pitney bowes

Customer Engagement

EngageOne Vault

Version 7.5

CMIS Connector

Table of Contents

8 - Vault Configuration for ODBC Export to a SQL Server

9 - Enabling ODBC export

1 - Preface

This guide provides introductory information about how the Vault CMIS connector operates and how to install it.

In this section

Skills and training

Vault operates under Microsoft Windows™ and UNIX based environments. It is recommended that you have Windows OS and UNIX networking experience. It is also recommended that the Vault administrator has some understanding of your host environment and that they know how to transfer files between the host systems and a Windows or UNIX server environment.

Those using the Rendering Engine for web applications should be conversant in HTML, the relevant web server (such as Apache, IIS, etc.), and the operating systems which may be in place at the installation location. It is assumed that the web developers are knowledgeable in the specific application environments into which the Rendering Engine is to be installed.

While every effort is made to provide sufficient information in this User Guide, it is recommended that users attend a training course covering the features of the Vault environment and the interaction of Vault products on the various supported platforms. Your Vault supplier will be able to advise you on course availability.

2 - Vault CMIS Connector

In this section

Introduction

Content Management Interoperability Services (CMIS) is an OASIS standard for accessing content repositories. It can be used by general purpose Enterprise Content Management (ECM) clients to communicate to repositories of different types. Vault CMIS Connector is a component that exposes Vault document data via the CMIS specification.

Licensing considerations

This product contains Apache Chemistry, version number 0.13.0, which is licensed under the Apache license, version number 2.0.

The license can be downloaded from:

http://www.apache.org/licenses/

Source code for this software is available from:

http://chemistry.apache.org/java/download.html

Supported environments

Vault CMIS connector is only supported on the WebSphere 8.5.5. application server. Some Vault features mentioned in this document are only available in Vault release 7.2.1 and higher.

3 - Planning and installation

The CMIS specification has an underlying model for organizing and accessing content repositories.

In this section

Overview of the Vault CMIS Connector model

A CMIS repository is a collection of documents. Logically, the CMIS model behaves like a file system but with optional features such as SQL-like queries. Each document in a CMIS repository has metadata associated with it. CMIS mandates some standard metadata and allows repositories to supplement the standard metadata with repository specific metadata. Vault documents also have metadata associated with them. Determining the relationships between the Vault document metadata and the CMIS document metadata is a critical piece of the setup. The bridge between the Vault data model and the CMIS data model is a SQL database. Vault document properties are exported to the associated SQL table when a Vault job is loaded for the first time or a manual export is triggered. The list of Vault document properties to be exported (and the SQL column names where they will go) must be configured as part of the Vault job setup.

CMIS elements not supported

Vault CMIS Connector is designed to support a model appropriate to Vault documents. As such, the set of CMIS capabilities supported is appropriate to a model of documents, properties and types that are fixed (cannot be created, modified or versioned) and document that can only be found by searching document metadata. The features not supported include (but are not limited to):

1. CMIS has a folder model for grouping documents within a repository. The Vault CMIS Connector does not support the folder model. All documents in a Vault CMIS repository can only be accessed by a query that returns document objects (referred to as unfiled documents within CMIS).
2. The following elements of the CMIS document model are not supported:
    a. Access Control Lists and ACL policies
    b. Creating or modifying documents in the repository
    c. Creating or modifying the document properties in the repository
    d. Retrieving document content stream using content ranges or HTTP ranges
    e. Alternate renditions for documents
    f. CMIS search joins
    g. CMIS searches "IN FOLDER" , "CONTAINS", "IN TREE" and "SCORE" keywords
    h. Full text search
    i. Document filing
3. The following elements are modified for the Vault CMIS Connector:

a. Documents that are not Vault collections mode documents are only available in a PDF content stream.
b. Documents that are not Vault collections mode documents use a default stream size. These documents are generated upon request from the Vault jobs and the size of the generated document is not known precisely in advance. Where a known size is required by the client, you can set up the Vault CMIS Connector to create an estimate of the size based on the number of pages and any PDF document requested will be padded out to that size.

URL paths for Vault CMIS Connector

The CMIS standard defines several different interfaces to its underlying functionality. These interfaces, or protocol bindings, define how the CMIS client and server communicate and format requests and responses. A particular CMIS client will typically use a single binding to communicate to a CMIS server.

Vault CMIS connector supports the following CMIS bindings

CMIS binding support

CMIS Version	Binding	Example URL
1.1	Web Services (SOAP)	http://localhost:8080/vault-cmis/services11/cmis?wsdl
1.1	AtomPub	http://localhost:8080/vault-cmis/atom11
1.1	Browser (REST/JSON)	http://localhost:8080/vault-cmis/browser
1.0	Web Services (SOAP)	http://localhost:8080/vault-cmis/services/cmis?wsdl
1.0	AtomPub	http://localhost:8080/vault-cmis/atom

Substitute the application server hostname, port and application context path as appropriate.

Note:

1. The URLs will be listed on the home page at http://yourCMIShostname/vault-cmis/

2. For configuring IBM Content Navigator, to use a CMIS repository, use the CMIS 1.1 AtomPub URL.

Determine Vault databases to expose via Vault CMIS Connector

Vault segregates its documents into a series of databases and each database has a series of indexes that can be searched to locate any particular document. (Note that a document may exist in more than one Vault database). CMIS has a similar concept called repositories where documents are stored. The Vault CMIS Connector model basically maps Vault databases to CMIS repositories. Part of the connector setup is to determine which Vault databases (meaning which set of documents) you want to want to expose as CMIS repositories.

Determine document metadata to expose via Vault CMIS Connector

The CMIS protocol requires that each document have properties. Each property is defined by its name and its type. CMIS also defines a series of properties that must exist for each document. You can also define custom document properties for each document in a repository. For the required CMIS properties (CMIS base properties), you can choose to either map the property to an underlying Vault document property, define a custom fixed value or use the default fixed value set up in the Vault CMIS Connector. Mapping the base property to a Vault document property is done by linking it to a column in the SQL database table so each Vault document property that you want to use for mapping must be defined in the associated repository SQL table. Each CMIS document property has a series of mandatory attributes. The Vault CMIS Connector allows you to define the following attributes related to a CMIS document property:

CMIS property types

The base CMIS property types, their type and default fixed value in the Vault CMIS Connector are:

CMIS property types

CMIS base value	CMIS Type	Default value
cmis:creationDate	Datetime	2000/01/01
cmis:isLatestVersion	Boolean	True
cmis:lastModificationDate	Datetime	2000/01/01
cmis:objectId	ID	Each document must have a unique objectid. This will be taken from the CmisId column in the SQL table. CMIS ID fields are required to be unique for each document with the repository so care must be used when choosing how map this field. You may choose a unique ID generated by Vault when the document is loaded, the Instance GUID from the generating application (if supplied) or you may use the new int.hash support in the Vault ODBC export setup.
cmis:name	String	Will be taken from the CmisId column in the SQL table
cmis:description	String	Will be taken from the CmisId column in the SQL table
cmis:versionLabel	String	"N/A"
cmis:versionSeriesId	String	Will be taken from the CmisId column in the SQL table
cmis:isVersionSeriesCheckedOut	Boolean	False

CMIS property types

cmis:versionSeriesCheckedOutId	String	Will be taken from the CmisId column in the SQL table
cmis:baseTypeId	ID	"cmis:document"
cmis:createdBy	String	"System"
cmis:lastModifiedBy	String	"System"
cmis:changeToken	String	"0"
cmis:isImmutable	Boolean	True

Custom document properties can appear as CMIS base properties. The property names should not use the "cmis:xxxx" format used by predefined CMIS base properties.

