VaultCryptoException e)
{
 System.Console.WriteLine("a VaultCryptoException : " + e.Message);
 return;
}
```

```
catch (VaultConnectionException e)
{
   System.Console.WriteLine("a VaultConnectionException : " + e.Message);

   return;
}
catch (VaultServerErrorException e)
{
   System.Console.WriteLine("a VaultServerErrorException : " + e.Message);

   return;
}
catch (e2Exception e)
{
   System.Console.WriteLine("an e2Exception : " + e.Message);
   return;
}
catch (Exception e)
{
   System.Console.WriteLine("an exception : " + e.Message);
   return;
}
finally
{
   if (cmd != null)
   {
   cmd.close();
   }
}
```

How to use a specific command

```
string dbname = "trymeafp";
string index = "GUID";
string searchkey = "";
string date = "";
string account = "12345678";
int pageno = 1;
int pagesize = 20;
//set up login username and password if required
string user="myuser";
string password="mypassword";
DocumentCommand cmd = null;
try
{
    //create an object of DocumentCommand
    cmd = new DocumentCommand(host, port);
   //call methods
cmd.Username=user;
cmd.Password=password;
   //
   //for minimum document search
   //
   //focusing on any index
   System.Console.WriteLine("minimum search on any index :");
   List<MinimumDocument> docminlist = cmd.minimumDocumentList(dbname,
index, searchkey, account, pageno, pagesize);

   int count = 1;
   docminlist.ForEach(
      delegate(MinimumDocument doc
      {
      System.Console.WriteLine("    "+count + " :: "+doc.ToString());
      count++;
      }
   );
   //focusing on INVLINK index
   System.Console.WriteLine("minimum search on INVLINK index :");
  List<e2DocumentEx> docexlist = cmd.minimumDocumentList(dbname, account,
 date, pageno, pagesize);
   count = 1;
   docexlist.ForEach(
     delegate(e2DocumentEx docex)
     {
       System.Console.WriteLine("   " + count + " :: " +
docex.ToString());
       count++;
       }
    );
    //
```

```
    //for heavy document search
    //
    //focusing on any index
    System.Console.WriteLine("heavy search on any index :");
    List<Document> doclist = cmd.documentList(dbname, index, searchkey,
 account, pageno, pagesize);
    count = 1;
    doclist.ForEach(
      delegate(Document doc)
      {
        System.Console.WriteLine("    "+count + " :: " + doc.ToString());

          count++;
      }
    );
    //focusing on INVLINK index
    System.Console.WriteLine("heavy search on INVLINK index :");
    List<Document> docinvlist = cmd.documentList(dbname, account, date,
 pageno, pagesize);
    count = 1;
    docinvlist.ForEach(
      delegate(Document docinv)
      {
         System.Console.WriteLine("  " + count + " :: " +
docinv.ToString());
         count++;
         }
       );
}
catch (VaultAuthException e)
{
 System.Console.WriteLine("a VaultAuthException : " + e.Message);
 return;
}
catch (VaultCryptoException e)
{
 System.Console.WriteLine("a VaultCryptoException : " + e.Message);
 return;
}
catch (VaultConnectionException e){
  System.Console.WriteLine("a VaultConnectionException : " + e.Message);

  return;
  }
  catch (VaultServerErrorException e)
  {
    System.Console.WriteLine("a VaultServerErrorException : " +
e.Message);
    return;
  }
  catch (e2Exception e)
  {
    System.Console.WriteLine("an e2Exception : " + e.Message);
```

```
    return;
  }
  catch (Exception e)
  {
    System.Console.WriteLine("an exception : " + e.Message);
    return;
  }
   finally
  {
  }
```

Namespaces

e2NetRender

Classes

`public e2Consts`	includes the constants used inside the package.

	Public Properties
	`public enum e2IndexType`: database index type
	`e2IndexType.NONE`
	`e2IndexType.CUSTOMER_RECORD`
	`e2IndexType.DOCUMENT_RECORD`

	`public enum e2SearchMode`: define the search mode when searching an index inside a database
	`e2SearchMode.GENERIC`
	`e2SearchMode.GUID`
	`e2SearchMode.IGUID`
	`e2SearchMode.INVLINK`

Classes

	`public enum e2OutputType`: output type value. Used in: `RenderClient2.RenderTransform()`/`RenderClient2.RenderTransformByFile()`.
	`e2OutputType.GIF`
	`e2OutputType.PDF`
	`e2OutputType.RAW`
	`e2OutputType.PNG`
	`e2OutputType.TIFF`
	`e2OutputType.BITMAP`
	`e2OutputType.STREAM`
	`e2OutputType.TEXT`
	`e2OutputType.COLLECT`
	`e2OutputType.COMMENTS`
	`e2OutputType.DOCINFO`
	`e2OutputType.TIFFG4`

	`public enum e2TransformMode`: used in `RenderClient2:RenderTransform()` or `RenderClient2:RenderTransformByFile()`
	`e2TransformMode.mem`
	`e2TransformMode.file`

	`public enum e2DocParameterType`: used in `RenderClient2.RenderTransform()` or `RenderClient2.RenderTransformByFile()`
	`e2DocParameterType.DocumentGUID`
	`e2DocParameterType.InstanceGUID`
	`e2DocParameterType.NORMAL`

`e2DataType`	Base type of data item classes: `e2Database`, `e2Index`, `e2Searche`, `e2SearchTitle2`, `e2SearchData2`, `e2Document`, `e2DocAttribute`, `e2RenderPages`, `e2KeyValue`, `e2DocumentPageRecord`, `e2DocumentPage`, `e2DocumentData`, `e2ResourceInfo`, `e2StatTitle`, `e2Stat`, `e2FileAttr`, `e2FileStatus`.

Classes

`e2DataList`	Base of collection classes: `e2DBList`, `e2IndexList`, `e2SearchList`, `e2SearchList2`, `e2DocumentList`, `e2DocAttributeList`, `e2DocPageList`, `e2KeyValueCollect`, `e2DocumentPageArray`, `e2DocumentDataList`, `e2ResourceList`, `e2StatList`, `e2FileAttrList`, `e2FileStatusList`. **Public Methods** `public System.Int32 Add (e2NetRender.e2DataType data)`: adds data type `public e2NetRender.e2DataType Get (System.Int32 pos)`: gets the data type at a specific position. `public System.Int32 Size ()`: gets size of the collection for database in Vault.
`e2Database`	`public string name [ get, set]`: database name. `public string desc [ get, set]`: database description
`e2DBList`	Vault database collection, see `e2DataList`.
`e2Index`	**Index details** **Public Properties** `public int indexno [ get, set]`: the index number. `public string name [ get, set]`: the index name. `public string [ get, set]`: the flag of this index. `public string attributes [ get, set]`: the attributes of this index. `public string desc [ get, set]`: the description of this index.
`e2IndexList`	Index collection, see `e2DataList`. `e2Search`, `e2SearchTitle`, `e2SearchData`: search in Vault.

Classes

`e2SearchTitle`	**Title details** **Public Properties** `public string matched [ get, set]`: the title name of the matched string `public string account [ get, set]`: the title name of the account `public string date [ get, set]`: the title name of the document date `public string type [ get, set]`: the title name of the document type `public string file [ get, set]`: the title name of the document file `public string offset [ get, set]`: the title name of the document offset `public string pages [ get, set]`: the title name of the document pages count `public string[] extendedtitle [ get, set]`: the title name of extended columns.
`e2SearchData`	**Search details** **Public Properties** `public string matched [ get, set]`: search result of matched data string. `public string account [ get, set]`: search result of the account. `public string date [ get, set]`: search result of the doc-date. `public string type [ get, set]`: search result of the doc-type. `public string file [ get, set]`: search result of the doc-file. `public string offset [ get, set]`: search result of the doc-offset. `public string pages [ get, set]`: search result of the document pages. `public string[] extendeddata [ get, set]`: search result of the extended column data.
`e2SearchList`	Search collection, see `e2DataList`

Classes

`e2Document`	**used for document list details** **Public Properties** `public string account [ get, set]`: the document's account. `public string date [ get, set]`: the document's date. `public string type [ get, set]`: the document's type. `public string file [ get, set]`: the document's file. `public string offset [ get, set]`: the document's offset. `public in pages [ get, set]`: the document's page count.
`e2DocumentEx`	**used for light-weight document list details** **Public Properties** `public string dbname [ get, set]`: the database name the document belongs to. `public string account [ get, set]`: the document's account. `public string date [ get, set]`: the document's date. `public string file [ get, set]`: the document's file. `public string offset [ get, set]`: the document's offset.
`e2DocumentList`	Document collection, see `e2DataList`
`e2DocumentExList`	**used for document light-weight searching collection** see e2DataList

Classes

`e2DocAttribute`	**Document attributes** **Public Properties** `public string attrname [ get, set]`: the attribute name. `public string attravail [ get, set]`: the attribute value. `public string reportlevel [ get, set]`: the report level value. `public string reportpages [ get, set]`: the report pages.
`e2DocAttributeList`	Document attribute collection, see `e2DataList`
`e2RenderPages`	**Retrieving document pages** **Public Properties** `public e2NetRender.e2OutputType outputtype [ get, set]`: the output type of the pages data. `public byte[] pagesdatabytes [ get, set]`: the document pages data bytes. `public int pagesdatasize [ get, set]`: the size of the page data. `public int startpage [ get, set]`: starting pageno of the page data. `public in totalpages [ get, set]`: total pages number of the document.

Classes

`e2SearchParameters2`	parameters for searching a database. Used in `e2RenderClient2.DatabaseSearch()`. **Public Properties** `public e2SearchMode [ get, set]`: database search mode, see `e2SearchMode` `public string dbname [ get, set]`: database name. `public int underindexno [ get, set]`: under index no. `public int indexno [ get]`: the no of the index. `public string index [ get]`: the of the index. `public bool SetIndex(int indexno, string)`: set index no and index. `public string account [ get]`: account number. `public string docdate [ get]`: document date. `public bool SetAccount_Date(string account, string date)`: set the account and date info. `public string guidstring [ get, set]`: the GUID string when searching GUID index `public string iguidstring [ get, set]`: the GUID string when searching IGUID index `public string prefix [ get, set]`: prefix string when search INVLINK index. `public string fields [ get]`: the field value `public string titles [ get]`: the titles `public bool SetFields_Titles(string fields, string titles)`: set the fields string and titles string `public string first [ get, set]`: the first matched partial string `public int reverse [ get, set]`: if search in reverse or not. 1=reverse searching `public int maxresult [ get, set]`: the max number of search results

Classes

`e2RenderParameters`	parameters of getting document pages. Used in `e2RenderClient2.RenderTransform()` or `e2RenderClient2.RenderTransformByFile()` **Public Methods** `public e2DocParameterType [ get, set]`: //if using account/date/type/file/offset or GUID string `public string dbname [ get, set]`: database name. `public string account [ get]`: account number. `public string docdate [ get]`: document date. `public string docfile [ get]`: document file. `public string doctype [ get]`: document type. `public string docoffset [ get]`: document offset. `public SetNormalParameters(string account, string docdate, string doctype, string docfile, string docoffset)`: set parameters of account/date/type/file/offset `public string guidstring [ get]`: get GUID string `public bool SetGUID(string guidstring)`: set GUID string. `public bool SetIGUID(string guidstring)`: set IGUID string. `public string iguidstring [ get]`: get IGUID string. `public int startpage [ get, set]`: starting page no. `public int totalpages [ get, set]`: total pages number. `public e2OutputType outputtype [ get, set]`: output type. Can use `SetOutputType()` to set the value.

Classes

`e2RenderParameters` (cont.)	`public System.Int32 SetOutputType ( System.Int32 output )`: Set the output type of the page data : 0=GIF, 2=PDF, 3=RAW, 4=TIFF, 5=PNG, 9=TEXT. `public System.Int32 SetOutputType ( System.String outtype )`: Set the output type of the page data : "GIF", "PDF", "RAW", "TIFF", "PNG", "TEXT". `public System.Int32 GetOutputType ()`: return output type. `public int background [ get, set]`: need background (for PDF)? 1=yes, 0=no. indicates whether the background should be shown in PDF export mode `public int orientation [ get, set]`: page's orientation: 0=0 degree, 1=90 degree. 2=180, 3=270. `public int resolution [ get, set]`: page's resolution : 512/640/800/1024/1280. `public int cpix [ get, set]`: for text output : the characters per inch horizontally for text conversion `public int cpiy [ get, set]`: for text output: the characters per inch vertically for text conversion `public string textencoding [ get, set]`: use the specific text encoding for text output. `public int mark [ get, set]`: start text output with a byte order mark 0/1 `public string findtext [ get, set]`: next to highlight `public int findcolour [ get, set]`: color to highlight the text in `public int findcase [ get, set]`: 0/1, 1 if the match should be case sensitive `public int pdfsecuritymode [ get, set]`: for PDF security settings (user password, owner Password, permission); only valid when the parameter of `PDFEncryption=1` in `PROFILES.INI`. Otherwise, all parameters will be from `PROFILES.INI`.

Classes

`e2RenderParameters` (cont.)	`public string pdfuserpassword [ get, set]`: set PDF user password for the purpose of opening the PDF document. `public string pdfownerpassword [ get, set]`: set PDF owner password for the purpose of modifying the PDF document. `public int pdfpermission [ get, set]`: set PDF permission when opening the PDF document.

e2NetRender.render2

Classes

Base	Basic class of rendering functions **Public Methods and Properties** `public System.String Password`: login password `public System.String GetMyVersion ()`: get version number of the .NET API. `public System.String GetMyMsg ()`: get message. `public System.StringGetCharsetName`: get character set. `public System.String GetCharsetName ()`: get character set. `public System.Boolean SetHost ( System.String host )`: set the host name of the server. `public System.String GetHost ()`: get the host name of the server. `public System.Boolean SetPort ( System.Int32 port )`: set the port number of server. `public System.Int32 GetPort ()`: get the port number of SERVER. `public System.Int32 close ()`: close the connection to SERVER. `public System.Int32 connect ()`: connect to the HOST. `public System.Boolean IsConnected ()`: check if connect to SERVER. `public System.Boolean SetCharsetName(String charsetname)`: set new character encoding. `public System.String GetCharsetName ()`: get the character encoding name that API is using. `public System.Boolean SetCodePage(int codepage)`: Set a new character encoding name through codepage.

Classes

Base (cont.)	`public System.Int32 GetCodePage()`: get the codepage that API is using. `public System.Int32 usesslsocket [ get, set]`: when SERVER is using SSL(secured socket layer) mode, client must set up usesslsocket=1 to connect; default value is 0. `public System.String targethostname [ get, set]`: when `usesslsocket=1`, use this parameter to set up server target host name (common name of the certificate on server side) `public System.Security.Authentication.SslProtocols clientsslprotocol [ get, set]`: When usesslsocket=1, use this parameter to set up SSL protocol; mush match the protocol that SERVER is using

Classes

`e2RenderClient2`	Main rendering class. The base type is `BaseClient`, see `e2BaseClient`. **Public Methods** `public e2DBList RenderClient2:DatabaseList ()`: get a list of the databases in a vault along with their descriptions. `public e2IndexList RenderClient2:DatabaseInfo ( String dbname )`: get a list of searches available for a database. `public e2SearchList2 RenderClient2:DatabaseSearch ( e2SearchParameters2 param2 , out int moredata )`: search an index for matching records. `public Boolean RenderClient2:DatabaseResolve ( String dbname , String account ,String date , out String file , out String offset )`: used to determine if a customer or document record exists. Fetches the file and offset parameters. `public e2DocumentPageArray RenderClient2:DocumentPageArray ( String file , String recordoffset )`: fetch offsets of raw pages in a document page data file. `public e2DocumentPage RenderClient2:DocumentPage ( String file , String recordoffset , int pageno )`: fetch raw internal data for a document page. `public e2DocumentDataList RenderClient2: DocumentData ( String file , String recordoffset )`: retrieve report list and properties of a document.

Classes

`e2RenderClient2` (cont.)	`public e2RenderPages RenderClient2: RenderTransform (e2RenderParameters param)` : convert raw data into a application usable form through memory; used to render pages to GIF, PDF, text, etc. `public int RenderClient2: RenderTransformByFile (e2RenderParameters param)`: convert raw data into a application usable form through a file. Used to render pages to GIF, PDF, text, etc. `public e2ResourceList RenderClient2: ResourceList ( string resourcesetname, string filenamepattern )`: recursively enumerate upstream resource files. `public e2ResourceInfo RenderClient2: ResourceInfo ( String resourcesetname, String filename )`: Determine if a specified resource exists. Fetch basic information if it does. `public Byte[] RenderClient2: ResourceData ( String resourcesetname , String filename , long filesize )`: fetch a block of data from a resource. `public long RenderClient2: ResourceDataByFile ( String resourcesetname , String filename , long filesize , String outputfilename )`. `public e2StatList RenderClient2: StatList ( )`: fetch statistics about function execution times. `public String[] RenderClient2: ProfileList ( )`: list the names of available profiles known to the server. `public e2KeyValueCollect RenderClient2: ProfileData ( String profile )`: fetch key value pairs from a profile. `public e2FileAttrList RenderClient2: FileList ( String filenameprefixpattern )`

Classes

`e2RenderClient2` (cont.)	`public Boolean RenderClient2:FileData (String filename ,out String profile ,out String resource ,out String pagedatafile )`: retrieve metadata associated with a compressed file. `public Byte[] RenderClient2: FilePage ( String filename , String pageoffset )`: fetch a raw page from a compressed file. `public Boolean RenderClient2:FileIsOnline ( String filename ,out Boolean online )`: determine if a compressed file is online. `public Boolean RenderClient2: FileRecallOffline (String filename )`: request that an offline compressed file be placed online. `public long RenderClient2: PostscriptHeaderInfo (String filename )`: return size of Postscript file header. `public Byte[] RenderClient2: PostscriptHeaderData (String filename ,String offset ,int bytesread )`: fetch a block of data from a Postscript header. `public e2FileAttrList RenderClient2: CompressedList (String filenameprefixpattern )`: list the names of compressed files. `public Boolean RenderClient2:CompressedOpen (String filename ,out long filehandle )`: open a compressed file for access. `public Boolean RenderClient2:CompressedClose (long filehandle )`: close an open compressed file handle. `public e2FileStatusList RenderClient2: CompressedStatus (long filehandle )`: get the background scanning status for a compressed file. `public Byte[] RenderClient2:CompressedPage (long filehandle ,int pageno )`: fetch raw page data from a compressed file. `public e2DocumentDataList RenderClient2: CompressedDocData (long filehandle )`: get the simulated document data for a compressed file.

Classes

`e2RenderClient2` (cont.)	`public Boolean RenderClient2: CompressedFileData ( long filehandle , out String profile , out String resource , out String pagedatafile )`: get compressed file information associated with an open compressed file. `public Byte[] RenderClient2: CompressedProfileData ( long filehandle )`: get profile settings associated with an open compressed file. `public Boolean RenderClient2: CompressedPostscriptHeaderInfo ( long filehandle ,out long headerinfosize )`: determine if a given compressed file exists. Return the size of its Postscript file header. `public Byte[] RenderClient2: CompressedPostscriptHeaderData ( long filehandle ,String offset ,int sizeread )`: fetch a block of data from a Postscript header for a compressed file.

New data types

Note: The namespace is e2NetRender.model

Account search result data type