Note: Some CMIS clients will limit their selection lists of document properties to the CMIS base properties (those named cmis:xxxx and custom properties that appear as base properties). If you do not want to create new base properties, you should consider mapping your custom property to a predefined CMIS base property.

Determine Vault metadata required for searching and rendering

Beyond the Vault document metadata that you will decide to expose to a CMIS client using the connector, there are a number of Vault document metadata fields that must be stored with each document in the connector to allow the connector to render documents from Vault. The actual properties to be stored depend on the type of Vault document you are storing (standard or collections mode documents) and how the document is to be identified for rendering to the CMIS client.

A Vault document can be identified within Vault by either its Filename or Offset properties (assigned by Vault) or by a unique document identifier assigned by the document generating application (instance GUID). For the file or offset model, the Vault account (`doc.account`), date (`doc.date`), file (`int.file`) and offset (`int.offset`) needs to be stored in the SQL table. For the instance GUID model, the GUID also needs to be stored (`docInstanceGUID`).

If the Vault collections mode is being used, then the Vault properties modes (`int.modes`), document type (`doc.type`), document file name (`file.name`) and file size (`file.size`) are also required

to be stored in the SQL table in order for the CMIS Connector to return the document to a CMIS client.

For non-collections mode document, the Vault document page count (`doc.pages`) is also a generally useful metric to expose to CMIS clients as the standard CMIS content stream length is not available for each document as the content stream is created dynamically by Vault on demand. If your Vault CMIS Connector is configured to deliver a content stream size based on the page count, then page count is a required Vault property to be exported.

Note that it is possible to add "fixed" values to be exposed through the CMIS Connector. This is possible as the setup of the values to be written into the SQL table is at the Vault profile. The "value" to be written does not have to be based on document Vault metadata, it can also be a simple fixed value. The value can also be a composed value that concatenates multiple Vault metadata values or fixed values.

Standard Vault properties

The table below has the list of standard vault property names, its description, the Vault data type, and the suggested SQL column name.

Standard Vault properties

Vault property name	Vault data type	Suggested SQL column name	Description
`doc.account`	String	VaultAccount	The account of the customer associated with the document. This is a mandatory field if you are using the account/date/file/offset model.
`doc.address`	String	VaultAddress	The address field of the customer associated with the document.
`doc.date`	String	VaultDate	The date field for the document. Note that the date as a string can be coerced into a SQL date as part of the ODBC setup for Vault. The date field for the document must be a standard date for it to be coerced. This is a mandatory field if you are using the account/date/file/offset model and must be a String type.

Standard Vault properties

int.file	String	VaultFile	The filename for the job that has the document. This is a mandatory field if you are using the account/date/file/offset model.
file.name	String	VaultFilename	For collections mode, this is the filename of the original document within the collection.
doc.name	String	VaultName	The name of the customer associated with the document.
int.profile	String	VaultProfile	The profile name associated with the document.
docInstanceID	String	VaultInstance	The document instance GUID.
doc.guid	String	VaultMaster	The document master GUID.
doc.type	String	VaultType	The document type.
file.size	Int	VaultFileSize	The document file size for a collections mode document.
int.modes	Int	VaultModes	The modes that a document can be rendered in.
doc.pages	Int	VaultPages	The number of pages in the document.
int.offset	String	VaultOffset	The offset of the document record. This is required when using the Account/Date/File/Offset model.

Standard Vault properties

`int.database`	String	VaultDatabase	The name of the Vault database on the Vault server that the document should be associated with. If this property is defined to the Vault CMIS Connector and the value in the SQL table is not a null string, then the value will be used as the Vault database name when rendering the document. Note that the int.database is a tag for the Vault CMIS Connector and cannot be used in the Vault ODBC export definition.
`int.store`	String	VaultStore	The name of the Vault CMIS store related to this document. If the store name is supplied and matches a defined store name in the connector configuration, then that store will be used to render the document. Note that the int.store is a tag for the Vault CMIS Connector and cannot be used in the Vault ODBC export definition.

Determine layout of SQL database

Once you have determined the CMIS base values and Vault metadata values that will be exposed via the connector, the next step is to create the SQL table that Vault will insert rows into for each document. For each piece of metadata that will be exposed as a non-fixed value CMIS property, you will need to determine a unique column name in the SQL table and its associated datatype. In addition, you may need to define additional columns for self-defining properties. Specifically, you may need an "ID" field that the SQL database engine will populate with a unique value for each document to use as the CMIS document id (if you are not already using a Vault metadata field like IGUID or int.hash for this purpose).

The supported SQL datatypes are integer, datetime, varchar and decimal. The date type should only be used for the CMIS document date properties (`cmis:creationDate`, `cmis:lastModificationDate`). The integer type is only used for storing the Vault document page count (`doc.pages`) and the CMIS ID (`cmis:id`) if it is auto generated. All other fields will be of the type varchar. For the each varchar type field, you will have to determine the maximum string length that will be stored.

The following configurations should be considered

1. Designating some columns as key fields if you want to index the table for faster searching and lookup. Your primary search properties are good candidates.
2. Define the CMIS ID column as a primary key and unique. CMIS requires that this value be unique for each document in a repository.
3. The File and Offset properties for a given document within a Vault database are intended to be unique. Making the combination unique in the SQL database will help identify if this rule is broken.

Finally, you will need to determine a table name and create a script to create the SQL table based on your requirements.

Create properties file for the connector

The Vault CMIS Connector configuration file is a file in JSON (JavaScript Object Notation) that describes to the Vault CMIS Connector the repositories associated with the connector. For each repository, you will need to specify the repository name and description, the SQL database connection information, the Vault server connection information, the CMIS base properties (property name, SQL column name and the Vault property name, optional default values, CMIS base type, description) and the custom properties. This property file is used by the connector to convert CMIS search queries into SQL searches, convert search results into CMIS document object properties, and to access the Vault server to render documents to PDF or retrieve documents stored in the collections mode.

Basic properties file layout

The basic properties file layout is:

```
{
 "security": { … },
 "databases": { … },
 "stores" : { … },
 "vaultDatabases": { … },

 "repositories" :
{
"repository1" :
     {
     "info" : { … },
     "vaultDatabase":  …,
     "database": …,
     "types": { … },
     "security" : { … },
     "profile" : { … },
     "store":  …
    },
"repository2" :
     {
     "vaultDatabase": … ,
     "database": … ,
     "info" : { … },
     "types": { … },
     "security" : { … },
      "profile" : { … },
     "store":  …
   }

}
```

Databases section

The Databases section describes the SQL database connections. It has the following syntax:

```
"databases": {
       "connection name 1":  ... {
            "driver":  … ,
            "url": …
            },
       "connection name 2":  ... {
            "driver":  … ,
            "url": …
            }
        },
```

Where

Attribute	Description	Example
connection name	The name of this SQL connection. This name will be used to select a database for the "database" attribute at the repository definition.	`"sql1" : { … }`
driver	Specifies the type of database. If driver is not specified then URL is the JNDI name for the datasource, otherwise the driver will be registered. Note that the JDBC driver must be at least type 4 (sqljdbc4.jar for example)	`"driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"`

url	The JDBC connection string for this repository
	DriverManager example:
	`"url": "jdbc:sqlserver://localhost:1433;databaseName=exportdata;user=vault;password=test;"`
	Datasource example:
	`"url":"java:comp/env/jdbc/sqlserver"`	

Note: If you are using encrypted strings for URL, then the system property for `vault.default.key` will need to be set up in the Java environment. See "Setting the default key file for Java environments" in the Vault Customizing Guide for details.

Stores section

The stores section specifies how the Vault CMIS Connector will connect to the Vault server. It has the following syntax:

```
"stores": {
            "name of store 1": {
               "host": … ,
               "port": … ,
               "ssl": {
                  "keyPw": … ,
                  "keystoreFilePath":  … ,
                   "keystorePw":  … ,
                   "truststoreFilePath":  … ,
                   "truststorePw":  …
                              }
               "login": {
               "serviceUser" : ...
               "servicePw" : ....
      }
},
```

Where

Attribute	Description	Example
name of store	The name assigned to this store definition. This name will be used for the "store" attribute at the vault Database definition or at the repository level.	`"vault1": { …`

host	The host name or IP address of the Vault server associated with this Vault CMIS Connector	`"host": "127.0.0.1"` `"host": "cmis1.local.pvt"`
port	The port number of the Vault server that the Vault CMIS Connector will connect to when requesting documents	`"port": 6003`
ssl	The section heading is for supporting ssl socket communications between the Vault CMIS Connector and the Vault server. SSL support is optional and not enabled by default. SSL communications requires setup on the Vault server side and SSL certificates. For details of how to create a certificate and place it in a trusted store, see "Vault REST SSL setup" in the "Vault REST API" Guide.	
keyPw	the password for the key file	`"keyPw": "changeit"`
keystorePW	the password of the key store	`"keystorePw": "changeit"`
keystoreFilePath	the key store file path	`"keystoreFilePath”:` “C:/Program Files (x86)/Java/jdk1.8.0_25/jre/lib/security/cacerts"
truststoreFilePath	the key store file path	`“truststoreFilePath"` "C:/Program Files (x86)/Java/jdk1.8.0_25/jre/lib/security/cacerts"
truststorePw	the password of Java truststore	`"truststorePw": "changeit"`
login	This section heading is for supporting the optional Vault authentication between the Vault CMIS Connector and the Vault server.	
serviceuser	the service userid for Vault authentication	`“serviceUser”: “cmis-connector”`

servicePW	the password assigned to the service user	`"servicePW": "thepassword"`

Note: If you are using encrypted passwords, then the system property for `vault.default.key` will need to be set up in the Java environment. See "Setting the default key file for Java environments" in the "Vault Customizing Guide" for details.

Vaultdatabases section

The vaultDatabases section sets the name of the Vault database that the Vault CMIS Connector will use when rendering documents and the Vault server (store attribute) that the document will rendered from. It has the following syntax:

```
"vaultDatabases" : {
     "name of database 1": {
          "store": …,
          "name": …
         },
     "name of database 2": {
          "store": …,
          "name": …
         },
     …
     }
```

Where

Attribute	Description	Example
name of database	The database name that will be used in the "vaultDatabase" attribute when this database is selected.	`"vaultdb1": …`
store	The "store" by name (Vault server) where this Vault database resides.	`"store": "vaultsystem1"`
name	The actual name of the Vault database at the Vault server. If name is omitted, then the "name of the database" is used.	`"name": "production1"`

Security section

The security section lists which roles will be allowed to access the general repository system. Role verification must be enabled at the application server for this to be effective. It is a simple list of comma separated role names in double quotes in a JSON array. If no role verification is desired, then this section should be omitted. Note that there is an additional (optional) security section at the repository definition level which limits access to a specific repository. The syntax of the section is:

```
"security": {
    "roles":  [
              "role1",
              "role2",
             "…"
             ]
  }
```

Repositories section

The database connection must be specified for each repository. The syntax is:

```
"database": "one of the connection names defined in the databases section"
```

The vaultDatabase, table and store attributes that can be set at the repository and type level. The selection at the type level is optional and will override the one at the repository level for that specific type. Store is optional as it is required in the vaultDatabases definition but specifying overrides it. The syntax is:

```
"vaultDatabase": " one of the Vault database names defined in the
vaultDatabases section",
"table": "the name of the SQL table for this repository"
"store": "one of the store names defined in the stores section"
```

Info subsection

The info section sets the repository name and description visible to the CMIS clients. It has the following syntax:

```
"info": {
            "description":  … ,
            "name":  …
         }
```

Where

Attribute	Description	Example
description	The repository description as presented to a CMIS client	`description": "A Demonstration Vault Repository"`
name	The repository name as presented to a CMIS client	`"name": "Demo"`

Types subsection

The types section is used to:

1. Redefine the default CMIS base properties (named cmis:xxxx by convention)
2. Define Vault properties that are required but not visible to the CMIS client
3. Define any custom properties that will be visible to a CMIS client

The "`cmis:documents`" section (if it is defined) must precede any subtype definitions.

The general layout is:

```
"types": {
    "cmis:document": {
         "properties":  [
                {cmis property definition 1},
                {cmis property definition 2},
                 …
                {vault property definition 1},
                {vault property definition 2},
                …
            ] ,

         … Optional selections for table, store and vaultDatabase
```

```
        }
        }
```

The `cmis:document` section is where you will possibly redefine the CMIS base properties from their default definition, where you can add custom property definitions that will be exposed to a CMIS client that uses the Vault CMIS Connector (by assigning a `cmisName` attribute) and define the Vault properties that the Vault CMIS Connector needs to operate correctly. Each custom property in this section must either have a column in the SQL table where the property value will be taken from or a fixed value.

Custom properties that are exposed to CMIS clients are effectively introducing new CMIS base document properties with non-standard naming. The impact on the behavior of any given CMIS client cannot be predicted in advance and will require testing with that client.