```
MinimumAccount class
{
String dbname; # database name the index resides on
String match; # the search column data matched, i.e. name-string for
Name index
String account; # the account number string associated the customer
String pointer; # the offset  to the account table
}
Account class
{
String dbname; # database name the index resides on
String match; # the search column data matched, i.e. name-string for
Name index
String account; # the account number string associated the customer
String name;# the customer name
String address; #the customer address
String pointer; # the offset to the account table
Dictionary<string,string> customFields; # the customer fields defined
in database.ini for this index.
}
```

Document search result data type

```
MinimumDocument class
{
 string dbname; # database name the index resides on
string match; # the search column data matched, i.e. GUID string for
GUID index, DATE for INVLINK index
string account; # the account number string associated the customer
string file; # the file-name of the document data;

string pointer; # the pointer in the document data file.
}
Document class
{
String dbname; # database name the index resides on
String match; # the search column data matched, i.e. GUID string for
GUID index, DATE for INVLINK index
String account; # the account number string associated the customer
String date;#the document date string
String file; # the file-name of the document data;
String pointer; # the pointer in the document data file.
Int pageCount; #the document page count
Int modes; #the supported document output modes
String profileFormat;#the document format defined in the PROFILE in
```

```
profiles.ini for this document
String documentType; #document type
Dictionary<string,string> customFields; # the customer fields defined
in database.ini for this index.}
```

Document attributes data type

```
DocumentInfo class
{
String dbname; # database name the index resides on
String account; # the account number string associated the customer
String date; # document date string
String name;# the customer name
String address; #the customer address
String invoice; #the document invoice number
String type; #the document type
Int pageCount; #the document page count
String masterGUID; # the document master GUID string
String instanceGUID; # the document instance GUID string
Int sectionCount; #the number of document sections
List<DocumentSection> sections; #the document section list
Dictionary<string, string> customAttributes; #the customer attributes
defined in JRN file for this document
}
```

New methods

Note: The namespace is e2NetRender.service.

Database information, account and document search processing

```
Command class
{
Methods:
Public List<e2Database> dbList();
Public List<e2Index> dbInfo(string dbname);
Public List<Account> accountList(string dbname, string index, string
searchkey, int pageno, int pagesize);
Public List<MinimumAccount> minimumAccountList(string dbname, string
index, string search, int pageno, int pagesize);
Public List<Document> documentList(string dbname, string index, string
searchkey, string account, int pageno, int pagesize);
Public List<e2DocumentEx> documentList(string dbname,  string account,
string date, int pageno, int pagesize);
Public List<MinimumDocument> minimumDocumentList(string dbname, string
index, string searchkey, string account, int pageno, int pagesize);
Public List<MinimumDocument> minimumDocumentList(string dbname, string
account, string date, int pageno, int pagesize);
Public DocumentInfo documentInfo(string dbname, string string file,
string pointer);
}
```

Remarks

Command class is a parent class for other commands classes. In order to to get data, create an object, and connect or close manually. For other sub-command classes (for example, `DatabaseListCommand`, `DocumentCommand`), they call `connect()/close()` methods. You need to create an object of the sub-command class, and call the related methods (no need to call `connect()` and `close()` methods).

`DatabaseListCommand` class

```
{
Methods:
Public List<e2Database> dbList();
}
```

`DatabaseInfoCommand` class

```
{
Methods:
Public List<e2Index> dbInfo(string dbname);
}
```

`AccountCommand` class

```
{
Methods:
Public List<Account> accountList(string dbname, string index, string
searchkey, int pageno, int pagesize);
Public List<MinimumAccount> minimumAccountList(string dbname, string
index, string searchkey, int pageno, int pagesize);
}
```

Remarks

These methods focus on account based indexes. The names of minimum* methods focus on columns defined inside the index, which means returned the columns in the index.

Parameters

`string dbname`: database name where the index resides

`string index`: account-based index name

`string searchkey`: search key used on the index. It will partially match (for example, it will be a name string for the Name index, an account number string for the Account index).

`Int pageno`: search time (1st, 2nd, 3rd, etc.). This should be greater than 0.

`Int pagesize`: size of returned result for every search. This should be greater than 0 (for example, 30).

`documentCommand` class

```
{
Methods:
Public List<Document> documentList(string dbname, string index, string
searchkey, string account, int pageno, int pagesize);
Public List<e2DocumentEx> documentList(string dbname,  string account,
string date, int pageno, int pagesize);
Public List<MinimumDocument> minimumDocumentList(string dbname, string
index, string searchkey, string account, int pageno, int pagesize);
Public List<MinimumDocument> minimumDocumentList(string dbname, string
account, string date, int pageno, int pagesize);
}
```

Remarks

The methods are focusing on the document indexes.

The names of minimum* methods are focusing on the columns defined inside the index which means only returned the columns in the index.

For methods without an index-name argument, they focus on the INVLINIK index.

Parameters

`string dbname`: database where the index resides

`string index`: document-based index name

`string searchkey`: search key used on the index. It will partially match.

`string account`: account number if the index has ACCOUNT number prefixed (contains an “a” flag)

`int pageno`: search time (1st, 2nd, 3rd, etc.). This should be greater than 0.

`int pagesize`: size of returned result for every search. This should be greater than 0.

`string date`: date string for INVLINK index. This can fully or partially match

> **Note:** These methods are based on a document-based index. If the index is an account-prefixed document index (invlink or invlink like), the `MinimumDocument` object in the returned list will have Account-number value. Otherwise, it will have null Account-number value.

`documentInfoCommand` class

```
{
Public DocumentInfo documentInfo(string dbname, string file, string
pointer);
}
```

Index examples:

The `GUID` index is not an account-prefix document index, so it will have three columns in the index(GUID string + File + Pointer). The `MinimumDocument` object in the returned `MinimumDocument` object list will have [GUID string(as MATCH column), File, Pointer] and null account number.

The `INVLINK` index is an Account-prefixed document index and it will have four columns (Account + Date + File + Pointer). The `MinimumDocument` object in the returned `MinimumDocument` object list will have [Account, Date (as MATCH column), File, Pointer].

3 - Java API

In this section

Software requirements

The software required is:

Java 2 Platform, Standard Edition, v 7 (J2SE 7 Java platform/JDK).

Installing the Java API

If using the command line to build and run your custom application: specify the Java API JAR file using the "`-cp`" option:

- Compiling your custom application: `javac -cp e2Vault.jar` and `MyCustomApp.java`
- Running your custom application: `java -cp e2Vault.jar` and `MyCustomApp.java`
- For integration into an IDE (Netbeans, Eclipse, etc.) follow your IDE vendor's instructions for integrating the Java API class library.

Overview of classes and usage details

All classes provided by the Vault Java API are defined in the `com.g1.e2.vault` package. The primary class for use by client code is `VaultClient` and it serves as the main interface class between the Vault Java API and client code.

Data stored in the Vault is exposed through the following classes:

- `Database`: represents a database on the Vault server
- `SearchIndex`: represents one of many search indexes available for a database on the Vault server
- `Account`: represents an account (customer account or customer record) on the Vault server
- `Document` and `DocumentEx`

  Both of these classes are used to return a document list from the server. Depending on how detailed you need the search to be, you can choose between the following two classes:

  - `Document`: represents a document (document record) on the Vault server

    Note: This class is used for performing detailed document searches. It needs to open each DRD file which has a document that matches the search criteria so it can consume

a great deal of server resources generating the search result. Therefore it is recommended that you only use this when you need to run a detailed search. Otherwise it is recommended that you use the `DocumentEx` class if possible because it is more efficient.

- `DocumentEx`: represents document parameters from the INVLINK index on the Vault server

  Note: This class is used for performing less detailed but faster document searches. Because it only uses the INVLINK index for returning documents, it is a more efficient way of searching for documents. It is recommended that you use this method of returning the document list whenever possible.

For information on using these classes, please refer to **Classes and usage details** on page 62.

Connection and disconnection

Connection to a Vault server can be established as follows:

```
VaultClient myClient = new VaultClient();

try {
    // connect to the Vault server running on localhost on
    // port 6003
    myClient.connect("localhost", 6003);
}
catch (UnknownHostException e) {
    ...
}
catch (IOException e) {
    ...
}

...

try {
    // terminate connection when done
    myClient.shutdown();
}
catch (IOException e) {
    ...
}
```

Connecting to an SSL-enabled Vault server

Connecting to an SSL-enabled Vault server requires the server's SSL certificate in a PEM format file with a ".crt" extension (for example, `e2vault-server.crt`), and the contents of the certificate file must begin with "-----BEGIN CERTIFICATE-----" and end with "-----END CERTIFICATE-----" markers.

Connecting to an SSL-enabled Vault server is done using the following API method:

```
VaultClient.connectSSL(String hostname, int port, TruststoreParams
tsParams, KeystoreParams ksParams)
```

This can be done in either one of the following two ways:

- `VaultClient.connectSSL(String hostname, int port, (TruststoreParams)null, (KeystoreParams)null)`

  Use the API method if you have imported the Vault server's SSL certificate into the Java system-wide CA certificates truststore (usually `\path\to\jre\lib\security\cacerts`).

- `VaultClient.connectSSL(String hostname, int port, new TruststoreParams(truststorePath, truststorePasswd), (KeystoreParams)null)`

  Use this API method if you have imported the Vault server's SSL certificate into a separate, private truststore file. Refer to the javadocs for more details on the `com.g1.e2.vault.TruststoreParams` class.

  If the Vault server has enabled SSL Client Authentication, then the `ksParams` parameter must be non-null and must contains details of a JKS-format keystore containing the client application SSL credentials, which will be used for authenticating it to the Vault server. Refer to the javadocs for more details on the `com.g1.e2.vault.KeystoreParams` class.

Importing Vault server's SSL certificate in the Java system wide CA certificates trustore

Before starting:

Make a backup copy of your Java system-wide CA certificates truststore before importing the Vault server SSL certificate:

```
cd /path/to/Java JRE/lib/security
```

- On Windows platforms:

  copy `cacerts cacerts.ORIG`

- On LINUX platforms:

  copy `cacerts cacerts.ORIG`

If a JDK is installed as default Java

Windows platforms

```
keytool -import -v -alias e2VaultServer -file e2vault-server.crt -keystore
 "%JAVA_HOME%\jre\lib\security\cacerts"
```

LINUX platforms

```
keytool -import -v -alias e2VaultServer -file e2vault-server.crt -keystore
 "$JAVA_HOME/jre/lib/security/cacerts"
```

If a JRE installed as default Java

Windows platforms

```
keytool -import -v -alias e2VaultServer -file e2vault-server.crt -keystore
 "%JAVA_HOME%\lib\security\cacerts"
```

INIX platforms

```
keytool -import -v -alias e2VaultServer -file e2vault-server.crt -keystore
 "$JAVA_HOME/lib/security/cacerts"
```

Importing the Vault server SSL certificate

Import the Vault server's SSL certificate into a separate, private JKS-format truststore as follows:

```
keytool -import -v -alias e2VaultServer -file e2vault-server.crt -keystore
 e2vault-server.jks -storepass <your Truststore password> -storetype JKS
```

SSL client authentication

For SSL Client Authentication, a JKS-format keystore is needed that will contain both the SSL Certificate and corresponding Private Key of your Java client application.

1. Generate an SSL Certificate and Private Key using OpenSSL that will be used to authenticate your Java client application to the SSL-enabled Vault server. You should have two files in PEM format:

   Example

   `javaclient-cert.crt` (SSL Certificate in PEM format)

   `javaclient-key.pem` (Private Key in PEM format)

2. Convert the SSL Certificate to DER format:

   Example

   ```
   openssl x509 -inform PEM -in javaclient-cert.crt -outform DER -out
   javaclient-cert.der
   ```

3. Convert the Private Key to the PKCS#8 DER format:

   Example

   ```
   openssl pkcs8 -topk8 -nocrypt -inform PEM -in javaclient-key.pem
   -outform DER -out javaclient-key.der
   ```

4. Use the supplied ImportKey utility application (in the `contrib/` folder of the JavaAPI distribution) to import both the Private Key and SSL Certificate files in DER format (created above) into a new JKS-format keystore:

   Example

   ```
   javac ImportKey.java
   java ImportKey javaclient-key.der javaclient-cert.der javaclient
   Enter a password for the keystore: password
   Creating new keystore file: javaclient-keystore.jks
   One certificate, no chain.
   Key and certificate stored with alias "javaclient" to keystore file
   "C:\temp\javaclient\javaclient-keystore.jks".
   ```

   Note: The ImportKey utility application will create a new entry for the SSL Certificate and Private Key with the same password as for the keystore.

After successfully completing the previous steps, you can connect to your SSL-enabled Vault server and supply the necessary credentials (SSL Certificate and Private Key) for your Java client application to authenticate itself to the Vault server as follows:

Example

```
final String ksPath = "C:\\temp\\javaclient\\javaclient-keystore.jks";
final String ksPasswd = "password";
final String keyPasswd = ksPasswd;
KeystoreParams ksParams =
new KeystoreParams(ksPath, ksPasswd.toCharArray(),
keyPasswd.toCharArray());
VaultClient vaultClient = new VaultClient();
VaultClient.connectSSL(hostname, port, null, ksParams);
...
```

Classes and usage details

Database class

The `Database` class represents a Database on the Vault server. The list of Databases on the Vault server can be obtained using the `VaultClient.getDatabases()` method:

```
Set<Database> databases = myClient.getDatabases();
for (Database db : databases) {
    ...
}
```

A specific Database can also be retrieved by it's name:

```
Database db = myClient.getDatabase("Invoices");
if (db != null) {
    ...
}
```

SearchIndex class

The `SearchIndex` class represents one of many search indexes available on the Vault server, and is used for searching for accounts and documents. Search indexes are associated with each database and each search index can be used for a specific type of search.

The list of search indexes available for a database can be obtained as follows:

```
Database db = myClient.getDatabase("Invoices");
if (db != null) {
    Set<SearchIndex> searchIndexes = db.getSearchIndexes();
    ...
}
```

A standard, default Vault configuration defines the following search indexes for each Database in the Vault:

```
SI number     SI name     Type of SI     Purpose

1             account     account        Search for Accounts by account number
2             name        account        Search for Accounts by (customer) name
3             address     account        Search for Accounts by (customer) address
4             invlink     document       Search for Documents associated with an Account (Account is required)
5             guid        document       Search for Documents by Master GUID             (Account is not required)
6             iguid       document       Search for Documents by Instance GUID           (Account is not required)
```

An account search index is a search index that returns matching account objects when it is searched on. A document search index is a search index that returns matching Document objects when it is searched on.

Various properties of the search indexes can be queried using the following methods of the `SearchIndex` class:

```
SearchIndex.getIndex()
SearchIndex.getName()
SearchIndex.getDescription()
SearchIndex.getFields()
SearchIndex.isAccountIndex()
SearchIndex.isDocumentIndex()
SearchIndex.isAccountRequired()
SearchIndex.isInvisible()
SearchIndex.containsMultipleRotations()
```

Search indexes can be obtained directly by their name and number using the `VaultClient.Database.findSearchIndexByName()` and `VaultClient.Database.findSearchIndexByNumber()` methods, respectively:

```
// find a search index by name - in this case the Account
// search index which is used to search for Accounts by
// account number
SearchIndex acctSearchIndex =
    db.findSearchIndexByName("account");
if (acctSearchIndex != null) {
    ...
}

// find a search index by number - in this case the search
// index with index number 5 which is the standard (Master)
// GUID search index
SearchIndex guidSearchIndex = db.findSearchIndexByNumber(5);
if (guidSearchIndex != null) {
    ...
}
```

Search indexes can also be obtained by specific predefined criteria by using the `VaultClient.Database.findSearchIndexByMatcher()` method. A number of predefined criteria are available for use with this method:

`SearchIndex.FindsAccountsByAccountNumber`: Match against account search indexes using an account number.

`SearchIndex.FindsAccountsByName`: Match against account search indexes using a name.

`SearchIndex.FindsAccountByAddress`: Match against account search indexes using an address.

`SearchIndex.FindsDocumentWithAccounts`: Match against document search indexes that require an account number.

`SearchIndex.FindsDocumentsWithoutAccounts`: Match against document search indexes that do not require an account number.

`SearchIndex.FindsDocumentWithMasterGUID`: Match against document search indexes using the Master GUID.

`SearchIndex.FindsDocumentWithInstanceGUID`: Match against document search indexes using the Instance GUID.

The previous code example can be rewritten as follows to use the predefined search index criteria:

```
// find an (account) search index that finds Accounts by
// account number
SearchIndex acctSearchIndex =
    db.findSearchIndexByMatcher(SearchIndex.FindsAccountByAccountNumber);
if (acctSearchIndex != null) {
    ...
}

// find a (document) search index that finds Documents by
// Master GUID
SearchIndex guidSearchIndex =
    db.findSearchIndexByMatcher(SearchIndex.FindsDocumentByMasterGUID);
if (guidSearchIndex != null) {
    ...
}
```

Once a particular search index of interest has been obtained, it can be used to search for accounts and/or documents.

Account class

Accounts are contained in databases on the Vault. Searching for accounts in a database requires an account search index:

```
// get the search index that finds Accounts by account number
SearchIndex acctSearchIndex =
    db.findSearchIndexByMatcher(SearchIndex.FindsAccountByAccountNumber);

// search for Accounts with account numbers matching the
// string "6107-"
SearchMatchesIterator<Account> smi =
    db.searchForAccounts(acctSearchIndex, "6107-");

// iterate over resulting/matching Accounts (if any)
if (smi.hasNext()) {
    while (smi.hasNext()) {
        ArrayList<Account> accounts = smi.next();
        ...
    }
}
```

The number of matching accounts returned in each iteration by the `SearchMatchesIterator<T>.next()` method can be specified if so desired by specifying the `maxResults` parameter to `VaultClient.Database.searchForAccounts()`:

```
// the maximum number of matching Accounts to return in
// each iteration
final int maxResults = 50;

// search for Accounts with account numbers matching the
// string "6107-"
SearchMatchesIterator<Account> smi =
    db.searchForAccounts(acctSearchIndex, "6107-", maxResults);

// iterate over resulting/matching Accounts (if any)
if (smi.hasNext()) {
    while (smi.hasNext()) {
        ArrayList<Account> accounts = smi.next();
        ...
    }
}
```

Also, all accounts present in a database can be retrieved by the `VaultClient.Database.getAllAccounts()` convenience method:

```
existing lines:
List<Account> allAcounts=
 db.getAllAccounts();
for (Account account:allAccounts){
  ...
}

List<Account> allAcounts=
 db.getAllAccounts();
or
List <Account> allAccounts=
  db.getAllAccounts(int maxResults);

for (Account account:allAccounts){
  ...
}
```

For the `getAllAccounts(int maxResults)` function, the maxResults parameter is the maximum results (or window size) returned from the server.

If you have problems with the server looping or too many results, you can process the problems through catching `ServerLoopingException` or `TooManyResultsException`.

When you catch the exception, use the indexcheck utility to check the duplicated size of the account on the server side, change the `maxResult` parameter and try again. You can put the parameter into a configuration file.

> **Note:** Only change the value through the configuration file when needed, as there is no need to change your code.

Document class

Documents stored in a Vault are always associated with a corresponding account. However, documents can be searched for with or without the corresponding account.

> **Note:** This class is used for performing detailed document searches. It needs to open each DRD file which has a document that matches the search criteria so it can consume a great deal of server resources generating the search result. Therefore it is recommended that you only use this when you need to run a detailed search. Otherwise it is recommended that you use the `DocumentEx` class if possible because it is more efficient.

Documents associated with an account can be searched as follows:

```
// Search for Documents associated with this Account whose
// date matches the year 2007
SearchMatchesIterator<Document> smi =
    account.searchForDocuments("2007");

if (smi.hasNext()) {
    while (smi.hasNext()) {
        ArrayList<Document> documents = smi.next();
        ...
    }
}
```

All documents associated with an account can be obtained using the `VaultClient.Account.getAllDocuments()` convenience method:

```
List<Documents>documents = account.getAllDocuments();
for(Document d:documents){
...
}
|
List<Documents>documents = account.getAllDocuments();
or
List<Documents>documents = account.getAllDocuments(int maxResults);
for(Document d:documents){
...
}
```

For the `getAllDocuments(int maxResults)` function, the `maxResults` parameter is the maximum results (or window size) returned from server.

If you have problems with the server looping or too many results, you can process the problems through catching `ServerLoopingException` or `TooManyResultsException`.

When you catch the exception, use `indexcheck` utility to check the size of the documents for this account on the server side, change `maxResult` and try again.

You can put the parameter into a configuration file, only change the value through the configuration file when needed as there is no need to change your code.

Documents can also be searched for without requiring an Account object. This can be done in one of two ways. The predefined `SearchIndex` matcher, `SearchIndex.FindsDocumentsWithoutAccounts`, can be used to first obtain a search index that doesn't require an account and then the search for documents can be done using that search index:

```
SearchIndex si =
 db.findSearchIndexByMatcher(SearchIndex.FindsDocumentsWithoutAccounts);
SearchMatchesIterator<Document> smi =
 db.searchForDocumentsWithoutAccounts (si, "1234");

if (smi.hasNext()) {
 while (smi.hasNext()) {
  List<Document> documents = smi.next();
...
 }
}
```

Instead of using a matcher to find the document search index that doesn't require an account, if the Vault has been customized with additional (custom) search indexes you can use any custom document search index to search for documents:

```
SearchIndex myCustomSI = null;

// find the custom Document search index that meets your
// requirements
Set<SearchIndex> searchIndexes = db.getSearchIndexes();
for (SearchIndex si : searchIndexes) {
    if (si.isDocumentIndex() && !si.isAccountRequired() &&
        /* some other custom property */) {
        myCustomSI = si;
        break;
    }
}

// now search for Documents in this Database (without needing
// an Account object) using this (custom) search index
SearchMatchesIterator<Document> smi =
    db.searchForDocumentsWithoutAccount(myCustomSI, "111-222-333");

if (smi.hasNext()) {
    while (smi.hasNext()) {
        ArrayList<Document> documents = smi.next();
        ...
    }
}
```

Documents can be searched for by GUIDs (master or instance) in a similar manner:

```
// search by Master GUID
SearchIndex si =
```

```
 db.findSearchIndexByMatcher(SearchIndex.FindsDocumentByMasterGUID);
SearchMatchesIterator<Document> smi =
 db.searchForDocumentsWithoutAccount(si,
"5F822EB8B3B763FB681E0894ED927F4B")

if (smi.hasNext()) {
 while (smi.hasNext()) {
  List<Document> documents = smi.next();
  }
}
```

Use the `SearchIndex.FindsDocumentByInstanceGUID` matcher if you wish to search by Instance GUID.

DocID and obtaining a document by DocID

A DocID (document ID) is an identifier associated with every document in Vault. DocID can be used to locate and obtain a document without having to perform the intermediate search steps normally required to search for and find a document.

A document's DocID can simply be obtained by calling the `VaultClient.Document.getDocID()` method. At a later time, the same document object can be retrieved by its DocID using the `VaultClient.Database.getDocumentByDocID()` method:

```
String docID = document.getDocID();
...
Document doc2 = db.getDocumentByDocID(docID);
```

Since documents stored in the Vault are always associated with a database, it is necessary to call the `VaultClient.Database.getDocumentByDocID()` method on the same database that contains the document, otherwise the document will not be found.

> **Note:** The DocID identifier will change if the data files corresponding to a document are reloaded or reindexed on the Vault server. For this reason, a DocID cannot be and should not be treated as an unchanging, fixed identifier to identify and locate a document.

DocumentEx class

If you know an account number, you can use `VaultClient.getAllDocumentEx()` to search documents associated to the account number.

```
VaultClient client=new VaultClient();
client.connect(host, port);
List<DocumentEx> docexlist=null;
docexlist=client.getAllDocumentEx(dbname, account, date-string,
```

```
maxresults);
client.shutdown();
```

Note: This class is used for performing less detailed but faster document searches. Because it only uses the INVLINK index for returning documents, it is a more efficient way of searching for documents. It is recommended that you use this method of returning the document list whenever possible.

Specifying the number of maximum document results

For the following Document search methods:

```
VaultClient.Account.searchForDocuments()
VaultClient.Database.searchForDocumentsWithoutAccount()
```

an extra "`maxResults`" parameter can also be specified that will limit the number of matching documents returned in each iteration to the specified value.

Documents and document information

The `DocumentInfo` class contains extended information about a document including custom attributes (if any) and sections (over and above standard document information such as page count, format, GUID info, invoice number, etc.). This is obtained by the `VaultClient.Document.getDocumentInfo()` method:

`DocumentInfo docinfo=client.getDocumentInfo(DocumentEx docex)`, or you can use the below method to get the `DocumentInfo` object, but if possible, it is recommended that you use this one.

```
DocumentInfo docInfo = document.getDocumentInfo();
```

However, the following properties of a document (that are also available directly through the document class methods) are only available after a call to `Document.getDocumentInfo()`:

- (Customer) Name
- (Customer) Address
- Invoice number
- Master GUID
- Instance GUID

```
// load the Document's info
document.getDocumentInfo();

// now we can access name, address, invoice number, and GUID info
if (document.getName().length() > 0) {
 ...
```

```
}
if (document.getAddress().length() > 0) {
 ...
}
if (document.getInvoice().length() > 0) {
 ...
}
if (document.getMasterGUID().length() > 0) {
 ...
}
if (document.getInstanceGUID().length() > 0) {
}
```

Note: Document properties above are also available through the `DocumentInfo` class:

```
// load the Document's info
DocumentInfo docInfo = document.getDocumentInfo();

if (docInfo.getName().length() > 0) {
 ...
}
if (docInfo.getAddress().length() > 0) {
 ...
}
if (docInfo.getInvoice().length() > 0) {
 ...
}
if (docInfo.getMasterGUID().length() > 0) {
 ...
}
if (docInfo.getInstanceGUID().length() > 0) {
 ...
}
```

Rendering documents

Documents can be rendered into a number of Output Formats depending on the type (for example, format) of the document (for example, AFP, Metacode, Postscript, etc.). The list of supported output formats for a document can be obtained as follows:

```
Set<OutputFormat> outputFormats = document.getOutputFormats();
```

Commonly used output formats include GIF (`OutputFormat.GIF`) and PDF (`OutputFormat.PDF`). The `OutputFormat` class allows, among other things, checking whether or not a specific output format:

- supports multiple pages: a contiguous range of pages can be rendered
- supports resolution: a specific resolution can be specified
- supports rotation: a rotation can be specified

Resolution width and rotation parameters (for output formats that support them) are specified by the `ResolutionWidth` and `Rotation` enumeration classes respectively, which enumerate valid values for both resolution width and rotation since arbitrary values are not supported. A Document can be rendered into an output format using one of the following methods:

- `VaultClient.renderDirect()`: for rendering a document through the main class by detail parameters.
- `VaultClient.renderDirectByGUID()`: for rendering a document through the main class by GUID string.
- `VaultClient.renderDirectByDocex()`: for rendering a document through the main class by `DocumentEx` object

You can use Document class to render a document as below, but it is not recommended:

- `VaultClient.Document.renderPage()`: for rendering a single page of a document
- `VaultClient.Document.renderPages()`: for rendering a contiguous range of pages of a Document.
- `VaultClient.Document.renderAllPages()`: for rendering all pages of a document.

For detailed information, please refer to the `Javadoc.zip` or `Javadoc.rar` documentation included with the Java API.

Here is a summary of the commonly used output formats and their properties:

Output Format	Supports multiple pages	Resolution	Rotation
GIF	No	Yes	Yes
HTML	No	No	No
PDF	Yes	No	No
RAW	No	Yes	Yes
PNG	No	Yes	Yes
TIFF	Yes	Yes	Yes
BMP	No	Yes	Yes
TIFF_G4	Yes	Yes	Yes

Example of rendering a single page:

```
Set<OutputFormat> outputFormats = document.getOutputFormats();

OutputFormat of = OutputFormat.GIF;
if (outputFormats.contains(of)) {
    FileOutputStream fos =
        new FileOutputStream("GIF-rendering" + of.getFileExtension());

    // render page #2 of this document to GIF at a resolution
    // width of 800 with no rotation
    document.renderPage(fos,
                        of,
                        ResolutionWidth.WIDTH_800, Rotation.NONE,
                        2, null);
    ...
}
```

Example of rendering a range of pages:

```
Set<OutputFormat> outputFormats = document.getOutputFormats();

OutputFormat of = OutputFormat.PDF;
if (outputFormats.contains(of) && of.supportsMultiplePages()) {
    FileOutputStream fos =
        new FileOutputStream("PDF-rendering" + of.getFileExtension());

    // render pages 5-10 of this document to PDF
    document.renderPages(fos,
                         of,
                         ResolutionWidth.NONE, Rotation.NONE,
                         5, 5, null);
    ...
}
```

If you know the parameters of dbname, account, date, file, pointer, it is recommended that you use:

```
client.renderDirect(db name, account, date, file, pointer, page-start-no,
 page-count, Resolutionwidth.NONE,Rotation.NONE,null,fos);
```

or you can use

```
//docex is an object of DocumentEx
client.renderDirectByDocEx(docex, page-start-no,
page-count,Resolutionwidth.NONE, Rotation.NONE,null,fos);
```

Optional rendering parameters can be supplied to the Document rendering methods described above using the `RenderOptions` class. Currently, the only optional rendering parameter supported is `enableBackground`:

```
enableBackground =
true: enables the background when rendering a page or range of pages to
 PDF.
false: disables the background when rendering to PDF.
```

Backgrounds are enabled by default when rendering to PDF, and in this case nothing extra has to be done as in the code snippet shown above.

To disable the background image when rendering to PDF:

```
Set<OutputFormat> outputFormats = document.getOutputFormats();

OutputFormat of = OutputFormat.PDF;
RenderOptions renderOptions = new RenderOptions();
renderOptions.setEnableBackground(false);
if (outputFormats.contains(of)) {
    FileOutputStream fos =
        new FileOutputStream("PDF-rendering" + of.getFileExtension());

    // render page #2 of this document to PDF with background
    // disabled
    document.renderPage(fos,
                        of,
                        ResolutionWidth.NONE, Rotation.NONE,
                        2,
                        renderOptions);
    ...
}
```

Note: Error handling and exception handling have been elided from all code snippets shown in this guide.

For a full example, please contact your Technical Support team.

Java API PDF security settings

You can set up PDF output security (user password, owner password, permissions, print, modify, copy, add/modify, annotation and forms) from `PROFILES.INI`. You can also set up the settings through the Java API.

For your settings take effect, set up `PDFEncryption=1` in the profile in profiles.ini on Vault Repository server side.

The main class for PDF security settings is `PDFSecurityParam` and the related class is `RenderOption`.

Note: The PDF security settings are not for Postscript to PDF by Ghostscript.

Set up Java API PDF security settings

1. Create an object of `RenderOption`:

```
    RenderOptions opt=new RenderOptions()
```

2. Enable or disable background settings. The default setting for background is enabled when creating an object of `RenderOption`. If you want to disable the background, you need to call below method:

```
opt.setEnableBackground(false);//disable the background
```

3. Set up PDF security parameters:
   a) Create an object of `PDFSecurityParam`:

```
PDFSecurityParam pdfsec=new PDFSecurityParam();
```

   b) Set parameters:

```
pdfsec.setMode(1)//Set up the output pdf security settings from
Java API;
    default is 0, from PROFILES.ini
    pdfsec.setUserPassword("userpassword");
    pdfsec.setOwnerPassword("ownerpassword");
    pdfsec.setPermissionPrint(true/false); //true means permission,
 false means no
    pdfsec.setPermissionModifyContents(true/false);//true = yes,
```

```
false =no
     pdfsec.setPermissionCopy(true/false);//true = yes, false = no

     pdfsec.setPermissionAddModify(true/false);//true = yes, false
 = no
```

c) Set up PDFSecurityParam object in `RenderOptions`:

```
opt.setPDFSecurity(pdfsec);
```

4. Use `RenderOptions` object as parameter in rendering methods. You can use:

```
VaultClient:renderDirect(),
VaultClient:renderDirectByGUID(),
VaultClient:renderDirectByDocEx()
```

Example

The following code example is just a small sample of what the entire code would look like. For the full code sample contact support.

Example:

```
VaultClient client=VaultClient();
FileOutputStream fos=new FileOutputStream("myfilename.pdf");
client.connect ("127.0.0.1",6003);
//other code lines
client.renderDirect(
                   dbname,
                   account, date, file, pointer,
                 OutputFormat::PDF,
                     0, 999999,
                     ResolutionWidth::none, Rotate::none,
                     opt,
                     fos;
//shutdown
```

Java API authentication

The Java API can support the Vault authentication model if it is enabled on the Vault server.

Use the `VaultClient.setUser()` / `VaultClient.setPassword()` methods to setup the authentication process.

Please refer to "Vault Java API javadoc" for more details.

Note: The password may be encrypted using the Vault password encryption model.

Vault Java API connection pool

In the Vault Java API, there is a new package `com.precisely.vault.pool` which is based on Apache common pool2. This can be used to share socket connections to Vault for standalone Java API application and Java web-application.

For details of the Vault Java API pool package please refer to "Vault Java API" documentation.

Interfaces and classes

1. Interfaces and classes
   a) `VaultPoolConfig` class

   Configuration class used to setup a shared connection pool:

   `String host="127.0.0.1"` : server host name

   `int port=6003` : server port number

   `boolean sslconnection` : SSL connection flag (is server SSL enabled). If TRUE, we need SSL settings. If FALSE there are no SSL settings. The default is FALSE.

   `SSLOption ssloption` : SSL parameters (keystorepath, keystoretype, keystorepassword, keypassword, truststorepath, truststoretype, truststorepassword);

   Note: You can use the `SSLUtil.createSSLOption()` method to create this parameter.

   `AuthOption authoption` : authentication settings if server authentication is enabled, including (user, password).

   `PoolOption pooloption` :

   VaultPool settings:

   `int maxPool` : maximum pool size;

`int startup` : startup pool size;

`long connectionCheckInterval` : time interval (seconds) for sending echo msg to server to keep pool connections alive;

b) `VaultPool` class

Main class of Vault connection pool.

c) `VaultPoolWapper` class

Wrapper class for `VaultPool` . This should be used for your pool implementation.

d) `VaultPoolException` class

An exception class.

e) Other classes

`VaultSocketFactory`, `VaultEvictionPolicy`, and some interfaces or classes from Apache common pool2 packages.

2. How to use the Vault connection pool
   a) Create a configuration object
   b) Create a global connection pool object of `VaultPool` class (throw `VaultPoolWrapper` class), based on the above configuration
   c) Create a client object to use the connection pool
   d) Use the client object to retrieve information from Vault server
   e) When the application is finished, close the connection pool before ending the application
3. Libraries and JAR files
   a) Vault Java API jar : `e2Vault-<version>.jar`
   b) Apache common pool : `commons-pool2-2.4.1.jar`
4. Using password encryption with the socket pool

   If you are encrypting passwords for SSL and authentication settings, please refer to *Vault Customization Guide* for details.
5. Example using a connection pool:

Example for full multi-threading

Please refer to the file `api-java\samples\Archive4jExamples\src\Test_VaultPool.java` in the common folder of the Windows install set or `Common-Vault.zip` for the non-Windows release packages.

```
//
//a brief example
```

```
//
import com.precisely.vault.pool.VaultPool;
import com.precisely.vault.pool.VaultPoolWrapper;
import com.precisely.vault.pool.VaultPoolException;
import com.precisely.vault.pool.VaultPoolConfig;
import com.precisely.vault.pool.PoolOption;
......
public class TestVaultPool {

    public static main(String[] argv){

        //
        //if want to encrypting passwords
        //
        //String keyfile=......;
        //if(keyfile!=null && !keyfile.isEmpty()) {
        //    try{
        //        VaultSecret.init(keyfile);
        //    }
        //    catch(Exception e){
       //       msg=String.format("failed to call VaultSecret.init(%s)",
 keyfile);
        //        System.out.println(msg);
        //        return;
        //    }
        //}

        VaultPool vaultpool=createPool();//vaultpool object is a global
 object and can be used inside multi-threading environment.

        try{
            VaultClient client=new VaultClient(vaultpool);
            Set<com.g1.e2.vault.VaultClient.Database>
vdblist=client.getDatabases();
            //other operations

             com.precisely.vault.service.DatabaseCommand cmd=new
DatabaseCommand(vaultpool);
             List<com.precisely.vault.model.Database>
cmddblist=cmd.databaseList();

             //other client classes, for example, UInfo, Command,
DatabaseInfoCommand, RenderCommand, ....
        }
        catch(Exception e) {
            System.out.println("an exception : "+e.getMessage());
            return;
        }
        finally{
           if(vaultpool!=null){//this is very important : the pool needs
 to be closed, otherwise socket resources will be leaked.
                vaultpool.close();
            }
```

```
        }
    }
    //
    //the input parameters can be from any source
    //
    private VualtPool createPool() throws Exception {
        VaultPoolConfig vpconfig=new VaultPoolConfig();
        vpconfig.setHost("127.0.0.1");//Vault render host name
        vpconfig.setPort(6003);//Vault render port number

        PoolOption po=new PoolOption();
        po.setMaxPool(5);//max pool size is 5
        po.setStartup(3);//startup pool size is 3
        po.setConnectionCheckInterval(30);//every 30 seconds, the pool
will send echo message to Vault render server to keep the connections
alive.
        vpconfig.setPooloption(po);

        //SSL (Secured Socket Layer) settings
        boolean ssl=false;//true;//if Vault render SSL enabled
        SSLOption sslo=null;
        if(ssl){
            String keystorepath=
            String keystoretype="jks";
            String keystorepw="changeit";
            String keypw="changeit";
            String truststorepath=
            String truststoretype="jks";
            String truststorepw="changeit";

            sslo=SSLUtil.createSSLOption(keystorepath, keystoretype,
keystorepw, keypw,
                truststorepath, truststoretype, truststorepw);
            vpconfig.setSslconnection(ssl);
            vpconfig.setSsloption(sslo);
        }
        //authentication settings if Vault render Authentication model
is enabled
        AuthOption ao=new AuthOption();
        String user="test";
        String password="test";
        ao.setUser(user);
        ao.setPassword(password);
        vpconfig.setAuthoption(ao);

        VaultPoolWrapper wrapper=new VaultPoolWrapper(vpconfig);

        return wrapper.getPool();
    }
}
```

4 - Vault SOAP Web Service

In this section

Configuring and setting up your web service environment

The following information lists the requirements necessary to setup your Vault Web Service environment. Web service `E2VaultWS` is cross platform (Windows and Linux.).

The C# sample shown here is only for Windows, all others are for all platforms.

Example setup using Windows and Tomcat

- OS : Windows
- Java: 1.7
- Webservice, Servlet container : Tomcat 8.0.15
- Development IDE : Netbeans IDE 8

Files included

- war file package : `E2VaultWS.war`
- configuration file : `E2VaultWS.xml`

Installing Web service

1. Install the war file

   To Install the war file package for Apache Tomcat 8.0.15, you can place the `E2VaultWS.war` file in the `TOMCAT-DIR/webapps` directory.