Note that some clients (IBM Content Navigator for instance) will only allow you to configure properties for searching that are base types, not subtypes.

Property definition syntax

```
{
  "cmsiName":  … ,
  "columnName": … ,
  "default": … ,
  "defaultType": … ,
  "queryable": … ,
  "orderable": … ,
  "vaultName": … ,
  "description": … ,
  "displayName": …
}
```

Where

Attribute	Description	Example

`cmisName`	The name of the document property that will be visible to a CMIS client in the format "nameType:name". By convention, true CMIS base properties have a nameType of "cmis" so this is reserved. Any custom name should avoid using special characters. If this attribute is not supplied, the property will not be visible to CMIS clients.	`"cmisName": "cmis:createdBy"`
		`"cmisName": "vault:vaultOffset"`
		`"cmisName": "custom:regionCode"`

`columnName`	The name of the column in the associated SQL table where the data for this property will be fetched from. This attribute is optional. If it is omitted, then you are defining a fixed value property and you should specify a "default" attribute.	`"columnName": "cmisDate"` `"columnName": "VaultOffset"`

`default`	The value to set the property to if the property is a fixed value (no ColumnName defined) or if he value in the SQL column is null or empty	`"default": "admin"`
		`"default": 0`
		`"default": "2000/01/31"`

`defaultType`	The values that will be considered a proxy for a "not set" in the SQL table column. This attribute is optional. Multiple values are separated by commas. If this attribute is not set, then the database "not set" or "empty" value will trigger setting the property to the default value.	`"defaultType": ""` (empty string) `"defaultType": -1` (-1 value for an int field) `"defaultType": "placeholder,"` (empty string or the word placeholder means not set)
`queryable`	Sets whether the property is queryable per the CMIS definition. All properties that have the columnName attribute will be "queryable":"true" by default. Otherwise, they will be "queryable":"false".	`"queryable": "false"`

`orderable`	Sets whether the property is orderable per the CMIS definition. All properties that have the columnName attribute will be "orderable":"true". Otherwise, they will be "orderable":"false".	`"orderable":" false"`

`vaultName`	The Vault property name associated with this document property. This is generally required for each column in the SQL table that is filled by the Vault ODBC connector while loading documents. This attribute is used to identify Vault properties that the Vault CMIS Connector needs for its operation.	`"vaultName": "int.file"`
		`"vaultName": "doc.name"`
		`"vaultName": "doc.pages"`

`description`	The description of this property that will be presented to a CMIS client. This attribute is only required if the "cmisName" attribute is defined.	`"description": "document page count"`
`displayName`	The name of for this property that will be presented to a CMIS client. This attribute is only required if the "cmisName" attribute is defined.	`"displayName": "Account Number"`

`type`	The CMIS base type for this property. Valid types are "integer","boolean", "id", "string", "datetime", "decimal", "html", and "uri". The default for the CMIS base properties will be as defined by the CMIS specification. If you are defining custom properties, the type should match up with the type defined for the matching column in the SQL table if the "columnName" attribute is set. For html and uri, the SQL columns should be mapped as string.	`"type": "string"`
		`"type": "boolean"`
		`"type": "integer"`

Example cmis:document

In this example, we are overriding the CMIS base properties for some dates, created by (local default) and the CMIS ID field. Note that the CMIS ID is a special case of a property in terms of its uniqueness requirements. This also has the basic additional columns that are required for a Vault non-collections documents database.

```
"cmis:document": {
 "properties": [
  {
  "cmisName": "cmis:creationDate",
  "columnName": "CmisDate",
  "default": "2000/01/01"
  },
  {
  "cmisName": "cmis:lastModificationDate",
  "columnName": "CmisDate",
  "default": "2000/01/01"
  },
  {
  "cmisName": "cmis:createdBy",
  "default": "Your document team"
  },
  {
  "cmisName": "cmis:objectId",
  "columnName": "CmisId",
  "vaultName": "int.hash"
  },
  {
   "cmisName": "vault:pagecount","
   "columnName": "VaultPages",
   "description": "The number of pages in the document",
   "displayName": "Vault Pages",
   "vaultName": "doc.pages",
```

```
    "default": 999999,
    "type": int
   },
   {
    "cmisName": "vault:account",
    "columnName": "VaultAccount",
    "description": "The account associated with the document",
    "displayName": "Vault Account",
    "vaultName": "doc.account",
    "type": "string"
   },

{
    "cmisName": "vault:date",
    "columnName": "VaultDate",
    "description": "The date associated with the document",
    "displayName": "Vault Date",
    "vaultName": "doc.date"
   },
   {
    "cmisName": "vault:file",
    "columnName": "VaultFile",
    "description": "The job containing document",
    "displayName": "Vault File",
    "vaultName": "int.file"
   },
   {
    "columnName": "VaultOffset",
    "description": "The offset of the document record",
    "displayName": "Vault Offset",
    "vaultName": "int.offset"
                                          }
                                       ],
                    "table": "exportdata.dbo.VaultDocuments"
   }
```

Security subsection

The repository security section details which roles will be allowed to the specific repository being defined if role verification is enabled at the application server. It is a simple list of comma separated role names in double quotes in a JSON array. If no role verification at this level is desired, then this section should be omitted. Note that any roles defined at this level must also appear at in the primary security section if it is defined. The syntax of the section is:

```
"security": {
    "roles":  [
              "role1",
              "role2",
             "…"
             ]
   }
```

Note:

- If the "security" section or "roles" inside the "security" section is omitted, then role verification will be disabled.
- If the "roles" is configured with an empty role list, role verification will be enabled and since there are no roles, no one would be allowed access. At the general level, this means no access to the connector. At repository level, this would mean no roles would be allowed to access to the repository.

Profile subsection

The profile section defines attributes related to the Vault profile that is used to process the document on the Vault system itself. Currently, you can set up the optional PDF file stream size padding here. As the setting is based on the Vault profile, the Vault document property int.profile must be configured as a SQL database column otherwise the default section "*" is used. The file size for a document will be the sum of a base value and a multiplier based on the page count. The syntax of the section is:

```
"profile": {
            "*": {
               "pdfresize": {
                  "bytesbase":  … ,
                  "bytesperpage":  …
               }
            },
            "profile name": {
               "pdfresize": {
                  "bytesbase":  …  ,
                  "bytesperpage":  …
               }
            }
```

Where

Attribute	Description	Example
`profile name`	The name of the Vault profile for this document. This must match the contents of the SQL table column entry where vault.profile is stored. A profile of "*" indicates that this is the default set of attributes to be used if the Vault profile in the SQL table is not enumerated in this section.	`"*"` `"AFP1"`
`bytesbase`	The base number to be used for the document file size calculation.	`"bytesbase": 100000`

`bytesperpage`	The multiplier to be applied to the page count	`"bytesperpage": 10000`

Configure Vault to load metadata into SQL database

In order to support the CMIS document search model, the Vault CMIS Connector makes use of an external database to host a document properties table that allows for the queries that CMIS exposes. Vault needs to be set up to export properties of interest into this table when a job is loaded into Vault. These properties are used in queries and as parameters passed to Vault in order to render documents. See the "ODBC Export" section of the "Vault Customizing Guide" for details.

4 - Vault CMIS Connector examples

In this section

Scenario A - simple document model

For this example, we will model the setup of a repository for a Vault database that is using the standard document model.

The CMIS specifics are

1. The CMIS repository will be named "demo".
2. We want to expose the Vault document property named `COMMTYPE` (Communication type) from the Vault job journal file as a custom CMIS document property.
3. We will expose the customer name and account number as CMIS document properties.
4. We will use the Vault document date for several CMIS date related properties.

The SQL specifics are

1. The SQL server userid is vaultodbc.
2. The SQL table name is Vault.dbo.Documents.

The Vault specifics are

1. Vault database is "database1"
2. Vault profile is "DemoAFP"

The CMIS specifics are

1. We will map the CMIS Id to the Vault hash value for the document.
2. We will map creation and last modified dates to the Vault document date.
3. The document account and communication type will be custom base properties.

SQL table planning

SQL table planning

SQL column name	Column data type	Vault property to be loaded	Notes
CmisDate	datetime	`doc.date`	Document date in a CMIS compatible format
CmisId	string	`int.hash`	Unique ID from Vault ID processing

VaultAccount	string	`doc.account`	Need this to render document
VaultDate	string	`doc.date`	Need this to render document. Will not be exposed to CMIS client
VaultFile	string	`int.file`	Need this to render document. Will not be exposed to CMIS client
VaultOffset	string	`int.offset`	Need this to render document. Will not be exposed to CMIS client
VaultCommType	string	`COMMTYPE`	Custom property
VaultPages	integer	`doc.pages`	document page count
VaultName	string	`doc.name`	customer name

SQL script

`create-table.sql`	The script file name.
`USE [Vault]` `GO`	This part of the script switches to the database used by the CMIS Connector.
`SET ANSI_NULLS ON` `GO`	Specified the ANSI behavior when comparing nulls.
`SET QUOTED_IDENTIFIER ON` `GO`	Use ISO conventions for quoted identifiers.

`CREATE TABLE [dbo].[Documents](`	Begin creating the Documents table.

`[CmisId] [varchar] (64)NOT NULL,`
 	
`[CmisDate] [date] NOT NULL,`
`[VaultAccount] [varchar](128) NOT NULL,`
`[VaultDate] [varchar](64) NOT NULL,`
`[VaultFile] [varchar](256) NOT NULL,`
`[VaultOffset] [varchar](32) NOT NULL,`
`[VaultPages] [int] NOT NULL,`
`[VaultName] [varchar](128) NULL,`
 	
`[VaultCommType] [varchar](128) NULL`
	Specify the names and types of each column.

`CONSTRAINT PK_CmisId PRIMARY KEY`
 	
`(CmisId)`
	Declare any constraints on the table.

`) ON [PRIMARY]` `GO`	End table creation.

`CREATE INDEX IX_VaultAccount_VaultDate`
`ON [dbo].[Documents]`
`(VaultAccount,VaultDate);`
`GO`
	Create indexes for the table.

Vault side planning

The Vault `database.ini` file describes the indexes that Vault uses for tracking documents within Vault itself. To use the standard document model, the database definition for "database1" should have a standard invlink index defined. The file fragment should appear similar to the following:

```
[database1]
Index1=account,cust.account,wjctul,Account Numbers
Index2=invlink,doc.date,xdhasb,Dates under this account
```

The profiles.ini file would need the following lines added to the `profiles.ini` file where the profile named profile1 is defined. The `profiles.ini` would appear as follows (comments start with "#").

```
; name of profile
[DemoAFP]
; standard profile definition lines (not supplied)
; ODBC setup
; enable ODBC export for the profile.
ExportEnable=1


; Name of the System DSN set up to allow the load process to export data
 to ; the SQL server.
ExportSource=VaultCMIS

; credentials used to log into the SQL server for the export process.
ExportUser=vaultodbc
ExportPassword=<snip>

; target table name to insert records into
ExportTable=Vault.dbo.Documents


; export fields (numbered consecutively from 1)
; Syntax is SQL column name, Vault property name, data type
; store date for CMIS as a date field
ExportField1=CmisDate,doc.date,d
ExportField2=VaultAccount,doc.account,s
; store Vault date as a string
ExportField3=VaultDate,doc.date,s
ExportField4=VaultFile,int.file,s
ExportField5=VaultOffset,int.pointer,s
ExportField6=VaultPages,doc.pages,n
ExportField7=VaultName,doc.name,s
ExportField8=VaultCommType,COMMTYPE,s
ExportField9=CmisId,int.hash,s
```

Vault CMIS Connector planning

The next step is to create the Vault CMIS Connector properties file. For this scenario, the file would look like:

`{`
`  "databases": {`
`  "sql1" : {`
`  "driver":`
`"com.microsoft.sqlserver.jdbc.SQLServerDriver",`
 	
`  "url":`
`"jdbc:sqlserver://127.0.0.1:1433;databaseName=..."`
 	
`  }`
`  },`
	Define the SQL database connections. The driver specifies the type of database and the URL is the JDBC connection string.

`"stores": {`
`  "vault1" : {`
`  "host": "vault1.local.pvt",`
`  "port": 6003`
`  }`
`},`
 	
`"vaultDatabases": {`
`  "database1" : {`
`  "store": "vault1"`
`  }`
`}`
	Define the stores (Vault server access) and the vaultDatabases to be used by the repositories

`"repositories": {`	Begin definition the list of repositories.

`"demo": {`
`  "database" : "sql1",`
`  "vaultDatabase": "database1",`
`  "table": " Vault.dbo.Documents",`
	Start defining a specific repository. The name here is the repository id in CMIS. Select database, vaultDatabase and table.

`"info": { "description": "Demo Data", "name": "Demo" },`	Set the user visible name and description for the repository.
`"types": { "cmis:document": { "properties": [`	Begin declaring the properties for CMIS base properties.
`{ "cmisName": "cmis:objectId", "columnName": "CmisId" },`	Define the CMIS object id and link it to the CmisId column in the database.
`{ "cmisName": "cmis:creationDate", "columnName": "CmisDate" }, { "cmisName": "cmis:lastModificationDate", "columnName": "CmisDate" },`	Set creation and last modified date to the document date. We only need columnName as the rest is defaulted. The Vault ODBC setup will populate the CmisDate with the document date.
`{ "cmisName": "custom:account", "columnName": "VaultAccount", "description": "The account associated with the document", "displayName": "Vault Account", "vaultName": "doc.account" },`	Define the account number property and expose it to CMIS clients as a custom property. Account is also required to render the document (vaultName attribute)
	Define the communication type and expose it to CMIS clients as a Vault property

<pre>{
"cmisName": "vault:commtype",
"columnName": "VaultCommType",
"description": "The communication type associated with the document",
"displayName": "Vault Communication Type",
"vaultName": "COMMTYPE"
},</pre>	

<pre>{
"cmisName": "custom:name",
"columnName": "VaultName",
"description": "The customer name for the document",
"displayName": "Vault Customer Name",
"vaultName": "doc.name"
},</pre>
	Define the customer name and expose it to CMIS clients as a custom property.