```
<context-param>
     <description>the host name of Vault Rendering Engine</description>

      <param-name>vaulthostname</param-name>
       <param-value>localhost</param-value>
</context-param>
<context-param>
      <description>the port number of Vault Rendering
```

```
Engine</description>
      <param-name>vaultport</param-name>
      <param-value>6003</param-value>
</context-param>
```

2. Set up the Vault host name and port number

   Open `TOMCAT-DIR/webapps/E2VaultWS/WEB-INF/web.xml`.

3. Set up parameters for SSL enabled Vault servers

   Open `TOMCAT-DIR/webapps/E2VaultWS/WEB-INF/web.xml`.

   - Add or modify the following lines:

```
<context-param>
        <description>SSL enable flag</description>
        <param-name>sslenabled</param-name>
        <param-value>0</param-value>
    </context-param>

    <context-param>
        <description>SSL truststore path</description>
        <param-name>ssltruststorepath</param-name>
        <param-value>C:/Program
Files/(x86)/Java/jdk1.8.0_25/jre/lib/security/cacerts</param-value>
    </context-param>
    <context-param>
        <description>SSL truststore password, the default is
"changeit"</description>
        <param-name>ssltruststorepassword</param-name>
        <param-value>changeit</param-value>
        <!-- <param-value>
{default;
bb8db7e33332eba1bac98650583941a607a
09219dd816f998a41b484ad08a1c6e8365f058d5bc71568c8e0c9830e6508}
</param-value> -->
    </context-param>
    <context-param>
        <description>SSL truststore type</description>
        <param-name>ssltruststoretype</param-name>
        <param-value>jks</param-value>
    </context-param>
    <context-param>
        <description>SSL keystore path</description>
        <param-name>sslkeystorepath</param-name>
        <param-value>C:/Program Files
(x86)/Java/jdk1.8.0_25/jre/lib/security/cacerts</param-value>
    </context-param>
    <context-param>
        <description>SSL keystore password, the default is
"changeit"</description>
        <param-name>sslkeystorepassword</param-name>
        <param-value>changeit</param-value>
```

```
        <!--
<param-value>{default;
bb8db7e33332eba1bac98650583941a607a
09219dd816f998a41b484ad08a1c6e8365f058d5bc71568c8e0c9830e6508}
</param-value>
 -->
    </context-param>
    <context-param>
        <description>SSL key password, the default is
"changeit"</description>
        <param-name>sslkeypassword</param-name>
        <param-value>changeit</param-value>
        <!--
<param-value>{default;
bb8db7e33332eba1bac98650583941a607a
09219dd816f998a41b484ad08a1c6e8365f058d5bc71568c8e0c9830e6508}
</param-value>
 -->
    </context-param>
    <context-param>
        <description>SSL keystore type</description>
        <param-name>sslkeystoretype</param-name>
        <param-value>jks</param-value>
    </context-param>
```

4. Other parameter settings:
   a) Set up authentication with Vault servers that have authentication enabled.
      - Open `TOMCAT-DIR/webapps/E2VaultWS/WEB-INF/web.xml`.
      - Add or modify the following lines:

        Login user name:

```
<context-param>
     <description>login user name</description>
     <param-name>serviceuser</param-name>
     <param-value>test</param-value>
</context-param>
```

        Login password:

```
<context-param>
     <description>login password</description>
     <param-name>servicepassword</param-name>
     <param-value>test</param-value>
</context-param>
```

        Note: Passwords (`trustorepassword`, `sslkeypassword` and `sslkeystorepassword`) may be encrypted using the Vault password encryption model.

b) Set up connection-pool parameters:

- Open `TOMCAT-DIR/webapps/E2VaultWS/WEB-INF/web.xml`.
- Add or modify the following lines:

```
<context-param>
        <description>Vault connection pool enable flag, 1=enabled,
0=disabled</description>
        <param-name>poolenabled</param-name>
        <param-value>0</param-value>
    </context-param>
    <context-param>
        <description>Vault connection pool startup number</description>
        <param-name>poolstartup</param-name>
        <param-value>5</param-value>
    </context-param>
    <context-param>
        <description>Vault connection pool max number</description>
        <param-name>poolmax</param-name>
        <param-value>5</param-value>
    </context-param>
    <context-param>
        <description>Vault connection pool check interval</description>
        <param-name>poolconnectioncheckinterval</param-name>
        <param-value>30</param-value>
    </context-param>
```

5. Test the installation
   a) After installing files (`E2VaultWS.war` and `E2VaultWS.xml`):
   b) Start Tomcat and check the log or screen messages to see if there are any errors.
   c) If there are no any errors, you can check the `E2VaultWS` web service api.
6. Open a web browser.
7. Enter: "http://localhost:8080/E2VaultWS" and you will see the following message:
   "Hello Web Service World!"
8. Enter: "http://localhost:8080/E2VaultWS/E2VaultWS" and you will see the following message:

```
-------------------------------------------------------------------------------------------
| Endpoint                                    | Information                                 |
-------------------------------------------------------------------------------------------
| Service Name:                               | Address:                                    |
| {http://ws.vault.e2.pb.com/}E2VaultWS       | http://localhost:8080/E2VaultWS/E2VaultWS   |
 -----------------------------------------------------------------------------------------
| Port Name:                                  | WSDL:                                       |
| {http://ws.vault.e2.pb.com/}E2VaultWSPort   | http://localhost:8080/E2VaultWS/E2VaultWS?wsdl  |
 -----------------------------------------------------------------------------------------
|                                             | Implementation class:                       |
|                                             | com.pb.e2.vault.ws.E2VaultWSImpl            |
-------------------------------------------------------------------------------------------
```

9. Click the "WSDL: http://localhost:8080/E2VaultWS/E2VaultWS?wsdl" link and you will see the WSDL XML content.
   Your Web Service setup is complete.

Programming the SOAP Web Service

The following information will help you to program and customize your Vault Web Service environment.

Programming notes

- `E2VaultWS` is the Vault web service.
- `E2VaultWS.war` is the war file of Vault Web service API.
- `com.precisely.e2.vault.ws` is the namespace of Vault web service API.
- The name of the web service is case sensitive.

Data types

Data types across different languages can be used in different computer languages.

1. . `int` : integer type.
2. . `String` : can be across different languages, such as Java, C#.
3. . `List` : collection type, can be used in different languages.

Web service data types

`WsIndex`

The index information data structure.

Keywords and parameters	Description
`int no`	index no
`string name`	index name
`string fields`	index fields
`string`	index
`string desc`	index description
`WsIndexAttributes` attributes	index attributes

WsSearchParam

The index search input data structure.

Keywords and parameters	Description
`string dbname`	database name
`string index`	index string, can be index no or index name (for example, "1" or "account").
`string account`	account number associate with the index if applicable.
`string date`	document date string, if applicable.
`string guidstring`	GUID string when searching GUID index.
`int maxresults`	max number of results returned from SERVER.
`string cursorkey`	starting string in the index. Use the last value of `WsSearchResult.WsSearchResultData.matched` to set up CURSORKEY for next search.
`string searchkey`	the string that will be search in the index.
`WsSearchMode searchmode`	define which index will be searched (for example, "generic", "invlink", "guid", iguid").

WsSearchResult

Search result data structure.

Keywords and parameters	Description
`WsIndexType indextype`	return index type of the index that is searched
`WsSearchResultTitle title`	customized titles for the index fields
`WsSearchResultData data`	search result data object
`int moredata`	any more data, 1=more-data, 0=no-more-data. Use CURSORKEY in `WsSearchParam` to start searching again.

WsSearchResultTitle

Search result index fields title data structure.

Keywords and parameters	Description
`string matched`	title name of matched string
`string account`	title name of account
`string date`	title name of document date
`string format`	title name of document format
`string file`	title name of document file
`string pointer`	title name of document offset
`WsKeyValue extended`	other title names of document info

WsSearchResultData

Index search result data structure.

Keywords and parameters	Description
`string matched`	matched search result (for example, ACCOUNT number when searching account index)
`string account`	account number
`string date`	document date string
`string format`	document format
`string file`	document file name
`string pointer`	document offset in the file
`int pages`	total page number of the document

Keywords and parameters	Description
`string name`	customer name related to the document
`string address`	customer address related to the document.
`WsKeyValue extended`	other key/value information defined for the document

`WsDocumentAttributes`

Document attributes data structure.

Keywords and parameters	Description
`string dbname`	database name
`string account`	account number
`string date`	document date string
`string format`	document format
`string file`	document file name
`string pointer`	document offset in the file
`int pages`	total document page number
`int modes`	document mode combine
`string name`	customer name related to the document
`string address`	customer address related to the document
`string invoice`	customer invoice number related to the document
`string section`	report section information
`string type`	data type when the document format is COLLECTION

Keywords and parameters	Description
`WsOutputFormat outputformats`	possible output formats for this document
`WsKeyValue customerfields`	other customer defined fields

`WsKeyValue`

Key/Value pair data structure.

Keywords and parameters	Description
`string name`	key name of an item
`string val`	value of an item

`WsDocumentParam`

Document input parameter when using the web service method `documentList()`.

Keywords and parameters	Description
`string dbname`	database name
`string account`	account number
`string date`	document date string
`string cursorkey`	the start string when getting next document list
`int maxresults`	max number of results

`WsDocumentResult`

Document result data structure.

Keywords and parameters	Description
`WsDocumentData docdata`	document data structure

Keywords and parameters	Description
`int moredata`	any more data, 1=has-more-data, 0=no-more-data

`WsDocumentData`

Document data structure.

Keywords and parameters	Description
`string account`	account number
`string date`	document date string
`string format`	document format
`string file`	document file name
`string pointer`	document offset in the file
`int pages`	page number of the document

`WsDocumentExResult`

`documentEx` result data structure.

Keywords and parameters	Description
`WsDocumentExData docdata`	documentEx data structure `int moredata` flag of indicating any more data (1=has-more-data, 0=no-more-data, when used in `DocumentExListAll()` function, this parameter is not meaningful).

`WsDocumentExData`

`documentEx` data structure.

Keywords and parameters	Description
`string dbname`	database name
`string account`	account number
`string date`	document date string
`string file`	document file name
`string pointer`	document offset in the file

`WsReprintParam`

Input parameter for document reprint command.

Keywords and parameters	Description
`string dbname`	database name
`string account`	account number
`string date`	document date string
`string format`	document format
`string file`	document file name
`string pointer`	document offset in the file
`int pagefrom`	the from page number for the reprint command
`int pagecount`	total page number for the reprint command
`WsReprintOptions options`	other document reprint info

`WsReprintOptions`

Document reprint option data structure.

Keywords and parameters	Description
`int none`	for future use (placeholder)

`WsReprintResult`

Data structure of document reprint command.

Keywords and parameters	Description
`int result`	result of document reprint command

`WsPDFSecurityParam`

PDF output security setting parameters.

Keywords and parameters	Description
`int pdfsecuritymode`	PDF output security mode, 1=enabled, the render server uses the parameters from the API; 0=disabled, the render server uses the parameters from the profile. If you want to make PDF security effective, set up `PDFEncryption=1` in the profile.
`string userpassword`	Secured PDF user password string
`string ownerpassword`	Secured PDF owner password string
`int permission_print`	Print permission, 1=enable, 0=disable
`int permission_copy`	Copy permission, 1=enable, 0=disable
`int permission_modifycontents`	PDF contents modification permission, 1=enable, 0=disable
`int permission_modifyannotationform`	Permission of adding or modifying PDF annotation and interactive form, 1=enable, 0=disable

`WsRenderParam`

Input parameter of rendering document page data.

Keywords and parameters	Description
`string dbname`	database name
`string account`	account number
`string date`	document data string
`string format`	document format
`string file`	document file name
`string pointer`	document offset in file
`int pagestart`	starting page no.
`int pagecount`	total page number
`WsOutputFormat outputformat`	output format for rendering
`int background`	background flag, 1=need background, 0=no-background
`WsOrientation orientation`	orientation parameter
`WsResolution resolution`	resolution parameter
`int cpix`	X value when rendering document as text
`int cpiy`	Y value when rendering document as text
`string textencoding`	text encoding
`int mark`	mark
`string findtext`	text string that will be found in the pages
`int findcolor`	hightlight color of found text
`int findcase`	case sensitive flag for finding text
`WsPDFSecurityParam pdfsecurity`	PDF security setting parameters

WsRenderParamBase

Basic input parameters of rendering document page data.

Keywords and parameters	Description
`string dbname`	database name
`int pagestart`	starting page no
`int pagecount`	total page number
`WsOutputFormat`	output format for rendering
`int background`	background flag, 1=need background, 0=no-background
`WsOrientation`	orientation parameter
`WsResolution`	resolution parameter
`int cpix`	X value when rendering document as text
`int cpiy`	Y value when rendering document as text
`string textencoding`	text encoding
`int mark`	mark
`string findtext`	text string found in the pages
`int findcolor`	hightlight color of found text
`int findcase`	case sensitive flag for finding text
`WsPDFSecurityParam pdfsecurity`	PDF security setting parameters

WsRenderPages

Document page data structure.

Keywords and parameters	Description
`WsOutputFormat outputformat`	rendering output format
`int pagestart`	starting page no.
`int pagecount`	total page number
`base64Binary pagedatabytes`	document page data bytes

`WsIndexAttributes`

Index attributes data structure.

Note: 1=set flag, 0=reset flag.

Keywords and parameters	Description
`int prefixaccount`	if an index has `prefixaccount`
`int multiplevalues`	if an index has multiple values
`int accountmustbeselected`	inside an index if account must be selected
`int linkbydocumentrecord`	if an index is linked by document record
`int linkbycustomerrecord`	if an index is linked by customer record
`int displaypage`	if display page
`int displaydocument`	if display document
`int displaynewestdocument`	if display the newest document
`int rotatestrings`	if `rotatestrings`
`int searchbackwards`	if an index is search backwards
`int invisibletouser`	if an index is invisible to users

Keywords and parameters	Description
`int updateoncusomter`	if an index has an update on customer
`int leaveonunindex`	if leave on `unindex`
`int jumponsinglematch`	if jump on single match when searching
`int webaccountindex`	if web account index
`int weblinkindex`	if web link index

`WsOutputFormat`

Output format of enumeration data structure.

Base type: String

Values:

{"GIF", "HTML", "PDF", "RAW", "PNG", "TIFF", "BUMP", "STREAM", "TEXT", "COLLECT", "COMMENTS", "DOCINFO", "TIFF_G4", "XML", "DECODE", "REPRINT", }

`WsOrientation`

Orientation values of enumeration data structure.

Base type: int

Values: {0, 90, 180, 270}

`WsResolution`

Resolution values of enumeration data structure.

Base type: int

Values: {512, 640, 800, 1024, 1280, 1600}

`WsIndexType`

Base type: String

Values: {"none", "cusomter_record", "document_record"}

`WsSearchMode`

Index search mode values of enumeration data structure.

Base type : String

Values : {"generic", "invlink", "guid", iguid"}

`WsVaultFault`

E2VaultWS exception data structure.

String message: exception message.

Parameters:

int id: error ID.

string funname: the function name that throws the exception.

Web service interfaces

`public String getVersion4Lib()`

Get version number of Vault Java API that `E2VaultWS` is based on.

Return parameters: version of Java API.

`public String getVersion4Ws()`

Get version number of `E2VaultWS`.

Return parameters: version of `E2VaultWS` web service API.

`public String getServerName()`

Get name of server to which `E2VaultWS` is connected.

Return parameters: server name.

`public String getServerVersion()`

Get version number of server to which `E2VaultWS` is connected.

Return parameters: server version.

`public List<com.precisely.e2.vault.ws.WsDB> databaseList()`

Retrieve the database list of Vault server.

Return parameters: the collection of database object of WsDB.

`public List<com.precisely.e2.vault.ws.WsIndex> databaseInfo(String dbname)`

Retrieve index list of a specific database defined by DBNAME.

Input parameters: String `dbname` - defined database name.

Return parameters: collection of index object of `WsIndex`.