	Define the remaining Vault properties besides account (date/file/offset) that are required to render documents from the Vault CMIS Connector. These are not exposed to the CMIS clients (no cmisName attribute).

```
{
   "columnName": "VaultFile",
   "description": "The job containing
 document",
   "displayName": "Vault File",
   "vaultName": "int.file"
},

{
   "columnName": "VaultOffset",
   "description": "The offset of the
 document record",
   "displayName": "Vault Offset",
   "vaultName": "int.offset"
},

{
   "columnName": "VaultDate",
   "description": "The date of the
document",
   "displayName": "Vault Date",
   "vaultName": "doc.date"
},
{
   "columnName": "VaultPages",
   "description": "The number of
pages in the document",
   "displayName": "Vault Pages",
   "vaultName": "doc.pages"
}],

              }
          }
       }
   }
}
```

Scenario - B document collection model

This scenario is similar to scenario A but instead of standard documents, the Vault documents in the repository are stored as collections. There will be no communication type or page count property stored or exposed to the CMIS clients.

There are a few key differences to the way collections documents are configured compared to standard documents. These are

1. A Vault collection stores and accesses documents in a collection job as series of standalone files that can be retrieved only in the original format (not converted to different formats). As such, the static nature of the document allows the following additional documents properties to be available:
    a. File size (content stream size)
    b. Filename
    c. Filetype
2. The document mode must be available to the Vault CMIS Connector via the SQL table in order for the connector to determine that the document is stored in a collection.

CMIS specifics

1. The CMIS repository will be named "collection".
2. We want to expose the Vault document property filename and filesize as CMIS base document properties.
3. We want to expose the document type (`doc.type`). If the filename does not have a file type that can be mapped to a MIME type, then document type is used to do the MIME type mapping.
4. We will expose the customer name and account number as CMIS document properties.
5. We will map the CMIS Id to the Vault hash value for the document.
6. We will map creation and last modified dates to the Vault document date.

SQL specifics

1. The SQL server userid is vaultodbc
2. The SQL table name is Vault.dbo.Collections

Vault specifics

1. Vault database is "collection1"
2. Vault profile is "DemoCollection"

SQL table planning

SQL table planning

SQL column name	Column data type	Vault property to be loaded	Notes
CmisDate	datetime	`doc.date`	Document date in a CMIS compatible format
CmisId	string	`int.hash`	Unique id from Vault id processing
VaultAccount	string	`doc.account`	Need this to render document
VaultDate	string	`doc.date`	Need this to render document. Will not be exposed to CMIS client
VaultFile	string	`int.file`	Need this to render document. Will not be exposed to CMIS client
VaultOffset	string	`int.offset`	Need this to render document. Will not be exposed to CMIS client
VaultName	string	`doc.name`	customer name
VaultType	integer	`doc.pages`	document page count
VaultName	string	`doc.type`	document type. Required for collections
VaultFileSize	int	`file.size`	document size. Required for collections
VaultFileName	string	`file.name`	document filename. Recommended for collections

SQL table planning

VaultMode	int	`int.modes`	The Vault document mode. Required for collections

SQL script

`create-table.sql`	The script file name.
`USE [Vault]` `GO`	This part of the script switches to the database used by the CMIS Connector.
`SET ANSI_NULLS ON` `GO`	Specified the ANSI behavior when comparing nulls.
`SET QUOTED_IDENTIFIER ON` `GO`	Use ISO conventions for quoted identifiers.
`CREATE TABLE [dbo].[Collections](`	Begin creating the Collections table.

`[CmisId] [varcahr] (48)NOT NULL,`
 	
`[CmisDate] [date] NOT NULL,`
`[VaultAccount] [varchar](128) NOT NULL,`
`[VaultDate] [varchar](64) NOT NULL,`
`[VaultFile] [varchar](256) NOT NULL,`
`[VaultOffset] [varchar](32) NOT NULL,`
`[VaultName] [varchar](128) NULL,`
 	
`[VaultType] [varchar](32) NULL,`
`[VaultFileName] [varchar](256)NULL,`
`[VaultFileSize] int NULL,`
`[VaultMode] int NULL`
	Specify the names and types of each column.

`CONSTRAINT PK_CmisId PRIMARY KEY (CmisId)`	Declare any constraints on the table.
`) ON [PRIMARY]` `GO`	End table creation.

Vault side planning

The Vault `database.ini` file describes the indexes that Vault uses for tracking documents within Vault itself. To use the collections document model, the database definition for "collections1" should have a standard invlink index defined. The file fragment should appear similar to:

```
[collections1]
Index1=account,cust.account,wjctul,Account Numbers
Index2=invlink,doc.date,xdhasb,Dates under this account
```

The `profiles.ini` file would need the following lines added to the `profiles.ini` file where the profile named profile1 is defined. The `profiles.ini` would appear as follows (comments start with "#").

```
; name of profile
[DemoCollections]
; standard collection profile definition lines (not all supplied)
Format=collection
; ODBC setup
; enable ODBC export for the profile.
ExportEnable=1


; Name of the System DSN set up to allow the load process to export data
 to the SQL server.
ExportSource=VaultCMIS

; credentials used to log into the SQL server for the export process.
ExportUser=vaultodbc
ExportPassword=<snip>

; target table name to insert records into
ExportTable=Vault.dbo.Collections
```

```
; export fields (numbered consecutively from 1)
; Syntax is SQL column name, Vault property name, data type
; store date for CMIS as a date field
ExportField1=CmisDate,doc.date,d
ExportField2=VaultAccount,doc.account,s
; store Vault date as a string
ExportField3=VaultDate,doc.date,s
ExportField4=VaultFile,int.file,s
ExportField5=VaultOffset,int.pointer,s
ExportField6=VaultName,doc.name,s
ExportField7=CmisId,int.hash,s
; Collections related fields
ExportField8=VaultFileName,file.name,s
ExportField9=VaultType,doc.type,s
ExportField10=VaultFileSize,file.size,n
ExportField11=VaultMode,int.modes,n
```

Vault CMIS Connector planning

The next step is to create the Vault CMIS Connector properties file. For this scenario, the file would look like:

<pre>{
 "databases": {
  "sql1" : {
  "driver":
"com.microsoft.sqlserver.jdbc.SQLServerDriver",
 	
  "url":
"jdbc:sqlserver://127.0.0.1:1433;databaseName=..."
 	
  }
 },</pre>
	Define the SQL database connections. The driver specifies the type of database and the URL is the JDBC connections string.

<pre>"stores": {
  "vault1" : {
  "host": "vault1.local.pvt",
  "port": 6003
  }
},
 	
"vaultDatabases": {
  "db1" : {
  "store": "vault1",
  "name": "collection1"
 	
  }
}</pre>
	Define the stores (Vault server access) and the Vault databases to be used by the repositories. Note that the vaultDatabases section uses the optional "name" attribute to define the name of the Vault database (collection1) at the Vault server.

<pre>"repositories": {
 "collection": {
  "database": "sql1",
  "table": "Vault.dbo.Collections",
  "vaultDatabase" : "db1",</pre>
	Start defining a specific repository. The name here is the repository id in CMIS. Set the SQL database and vaultDatabase

	Set the user visible name and description for the repository.

<pre>"info": {
   "description": "Demo Collection Data",
   "name": "Demo Collection"
},</pre>	

<pre>"types": {
   "cmis:document": {
      "properties": [</pre>
	Begin declaring the properties for CMIS base properties.

<pre>   {
         "cmisName": "cmis:objectId",
 	
          "columnName": "CmisId"
        },</pre>
	Define the CMIS object id and link it to the CmisId column in the database.

<pre>   {
          "cmisName": "cmis:creationDate",
          "columnName": "CmisDate"
        },
        {
          "cmisName": "cmis:lastModificationDate",
          "columnName": "CmisDate"
        },</pre>
	Set creation and last modified date to the document date. We only need columnName as the rest is defaulted. The Vault ODBC setup will populate the CmisDate with the document date.

	Associate the document filename and size with CMIS equivalents.

<pre>{
"cmisName": "cmis:name",
"columnName":
"VaultFileName",
"vaultName": "file.name"
}
 	
{
"cmisName":
"cmis:contentStreamLength",
"columnName":
"VaultFileSize",
"vaultName": "file.size"
}</pre>	

<pre>{
"cmisName":
"custom:account",
"columnName":
"VaultAccount",
"description": "The account
for the document",
"displayName": "Vault
Account",
"vaultName": "doc.account"
},</pre>
	Define the account number property and expose it to CMIS clients as a custom property. Account is also required to render the document (vaultName attribute)

<pre>{
"cmisName": "custom:name",
 	
"columnName": "VaultName",
 	
"description": "The customer
name for the document",
"displayName": "Vault
Customer Name",
"vaultName": "doc.name"
},</pre>
	Define the customer name and expose it to CMIS clients as a custom property.

	Define the remaining Vault properties besides account (date/file/offset) that are required to render collections documents from the Vault CMIS Connector. These are not exposed to the CMIS clients (no cmisName attribute).

```
  {
     "columnName": "VaultFile",
     "description": "The job
containing document",
     "displayName": "Vault File",
     "vaultName": "int.file"
  },

  {
     "columnName": "VaultOffset",
    "description": "The offset of the
 document record",
     "displayName": "Vault Offset",
     "vaultName": "int.offset"
  },

  {
     "columnName": "VaultDate",
    "description": "The date of the
document",
    "displayName": "Vault Date",
    "vaultName": "doc.date"
 },
 {
    "columnName": "VaultType",
    "description": "The filetype of
the document",
    "displayName": "Vault Type",
    "vaultName": "doc.type"
 },

{
    "columnName": "VaultMode",
    "description": "The mode of the
document",
    "displayName": "Vault Mode",
    "vaultName": "int.modes"
 }]

             }
          }
       }
   }
}
```

5 - Vault CMIS authentication configuration

Vault CMIS Connector authentication is driven by the connectors `web.xml` file. This file follows the Java EE standard deployment descriptor file via the web application's web.xml, which describes how URLs map to servlets, which URLs require authentication, and other information.

In this section

Overview

The `web.xml` file must reside in the application's WAR under the `WEB-INF` directory in the web application root.

This declarative JEE Servlet security model requires users to be authenticated and in the right roles before they can access secure resource. The below describes the deployment descriptors inside the web.xml that will be exposed for the JEE container (for example, WebSphere Application Server, Tomcat).

Security constraints

The `<security-constraint>` element is used to define the access privileges to a collection of resources using their URL mapping.

The following sub elements can be part of a security-constraint:

- Web resource collection (web-resource-collection): A list of URL patterns and/or HTTP operations you want to constrain
- Authorization constraint (auth-constraint): Specifies the role names to be constrain
- User data constraint (user-data-constraint): whether the web request needs to be received over a protected transport such as SSL

In this example, we are setting up access control for the GET verb and leaving the other verbs (POST,etc.) unprotected. In a practical scenario, you would usually omit the http-method altogether to protect all verbs.

```
<security-constraint>
              <!-- control access to everything in our context -->
              <web-resource-collection>
                     <web-resource-name>Vault CMIS
Connector</web-resource-name>
                     <url-pattern>/*</url-pattern>
              <http-method>GET</http-method>
              </web-resource-collection>
              <!-- require user to have vaultcmis role -->
              <auth-constraint>
                     <role-name>vaultcmis</role-name>
              </auth-constraint>
 <!-- enable HTTPS -->
              <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
      </user-data-constraint>
</security-constraint>
```

Authentication method

The <login-config> element specifies the user authentication method used to gain access to web content.

The supported methods are:

BASIC - This is the easiest authentication method, which requires the users to a user name and password to authenticate before accessing secure resources.

CLIENT-CERT – This authentication method is typically used for identifying the user by his X.509 certificate. This is a special usage of the Secure Sockets Layer (SSL) where both the server and the user are identified by their own certificates. As a result, this is a very strong form of authentication.

FORM - form-based authentication. Similar to BASIC authentication, but using a custom login and error page. The path to the login page and the login error page must be specified.

Example

```
<!-- Define the Login Configuration for this Application -->
<!-- set authentication mechanism to basic -->
<login-config>
<auth-method>BASIC</auth-method>
 </login-config>
```

Security roles

The <security-role> element declares the role names used in Vault CMIS Connector and made available to the JEE container.

Example

```
<!-- Possible roles that can be mapped in app server -->
<!-- vaultcmis Role -->
<security-role>
<role-name>vaultcmis</role-name>
</security-role>
<!--All Roles -->
<security-role>
```

```
<role-name>*</role-name>
</security-role>
```

Once you have established the roles required (if any), the configuration JSON file must update to determine how those roles map to repository access in the security sections. See "Create the properties file for the connector".

Default authorization setup

The Vault CMIS Connector comes with predefined roles (vaultcmis1 through vaultcmis10). If application security is enabled on the application server, a user/group must be assigned to anyone of these roles to even access the Vault CMIS Connector via the container. Further, the Vault CMIS Connector properties file (JSON) can be configured to have the Vault CMIS Connector manage the role on the connector, or repository level via one of the predefined roles.