```
public com.precisely.e2.vault.ws.WsIndex indexInfo(String dbname, String
index)
```

Retrieve index information for specific INDEX and DBNAME.

Input parameters:

- String dbname: defined database name.
- String index: defined index string (can be number or name. For example, "1" or "account")

Return parameters: The object of `WsIndex`.

```
public com.precisely.e2.vault.ws.WsSearchResult
databaseSearch(com.precisely.e2.vault.ws.WsSearchParam searchparam)
```

Search an index.

Input parameters: `WsSearchParam searchparam` - define search input parameters, such as dbname, index, etc.

Return parameters: object of `WsSearchResult` including search result data.

```
public com.precisely.e2.vault.ws.WsDocumentAttributes documentInfo(String
dbname, String account, String date, String format, String file, String
pointer)
```

Retrieve information for a document defined in a parameter set.

Input parameters:

- `String dbname`: database name.
- `String account`: account number for document.
- `String date`: date string for document.
- `String format`: format string for document.
- `String file`: file name for document.
- `String pointer`: file offset for document in a document file defined above.

Return parameters: the object of `WsDocumentAttributes`.

```
public com.precisely.e2.vault.ws.WsDocumentResult
documentList(com.precisely.e2.vault.ws.WsDocumentParam documentparam)
```

Retrieve document list.

Input parameters: `WsDocumentParam documentparam` - define parameters for a document.

Return parameters: the object of `WsDocumentResult`.

```
public com.precisely.e2.vault.ws.WsDocumentExResult documentExListAll(String
dbname, String account, String date, int searchwindowsize)
```

Retrieve all of `documentEx` list.

Input parameters:

- `String dbname` - define database name.
- `String account`: define account number associated with `documentex`.
- `String date`: define date string associated with `documentex`, can be partially matched.
- `int searchwindowsize`: define the window size for one search.

Return parameters: the object of `WsDocumentExResult`.

```
public com.precisely.e2.vault.ws.WsReprintResult
documentReprint(com.precisely.e2.vault.ws.WsReprintParam param)
```

Send a command of document REPRINT to the server.

Input parameters: `WsReprintParam param` - set up document information.

Return parameters: the object of `WsReprintResult`.

```
public com.precisely.e2.vault.ws.WsRenderPages
renderTransform(com.precisely.e2.vault.ws.WsRenderParam param)
```

Render document page data.

Input parameters: `WsRenderParam param` - define parameters for rendering document data.

Return parameters: the object of `WsRenderPages`.

```
public com.precisely.e2.vault.ws.WsRenderPages renderTransformByGUID(String
guidstr, com.precisely.e2.vault.ws.WsRenderParamBase param)
```

Render document page data through GUID string (from GUID index).

Input parameters:

- `String guidstr` - define GUID string from GUID index, `WsRenderParam`
- `param`: define parameters for rendering document data.

Return parameters: the object of `WsRenderPages`.

```
public com.precisely.e2.vault.ws.WsRenderPages renderTransformByIGUID(String
iguidstr, com.precisely.e2.vault.ws.WsRenderParamBase param)
```

Render document page data through IGUID string (from IGUID index).

Input parameters:

- `String iguidstr`: define GUID string from IGUID index, `WsRenderParam`
- `param`: define parameters for rendering document data.

Return parameters: the object of `WsRenderPages`.

Web service names

- Service name: `E2VaultWS`
- port name: `E2VaultWSPort`
- Binding name: `E2VaultWSPortBinding`

Web service interface calling procedures

1. Set up the parameters for Vault servers:
   - Vault host name
   - Vault port number
   - SSL parameters, if Vault servers are SSL enabled
2. Call the below methods to get version and server information.
   - `getVersion4Lib();`
   - `getVersion4Ws()`
   - `getServerName()`
   - `getServerVersion()`
3. Call `databaseList()` to get database list if you don't know the databases on server side.
4. Call `databaseInfo()` to get index list for a specific database.
5. If you don't know an ACCOUNT number, you can call `databaseSearch()` to search an index, such as ACCOUNT index, NAME index.
   You can get ACCOUNT list or DOCUMENT list.
6. Call the heavy-weight function of `documentList()` to get all documentex list, or call the light-weight function of `documentExListAll()` to get all of documentex list.
7. Call `renderTransform()` / `renderTransformByGUID()` / `renderTransformByIGUID()` to get document pages data.

Create web service client of Java console application with NetBeans 8.0.2 IDE

1. Download, install and run Tomcat 8.0.15 on local machine.
2. Refer related documents to install E2VaultWS web service into Tomcat 8.0.15.
3. Download, install and run Netbeans 8.0.2.
4. Create a java console application.
   a) From Netbeans 8.0.2, choose **File New Project**, a dialog of "New Project" displays.
   b) Choose **Java** from **Categories**.
   c) Choose **Java Application** from **Projects**.
   d) Choose **Next**.
   e) Input **Project Name** as "e2vaultwsconsole", **Project Location** as "c:\e2vaultwsconsole", leave others as default.

f) Choose **Finish**.
Then IDE will create a java console application in the directory of `c:\e2vaultwsconsole` and create a main class named `Main.java`.

5. import WSDL interfaces.
   a) Right click the project of **e2vaultwsconsole** from **Prjects List**.
   b) Choose **New Web Service Client**, a dialog of "WSDL and Client Location" displays.
   c) Choose **WSDL URL** and input "http://localhost:8080/E2VaultWS/E2VaultWS?wsdl".
   d) Input package name : e2vaultwsconsole.
   e) Choose **Finish**.
   Then IDE will create all web service stub files the client application needs.
6. Write the code to call the `E2VaultWS` web service interfaces.
   The directory for the Java code sample is:
   `webserivces/E2VaultWS/samples/console-JAVA/`.

   For the code sample which contains the lines of Main class and calling `E2VaultWS` web service interfaces, please refer to the API library along with this sample code from your Vault installation CD.

Create a web service client C# console with Visual Studio 2008

1. Download, install and run Tomcat 8.0.15 on the local machine.
2. Refer related documents to install `E2VaultWS` web service into Tomcat 8.0.15.
3. Download, install, and run Visual Studio 2010.
4. Create a C# console application named "e2vaultwsconsole".
5. Import WSDL interfaces.
   a) Right click the project of **e2vaultwsconsole** from **Solution Explore**.
   b) Choose **Add web reference**, a dialog of **Add Web Reference** displays.
   c) Input URL: "http://localhost:8080/E2VaultWS/E2VaultWS?wsdl".
   d) Click **Go**, E2VaultWS web service methods displays.
   e) Input **Web Reference Name** as "e2vaultwsconsole".
   f) Click **Add reference**.
   Then IDE will create all web service stub files the client application needs.
6. Write the code to call E2VaultWS web service interfaces.
   The directory for the C# code sample is: webserivces/E2VaultWS/samples/console-CS/.

   For the code sample which contains the lines of Main class and calling E2VaultWS web service interfaces, please refer to the API library along with this sample code from your Vault installation CD.

Note: When you use `e2vaultweb` service for the first time, the system will take some time to load the whole package. how long it will take depends on the machine and system configuration.

PDF output security settings

When rendering PDF, you can set up secured PDF output with user password, owner password and permissions (copy, print, modification, and adding/modifying annotation and interactive form).

1. Set `PDFEncryption=1` in `profile.ini`

```
[MyProfile]PDFEncryption=1
PDFUserPassword=11111
PDFOwnerPassword=aaaaa
PDFAllowPrint=0
PDFAllowModifyContents=0
PDFAllowCopy=0
PDFAllowModifyAnnot_Form=0
```

2. Set `pdfsecuritymode=1` in data type of `WsPDFSecurityParam`

   For Java-based usage:

```
WsPDFSecurityParam pdfsec=new WsPDFSecurityParam()
if pdfsec.pdfsecuritymode=1, PDF security parameters will be from this
 API side;
if pdfsec.pdfsecuritymode=0, then PDF security parameters will be from
 the profile;
```

3. Set up other parameters:

```
pdfsec.userpassword="userpassword";
pdfsec.ownerpassword="ownerpassword";
pdfsec.permissionCopy=1;//1=enable, 0=disable
pdfsec.permissionPrint=1;//1=enable, 0=disable
pdfsec.permissionModifyContents=1;//1=enable, 0=disable
pdfsec.permissionAnnotationform=1;//1=enable, 0=disable
```

4. Set up `WsPDFSecurtyParam` in rendering parameter:

```
WsRenderParam rparam=new WsRenderParam();
...
rparam.setOutputformat(WsOutputFormat.PDF);
rparam.setPdfsecurity(pdfsec);
...
```

5. Use interfaces of `renderTransform()`, `renderTransformByGUID()` or `renderTransformByIGUID()` to send the PDF rendering request to render server.

5 - Communicating directly to the Rendering engine

Communicating directly with the Rendering Engine via the Vault UINFO protocol is a method of “last resort”, should none of the programming-language specific APIs be applicable. This information may also be useful for those who wish to review or analyze log files from the Vault services such as e2renderd, e2serverd and e2routerd.

In this section

Wire API Function Reference

Symbol Key

The following symbols are used in the function descriptions below to indicate when the parameters and results are present in a message.

Request

! mandatory

? optional

* special

Result Values

! field is always present

- field is present on error

+ field is present on success

^ field is present conditionally

Notice Functions

notice.cacheflush

Tells clients to reset their cache. It is sent by the server to indicate a change in the loaded data. Render does not send this notification by default.

Request	`request.function`	`notice.cacheflush`
Result	none	
Data	none	

Server Functions

server.echo

Tests whether the server is active.

Request	! request.function	`server.echo`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	none	

server.connection

Get connection status information

Request	`! request.function`	`server.connection`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.requests`	number of requests made on this connection
	`+ result.read`	number of bytes read by the server on this connection
	`+ result.written`	number of bytes written by the server on this connection
	`+ result.duration`	number of seconds this connection has been open
Data	none, data returned in result message	

server.info

Get server information

Request		
	`! request.function`	`server.status`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.program`	name of server program
	`+ result.version`	version of server program
Data	none, data returned in result message	

storage.docdata

Fetches report list and properties of a document

Request		
	`! request.function`	`storage.docdata`
	`! request.file`	`<filename>`
	`! request.offset`	`<document record offset>`
Result		

	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not

Data	4 column hybrid table / key-value pair list
	`<key1> <value1a> <value1b> <value1c>`
	`<key2> <value2a> <value2b> <value2c>`	

storage.docpage

Fetches raw internal data for a document page or pages

Request		
	`! request.function`	`storage.docpage`
	`! request.file`	`<filename>`
	`! request.offset`	`<document record offset>`
	`! request.page`	`<page number>`
	`? request.page`	`<last page number>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	n x 1 table for binary objects `<pages data>`	

storage.docarray

Fetches the offsets of a document's raw pages in the page data file.

Request		
	`! request.function`	`storage.docarray`
	`! request.file`	`<filename>`
	`! request.offset`	`<document record offset>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`result.message`	error message if status not zero
Data	1x1 table for binary object <document page array>	

storage.profiledata

Fetches the key value pairs from a specified profile.

Request		
	`! request.function`	`storage.profiledata`
	`! request.profile`	`<profilename>`
Result		

	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	two column list of key-value pairs
	`<key1> <value1>`
	`<key2> <value2>`	

Storage Functions

storage.docdata

Fetches report list and properties of a document

Request	`! request.function`	`storage.docdata`
	`! request.file`	`<filename>`
	`! request.offset`	`<document record offset>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not

Data	4 column hybrid table / key-value pair list
	`<key1> <value1a> <value1b> <value1c>`
	`<key2> <value2a> <value2b> <value2c>`	

storage.docpage

Fetches raw internal data for a document page or pages

Request		
	`! request.function`	`storage.docpage`
	`! request.file`	`<filename>`

	! request.offset	<document record offset>
	! request.page	<page number>
	? request.page	<last page number>
Result		
	! result.status	0 for success, otherwise error code
	! result.blocks	number of subsequent message blocks, 1 if successful, 0 if not
	- result.message	error message if status not zero
Data	n x 1 table for binary objects <pages data>	

storage.docarray

Fetches the offsets of a document's raw pages in the page data file.

Request		
	! request.function	storage.docarray
	! request.file	<filename>
	! request.offset	<document record offset>
Result		
	! result.status	0 for success, otherwise error code
	! result.blocks	number of subsequent message blocks, 1 if successful, 0 if not
	result.message	error message if status not zero

Data	1x1 table for binary object <document page array>	

storage.profiledata

Fetches the key value pairs from a specified profile.

Request		
	`! request.function`	`storage.profiledata`
	`! request.profile`	`<profilename>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	two column list of key-value pairs
	`<key1> <value1>`
	`<key2> <value2>`	

storage.profilelist

Lists the names of all available profiles known to the server.

Request		
	`request.function`	`storage.profilelist`
Result		

	`!result.status`	0 for success, otherwise error code
	`!result.blocks -`	number of subsequent message blocks, 1 if successful, 0 if not
	`result.message`	error message if status not zero

Data	1 column table of values
	`<profilename1>`
	`<profilename2>`	

storage.filedata

Retrieves the metadata associated with a compressed file.

Request		
	`! request.function`	`storage.filedata`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.profile`	profile name associated with file
	`+ result.resource`	resource set name associated with file
	`+ result.pagedatafile`	the full (local) path to the page data file
Data	none, data returned in result message	

storage.filepage

Fetches a raw page from a compressed file.

Request		
	`! request.function`	`storage.filepage`
	`! request.file`	`<filename>`
	`! request.offset`	`<page offset>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<page data>`	

storage.bulkfilepages

Fetches all or specific number of raw pages from the same compressed block of compressed file

Request		
	`! request.function`	`storage.bulkfilepages`
	`! request.file`	`<filename>`
	`! request.page`	`page number`

	`? request.lastpage`	last page number, when ommited it is assumed that last page is request.page
	`! request.docarray.binary`	binary object `<document page array>` returned from docarray request
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	n x 1 table for binary objects `<pages data>`	

storage.headerinfo

Determines if a given compressed file exists and returns the size of its Postscript file header.

Request		
	`! request.function`	`storage.headerinfo`
	`! request.file`	`<filename>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.exists`	1 if file exists, 0 if it does not
	`^ result.size`	file size, if file exists

Data	none, data returned in result message	

storage.headerdata

Fetches a block of data from a compressed file's Postscript header.

Request		
	`! request.function`	`storage.headerdata`
	`! request.file`	`<filename>`
	`! request.offset`	`<offset as pointer>`
	`! request.size`	`<size of block in bytes>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<block data>`	

storage.list

Lists the names of compressed files.

Request		
	`! request.function`	`storage.list`

	`? request.file`	`<file name prefix pattern>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	3 column table
	`<filename1> <format1> <modes1>`
	`<filename2> <format2> <modes2>`	

storage.online

Lists the names of compressed files.

Request		
	`! request.function`	`storage.online`
	`! request.file`	`<filename>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
	`+ result.online`	0 if offline, 1 if online

Data	none	Data returned in the result message. Note this is currently supported on Windows platforms only.

storage.recall

Requests that an offline compressed file be placed online.

Request		
	`! request.function`	`storage.recall`
	`! request.file`	`<filename>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, always 0
	`- result.message`	error message if status not zero
Data	none	Data returned in the result message. Note that this is currently supported on Windows platforms only. This function returns before recall operation is completed.

Mirror Functions

mirror.find

Used internally by mirror functions to list ranges of compressed files.

Request		
	`! request.function`	`mirror.find`
	`! request.directory`	`<symbolic name of server directory>`
	`? request.prefix`	`<file prefix>`
	`? request.suffix`	`<file suffix>`
	`? request.start`	`<filename to resume search after>`
	`? request.max`	`<maximum number of results to return>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	1 column table
	`<filename1>`
	`<filename2>`	

mirror.info

Determines the status of a file to be mirrored.

Request		
	`! request.function`	`mirror.info`
	`! request.directory`	`<symbolic name of server directory>`
	`! request.file`	`<compressed filename>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`^ result.exists ^`	1 if file exists
	`^ result.date`	file timestamp if file exists
	`^ result.size`	file size if file exists
Data	none, data returned in result message	

mirror.data

Fetches a block of data from a file being mirrored.

Request		
	`! request.function`	`mirror.data`
	`! request.directory`	`<symbolic name of server directory>`
	`! request.file`	`<filename with extension>`
	`! request.offset`	`<offset of block>`
	`! request.size`	`<size of block>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<block data>`	

Resource Functions

resource.data

Fetches a block of data from a resource.

Request		
	`! request.function`	`resource.data`
	`? request.path`	`<<the subdirectory to search>>`
	`! request.file`	`<name of the resource file to check>`
	`! request.offset`	`<offset of block>`
	`! request.size`	`<size of block>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<block data>`	

resource.list

Used to recursively enumerate upstream resource files.

Request		
	`! request.function`	`resource.list`
	`? request.path`	`<the subdirectory to search>`
	`! request.pattern`	`<file search pattern>`
	`? request.recursive`	`0/1 return results from subdirectories`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
	`^ result.exists`	1 if file exists
	`^ result.date`	file timestamp if file exists
	`^ result.size`	file size if file exists

Data	3 column table
	if the entry is a directory, the size is -1 and the date is empty
	`filename1> <isdir1> <size1> <date1>`
	`<filename2> <isdir1> <size2> <date2>`
	Note that *isdir* is 0 or 1 size and date are in hex format	

Statistics Functions

stat.list

Fetches statistics about function execution times.

Request		
	`! request.function`	`stat.list`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	7 column table
	`Function Errors Cached Uncached Min Max Avg`
	`<function1> <errors1> <cached1> <uncached1> <min1> <max1> <avg1>`
	`<function2> <errors2> <cached2> <uncached1> <min2> <max2> <avg2>`	

Compressed File Functions

compressed.list

Lists the names of compressed files.

Request		
	`! request.function`	`compressed.list`
	`? request.file`	`<file name prefix pattern>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	3 column table
	`<filename1> <format1> <modes1>`
	`<filename2> <format2> <modes2>`	

compressed.open

Opens a compressed file for access.

Request		
	`! request.function`	`compressed.open`
	`! request.file`	`<file name to open>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`+ result.compressed`	the handle number associated with this open file
	`- result.message`	error message if status not zero
Data	none, data returned in result message.	

compressed.close

Closes an open compressed file handle.

Request		
	`! request.function`	`compressed.close`
	`! request.compressed`	`<file handle to close>`
Result		
	`! result.status`	0 for success, otherwise error code

	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	none, data returned in result message.	

compressed.status

Gets the background scanning status for a compressed file.

Request		
	`! request.function`	`compressed.status`
	`! request.compressed`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	a 7 column table
	`<handle1> <filename1> <pagecount1> <supportedoutputmodes1> <scancomplete1> <scanerrorcode1> <scanerrormessage1>`
	`<handle2> <filename2> <pagecount2> <supportedoutputmodes2> <scancomplete2> <scanerrorcode2> <scanerrormessage2>`	

compressed.page

Fetches raw page data from a compressed file.

Request		
	`! request.function`	`compressed.page`
	`! request.compressed`	`<file handle>`
	`! request.page`	`<page number to return>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<page content>`	

compressed.docdata

Gets the simulated document data for a compressed file.

Request		
	`! request.function`	`compressed.docdata`
	`! request.compressed`	`<file handle>`

Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	4 column hybrid table / key-value pair list
	`<key1> <value1a> <value1b> <value1c>`
	`<key2> <value2a> <value2b> <value2c>`	

compressed.filedata

Gets the compressed file information associated with an open compressed file.

Request		
	`! request.function`	`compressed.filedata`
	`! request.compressed`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`+ result.profile`	profile name
	`+ result.resource`	resource set name
	`+ result.pagedatafile`	the full (local) path to the page data file
	`- result.message`	error message if status not zero

Data	none, data returned in result message	

compressed.profiledata

Gets the profile settings associated with an open compressed file.

Request		
	`! request.function`	`compressed.profiledata`
	`! request.compressed`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	two column list of key-value pairs
	`<key1><value1>`
	`<key2><value2>`	

compressed.headerinfo

Determines if a given compressed file exists and returns the size of its Postscript file header.

Request		
	`! request.function`	`compressed.headerinfo`
	`! request.compressed`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.exists`	1 if file exists, 0 if it does not
	`^ result.size`	file size, if file exists
Data	none, data returned in result message	

compressed.headerdata

Fetches a block of data from a compressed file's Postscript header.

Request		
	`! request.function`	`compressed.headerdata`
	`! request.compressed`	`<file handle>`
	`! request.offset`	`<offset as pointer>`
	`! request.size`	`<size of block in bytes>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
Data	1x1 table for binary object `<block data>`	

Local File Functions

local.open

Opens a local file for access.

Request		
	`! request.function`	`local.open`
	`! request.file`	`<file name to open>`
	`! request.blocks`	1, the number of subsequent blocks

Parameters	two column list of key-value profile pairs
	`<key1><value1>`
	`<key2><value2>`	

Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.local`	the handle number associated with this open file
Data	none, data returned in result message	

local.close

Closes an open local file handle.

Request		
	`! request.function`	`local.close`
	`! request.local`	`<file handle to close>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	none, data returned in result message	

local.status

Gets the background scanning status for a local file.

Request		
	`! request.function`	`compressed.status`
	`? request.local`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not

	- result.message	error message if status not zero

Data	a 7 column table
	`<handle1> <filename1> <pagecount1> <supportedoutputmodes1>`
	`<scancomplete1> <scanerrorcode1> <scanerrormessage1>`
	`<handle2> <filename2> <pagecount2> <supportedoutputmodes2> <scancomplete2> <scanerrorcode2> <scanerrormessage2>`	

local.page

Fetches raw page data from a local file.

Request		
	! request.function	compressed.status
	! request.local	<local file handle>
	! request.page	<page number to return>
Result		
	! result.status	0 for success, otherwise error code
	! result.blocks	number of subsequent blocks, 1 if successful, 0 if not
	- result.message	error message if status not zero

Data	1x1 table for binary object `<page content>`	

local.docdata

Gets the simulated document data for a local file.

Request		
	`! request.function`	`local.docdata`
	`! request.local`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	4 column hybrid table / key-value pair list
	`<key1> <value1a> <value1b> <value1c>`
	`<key2> <value2a> <value2b> <value2c>`	

local.filedata

Gets the file information associated with an open local file.

Request		

	`! request.function`	`local.docdata`
	`! request.local`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`+ result.resource`	<resource set name>
	`+ result.profile`	<profile name>
	`- result.message`	error message if status not zero
Data	none, data returned in result message	

local.profiledata

Gets the profile settings associated with an open local file.

Request		
	`! request.function`	`compressed.profiledata`
	`! request.local`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	two column list of key-value pairs
	`<key1><value1>`
	`<key2><value2>`	

local.headerinfo

Determines if a given local file exists and returns the size of its Postscript file header.

Request		
	`! request.function`	`local.headerinfo`
	`! request.local`	`<file handle>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
	`+ result.exists`	1 if file exists, 0 if it does not
	`^ result.size`	file size, if file exists
Data	none, data returned in result message	

Licence Functions

licence.data

Gets a list of key-value pairs from the licence.

Request		
	`! request.function`	`licence.data`
	`! request.module`	module number to request licence data for
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	2 columns table of key value pairs
	`<key1> <value1>`
	`<key2> <value2>`	

Database Functions

database.list

Gets a list of the databases in a vault along with their descriptions.

Request		
	`! request.function`	`database.list`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	2 column table
	`<name1> <description1>`
	`<name2> <description2>`	

database.info

Gets a list of searches available for a specific database.

Request		
	`! request.function`	`database.info`
	`! request.database`	`<database name>`

Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	5 column table
	`<nunber1> <name1> <field1> <1> <description1>`
	`<nunber2> <name2> <field2> <2> <description2>`	

database.resolve

Used to determine if a customer or document record exists fetches the file and offset parameters.

Request		
	`! request.function`	`database.resolve <collection of search key/value pairs>`
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
	`^ result.offset`	offset of the customer or document record
	`^ result.file`	if a document was found, the file to which the document belongs

Data	none, data returned in result message. Note: that the function returns an error code if the document does not exist. Note: that if more than one match exists, the first is returned.	

database.search

Searches an index for matching records.

Request	This function maps to one of the following functions:
	`database.filtered`
	`database.unfiltered`
	The mapping depends on the following setting:
	`e2serverd.ini:`
	`[database1]`
	`filteredsearch=0`
	`0-do not filter searches (default, mitigation mode)`
	`When filteredsearch=0, database.search is an alias for database.unfiltered.`
	`1-filter searches normally (compatibility mode)`
	`When filteredsearch=1, database.search is an alias for database.filtered.`	

database.filtered

Searches an index for matching records.

Request		
	`! request.function`	`database.search <collection of search key/value pairs>`
	`? request.max`	limit the number of result rows to the specified value, otherwise uses an internal default
	`? request.fields`	request specific output columns for `database.search` separated by semicolons
	`? request.titles`	request specific output column titles for `database.search` separated by semicolons
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero
	`^ result.more`	1 if there are more matches, 0 otherwise
	`^ result.fixed`	the number of fixed (standard) columns

Data	a table of results with a title row
	`<title1> <title2>`
	`<data1,1> <data1,2>`
	`<data2,1> <data2,2>`
	Note that this may contain fixed, configured or specified columns. By default, it will return the fixed columns and those configured for the search in `database.ini`.
	• if `request.fields` or `request.titles` is present
	• if request.fields starts with a ; the fixed columns will be returned first then the specified columns
	• if request.fields ends with a ; the configured columns will be returned last
	• request.fields and request.titles must agree on the number of columns
	• the fixed columns returned depend on whether the index points to a customer record or document record
	• for customer records, 3 fixed columns are returned:
	`int.match;cust.account;int.pointer`
	• for document records, 9 fixed columns are returned:
	`int.match;doc.account;doc.date;int.file;int.pointer;doc.pages;int.modes;profile.format;doc.type`
	• the number of fixed columns may increase at a later date so make use of `result.fixed`

database.unfiltered

Search an index for matching records without filtering.

Request	Same as `database.filtered` except that filters will not be applied:
	`request.mode`
	`request.type`
	`request.file`
	`request.offset`

database.raw

Simple index search for matching records.

Request	Same as `database.unfiltered` except that it output columns cannot be customized using:
	`request.fields`
	`request.titles`

Data	For customer indexes, this function returns a 2-column table with the following fields:
	`int.match`
	`int.pointer`
	For document indexes, this function returns a 3-column table with the following fields:
	`int.match`
	`int.pointer`
	`int.file`
	Both types return a title row.

Render Functions

render.transform

Converts raw data into a application usable form and used to render pages to GIF, PDF, text, etc.

Request		
	`! request.function`	`render.transform`
	`! request.output`	`output mode`
	`? request.profile`	name of the profile to use instead of the job's profile
	`? request.resourceset`	name of the resourceset to use instead of the job's profile
	`! <collection of search key/value pairs>` `? <collection of rendering key/value pairs>`	
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent message blocks, 1 if successful, 0 if not
	`- result.message`	error message if status not zero

Data	1x1 table for binary object
	`<data>`
	Note:
	`GIF (mode 0, mask 1, mime image/gif)`
	`PNG (mode 4, mask 16, mime image/png)`
	`BMP (mode 7, mask 128, mime application/octet-stream)`	

	`? request.page`	page or starting page to be rendered
	`? request.orientation`	rotate the output bitmap N 90 degree increments(0-3)
	`? request.resolution`	the number of pixels wide the output bitmap should be (512,640,800,1024,1280,1600)
	`? request.findtext`	text to highlight
	`? request.findcolour`	colour to highlight the text in
	`? request.findcase`	0/1, 1 if the match should be case sensitive
	TIFF (mode 5, mask 32, mime image/tiff)	
	`? request.page`	page or starting page to be rendered
	`? request.pagecount`	the number of pages to be rendered **Note:** You can replace `request.page` and `request.pagecount` with `request.pageset`. See **Page sets** on page 165 for more information.
	`? request.background`	disable background
	`? request.orientation`	rotate the output bitmap N 90 degree increments (0-3)
	`? request.resolution`	the number of pixels wide the output bitmap should be (512,640,800,1024,1280,1600)
	`? request.depth`	8 for 8-bit paletted colour format (default) 32 for 24-bit RGB colour format
	`? request.findtext`	text to highlight

	`? request.findcolour`	colour to highlight the text in
	`? request.findcase`	0/1, 1 if the match should be case sensitive
	HTML (mode 1, mask 2, mime text/html)	
	`? request.page`	page or starting page to be rendered
	PDF (mode 2, mask 4, mime application/pdf)	
	`? request.page`	page or starting page to be rendered
	`? request.pagecount`	the number of pages to be rendered Note that you can replace `request.page` and `request.pagecount` with `request.pageset`. See **Page sets** on page 165 for more information.
	`? request.background`	disable PDF background
	`? request.pdfsecuritymode`	PDF security parameters from `API/USER` (when `PDFEncryption` defined in `PROFILES.ini`
	`? request.pdfuserpassword`	PDF user password (if `request.pdfsecuritymode=1`)
	`? request.pdfownerpassword`	PDF owner password (if `request.pdfsecuritymode=1`)
	`? request.pdfpermission`	PDF permission, the default is -1 (for every bit: 1=allow, 0=disable)
	RAW (mode 3, mask 8, mime application/octet-stream)	
	`? request.page`	page or starting page to be rendered
	STREAM (mode 8, mask 256, mime application/octet-stream)	

	? request.page ? request.pagecount	page or starting page to be rendered the number of pages to be rendered Note that you can replace request.page and request.pagecount with request.pageset. See **Page sets** on page 165 for more information.
	TEXT (mode 9, mask 512, mime text/plain)	
	? request.page ? request.pagecount	page or starting page to be rendered the number of pages to be rendered Note that you can replace request.page and request.pagecount with request.pageset. See **Page sets** on page 165 for more information.
	? request.cpix	characters per inch in the horizontal direction in text output grid (for example, 20)
	? request.cpiy	characters per inch in the vertical direction in text output grid (for example, 10)
	? request.textencoding	use the specific text encoding for text output
	? request.mark	start text output with a byte order mark 0/1
	COLLECTION (mode 10, mask 1024, mime varies) (none)	
	COMMENTS (mode 11, mask 2048, mime text/plain)	
	request.page	page or starting page to be rendered
	DOCINFO (mode 12, mask 4096, mime text/plain) (none)	
	TIFF G4 (mode 13, mask 8192, mime image/tiff)	

	? request.page ? request.pagecount	page or starting page to be rendered the number of pages to be rendered Note that you can replace request.page and request.pagecount with request.pageset. See **Page sets** on page 165 for more information.
	? request.orientation	rotate the output bitmap N 90 degree increments (0-3)
	? request.resolution	the number of pixels wide the output bitmap should be (512,640,800,1024,1280,1600)
	XML (mode 14, mask 16384, mime text/xml)	
	? request.page	page or starting page to be rendered
	DECODE (mode 16, mask 65536, mime text/xml)	
	? request.page ? request.pagecount ? request.textencoding ? request.mark	page or starting page to be rendered the number of pages to be rendered Note that you can replace request.page and request.pagecount with request.pageset. See **Page sets** on page 165 for more information. use the specific text encoding for text output start text output with a byte order mark 0/1
Search Parameters	Functions taking a set of search parameters	
	database.search	general purpose search for multiple entries with customizable output
	database.resolve	special function to resolve a single record
	render.transform	implicitly uses database.resolve to verify and locate documents

	Search modes	

	guid shortcut
	`! request.guid`
	iguid shortcut
	`! request.iguid`
		search guid index for specific guid
		search iguid index for specific iguid

	invlink shortcut `! request.account`	search invlink for documents associated with account
	`? request.date`	document date

	Generic prefix search:
	`! request.index`
	`? request.prefix`
	`? request.under`
		search number or index name
		match entries starting with prefix
		for "under x" indexes, the value for x (for example, account number)

	Exact search
	`! request.index`
	`! request.exact`
	`? request.under`
		search number or index name
		match entries that are equivalent to the specified value
		for "under x" indexes, the value for x (for example, account number)

	Range search
	`! request.index`
	`? request.under`
		search number or index name
		for "under x" indexes, the value for x (for example, account number)

	`? request.lower`	the lower bound of the search, may be ommitted to mean there is no lower bound
	`? request.upper`	the upper bound of the search, may be ommitted to mean there is no upper bound

	? request.excludelower	set to 1 to exclude exact matches of the lower bound from the results (<= vs <)
	? request.excludeupper	set to 1 to exclude exact matches of the upper bound from the results (<= vs <)
	? request.longupper	when set to 1 keys starting with the specified upper bound should be included in the result set (the range 200801 to 200810 does not include 20081031 unless longupper is set)
	Note that if you do not specify the required fields (marked with !), the search may default to the range search thus producing unexpected results	
	Common modifiers ? request.database ? request.reverse ? request.max	database name, defaults to "default" if 1, reverse the direction of the search maximum number of results to return
	Continuation modifiers ? request.first	partial key to start after when continuing
	Filters ? request.output ? request.file ? request.type ? request.offset	document's format must support the specified output mode document must be in specified file document's type field must match this value account or document record must be have specified offset

render.spool

Converts raw data into a form that is usable by the production reprint feature and transfer it to the server reprint input directory.

Request		
	`! request.function`	`render.spool`
	`! request.output`	`output mode`
	`? request.profile`	name of the profile to use instead of the job's profile
	`! <collection of search key/value pairs>` `? <collection of rendering key/value pairs>`	
Result		
	`! result.status`	0 for success, otherwise error code
	`! result.blocks`	number of subsequent blocks, always 0
	`- result.message`	error message if status not zero
Data	- none, data uploaded to server, status returned in result message	
Note: normally used with output mode 17 REPRINT		

Common scenarios

Looking up the list of databases		
	request.function	database.list
Looking up the list of searches available in a database		
	request.function request.database	request.database default
Search for customers with accounts starting with 45		
	request.function request.database request.index request.prefix request.max	database.search default 1 45 25
Searching for more matching customers		
	request.function request.database request.index request.prefix request.max request.first	database.search default 1 45 25 451234300
Search for documents under an account		
	request.function request.database request.account request.max	database.search default 451234300 25

Render a document as a GIF		
	`request.function`	`render.transform`
	`request.database`	default
	`request.account`	451234300
	`request.date`	2009/10/15
	`request.file`	20091015-resi-bill-mail
	`request.offset`	0000324007434FE
	`request.page`	1
	`request.resolution`	1024

Accessing Vault using sockets

Accessing Vault using sockets	Messages transmitted over the wire to the rendering engine are in INFO record format • the INFO record is a binary representation of the logical messages • logically this is in the form of a two dimensional table of values The INFO record consists of three basic parts • the header • the cell offset table • the cell data table The header consists of • the signature "INFO: in ASCII (0x49 0x4E 0x46 0x4F) • the length of the INFO record (as a 32-bit two's complement signed number in MSB format) • the number of rows in the INFO record (as a 32-bit two's complement signed number in MSB format) • the number of columns in the INFO record (as a 32-bit two's complement signed number in MSB format) The cell offset table • is a list of displacements from the start of the cell data • each entry corresponds to a certain cell in the logical table • rows appear from lowest to highest • in each row the columns appear lowest to highest • normally there are rows*columns+1 cell offset entries. If there are 0 rows or 0 columns, no cell offset table or cell data will be present, just the header (uncommon) • each displacement is stored as a 32-bit two's complement signed number in MSB format • the last entry is the offset 1 past the last byte of the cell data • the length of a cell's data is determined from the difference of its cell
Offset and the next cell offset	• they will be equal for an empty cell • there are certain validation properties • all cell offsets are greater than or equal to zero • the first cell offset is zero • the last cell offset is the length of the cell data table • each cell offset is greater than or equal to the previous cell offset • each cell offset is less than or equal to the next cell offset

	The cell data table • is a concatenated list of all data values • text data is stored as UTF-8 encoded code units • numbers are stored as decimal digits in text form • offsets are stored as hex digits in text form • binary data is stored as a sequence of arbitrary bytes • by convention binary data only appears in 1x1 tables • cell data appears in the same order as the cell offsets • cell data may not overlap
Sample INFO record	• this is a document record • document records are hybrid tables/key-value pairs • attributes are stored as key value pairs • reports are stored in 4 column rows • raw binary form

Sample

```
0x0B664B40  49 4e 46 4f 00 00 01 64 00 00 00 09 00 00 00 04
INFO...d........
0x0B664B50  00 00 00 00 00 00 00 07 00 00 00 18 00 00 00 19
................
0x0B664B60  00 00 00 1a 00 00 00 22 00 00 00 33 00 00 00 33
.......".3...3
0x0B664B70  00 00 00 33 00 00 00 3e 00 00 00 4a 00 00 00 4a
...3...>...J...J
0x0B664B80  00 00 00 4a 00 00 00 55 00 00 00 79 00 00 00 79
...J...U...y...y
0x0B664B90  00 00 00 79 00 00 00 82 00 00 00 83 00 00 00 83
...y...?...?...?
0x0B664BA0  00 00 00 83 00 00 00 8f 00 00 00 90 00 00 00 90  ...?.........
0x0B664BB0  00 00 00 90 00 00 00 9b 00 00 00 a3 00 00 00 a3
......?...£...£
0x0B664BC0  00 00 00 a3 00 00 00 ab 00 00 00 b5 00 00 00 b5
...£...«...µ...µ
0x0B664BD0  00 00 00 b5 00 00 00 bd 00 00 00 c0 00 00 00 c0
...µ...½...À...À
0x0B664BE0  00 00 00 c0                                     ...À
0x0B664BE0              73 65 63 74 69 6f 6e 53 74 61 72 74
sectionStart
0x0B664BF0  20 6f 66 20 44 6f 63 75 6d 65 6e 74 31 30 64 6f   of
Document10do
0x0B664C00  63 2e 6e 61 6d 65 4b 65 6c 76 69 6e 20 20 20 53  c.nameKelvin
   S
```

```
0x0B664C10  74 6f 63 6b 74 6f 6e 64 6f 63 2e 69 6e 76 6f 69
tocktondoc.invoi
0x0B664C20  63 65 34 34 37 20 36 35 30 20 37 36 35 33 64 6f  ce447 650
7653do
0x0B664C30  63 2e 61 64 64 72 65 73 73 35 38 36 35 20 57 65  c.address5865
 We
0x0B664C40  61 76 65 72 20 43 65 6d 65 74 65 72 79 20 52 64  aver Cemetery
 Rd
0x0B664C50  20 20 20 31 34 39 35 33 2d 31 39 33 32 64 6f 63
14953-1932doc
0x0B664C60  2e 70 61 67 65 73 35 64 6f 63 2e 73 65 63 74 69
.pages5doc.secti
0x0B664C70  6f 6e 73 31 64 6f 63 2e 61 63 63 6f 75 6e 74 30
ons1doc.account0
0x0B664C80  30 30 32 30 33 34 37 64 6f 63 2e 64 61 74 65 32
0020347doc.date2
0x0B664C90  30 30 32 2f 30 31 2f 31 31 64 6f 63 2e 74 79 70
002/01/11doc.typ
0x0B664CA0  65 61 66 70                                       eafp
```

Page sets

You can replace `request.page` with `request.pageset`.

Page sets are an alternate way of specifying which pages should appear in rendered output. Instead of providing `request.page` and `request.pagecount`, you can provide `request.pageset`.

The value for `request.pageset` is a comma delimited list of pages or page ranges to be included in the output.

The character * is used to indicate the last page when specifying a page or end of a page range. The same page may appear more than once in the output if desired.

Example:

1,1,3,5-7,*,10-*

Notices

precisely

1700 District Ave Ste 300
Burlington MA 01803-5231
USA

www.precisely.com

precisely

Documentation Feedback

You can help us to improve our product documentation by sharing your comments and feedback with us.

Please use the Adobe Acrobat Comments functionality to highlight any issues or to comment on a specific part of this document. Note that you will need to download this document to use the feedback option when viewed in a web browser.

If you are reading this document in hard copy and want to send us feedback, you may email us at EngageOneInformationDevelopers@precisely.com with the subject line "Documentation Feedback" and including the document information below

Document Title::
EngageOne Vault v2021.1.2 Programming Reference Guide

Software Version:
2021.1.2

Last updated date:
13th December 2022

Additional comments

Submit Feedback