For WebSphere Application Server, the role setup is accessed by the Management console.

Under Applications/Application Types/WebSphere Enterprise Applications, select the Vault CMIS Connector (usually named `vault-cmis-`...). In "**Detailed Properties**", select "**Security role to user/group mapping**"

1. Select the Role, and click **Map User** or **Map Groups**
2. Select User or Groups as required, and save.

The authorization layout is:

```
<security-constraint>
           <!-- control access to everything in our context -->
           <web-resource-collection>
                  <web-resource-name>Vault CMIS
Connector</web-resource-name>
                  <url-pattern>/*</url-pattern>
           </web-resource-collection>

           <!-- require user to have a vaultcmis role -->
           <auth-constraint>
                     <role-name>vaultcmis1</role-name>
                     <role-name>vaultcmis2</role-name>
                     <role-name>vaultcmis3</role-name>
                     <role-name>vaultcmis4</role-name>
                     <role-name>vaultcmis5</role-name>
                     <role-name>vaultcmis6</role-name>
                     <role-name>vaultcmis7</role-name>
                     <role-name>vaultcmis8</role-name>
                     <role-name>vaultcmis9</role-name>
                     <role-name>vaultcmis10</role-name>
           </auth-constraint>
```

```
    </security-constraint>

    <login-config>
       <auth-method>BASIC</auth-method>
    </login-config>
<!-- Possible roles that can be mapped in app server -->
    <security-role>
       <role-name>vaultcmis1</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis2</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis3</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis4</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis5</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis6</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis7</role-name>
    </security-role>
    <security-role>
       <role-name>vaultcmis8</role-name>
    </security-role>
    <security-role>
        <role-name>vaultcmis9</role-name>
    </security-role>
    <security-role>
        <role-name>vaultcmis10</role-name>
    </security-role>
```

6 - Installing Vault CMIS Connector under WebSphere 8.5.5

The Vault CMIS Connector runs as an application under WebSphere 8.5.5. The Java version required is IBM WebSphere SDK Java Technology Edition 7.

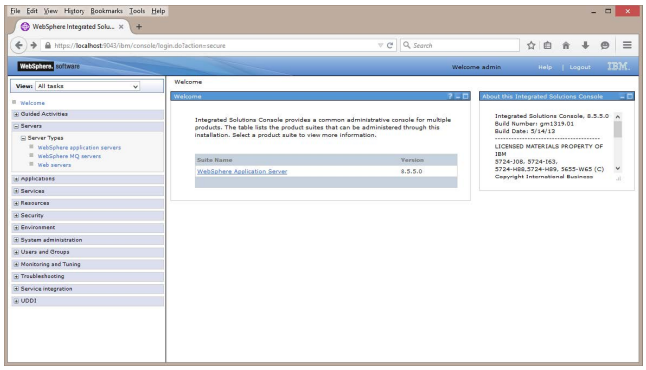
In this section

Overview

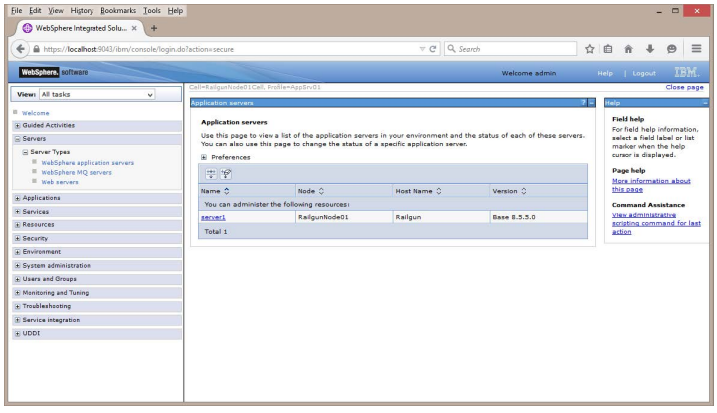
WebSphere 8.5.5 and IBM WebSphere SDK Java Technology Edition 7, must be installed on the platform using the IBM Installation Manager and then must be set up using the WebSphere management console as follows:

Set WebSphere to default to Java 7

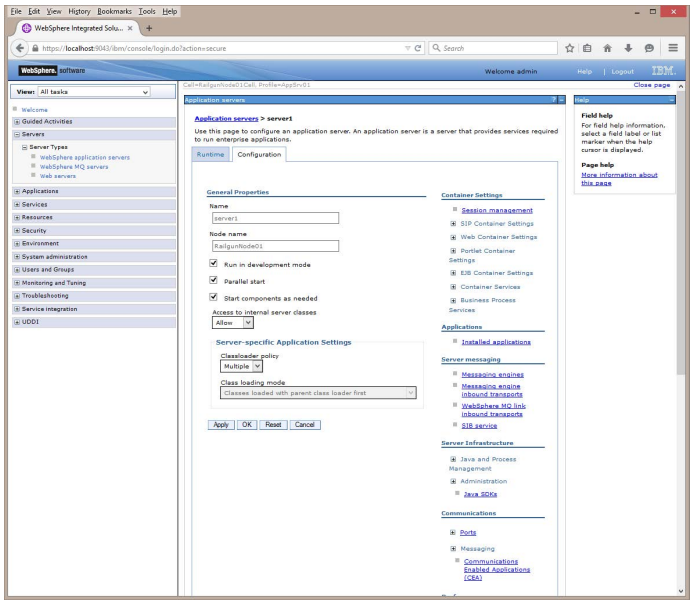
1. In the left hand pane, expand the **Servers** option.
2. Expand **Server Types**.
3. Click **WebSphere application servers**.



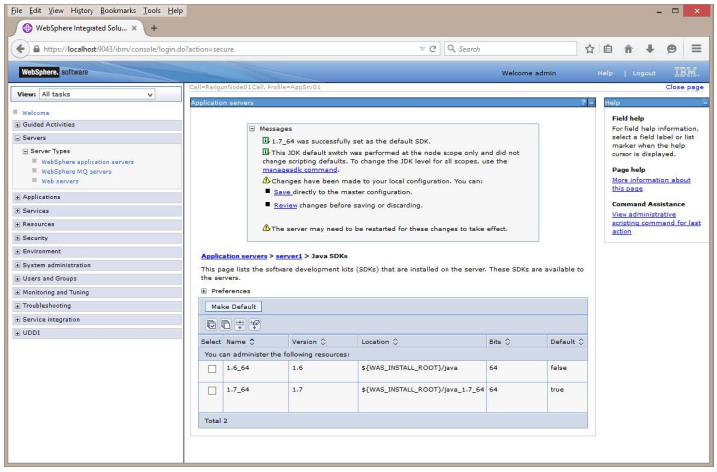
4. In the main pane, select the sever (server1 in this example).



5. Under **Server Infrastructure**, click **Java SDKs**.



6. Select the **1.7 Java SDK** and click **Make Default**.

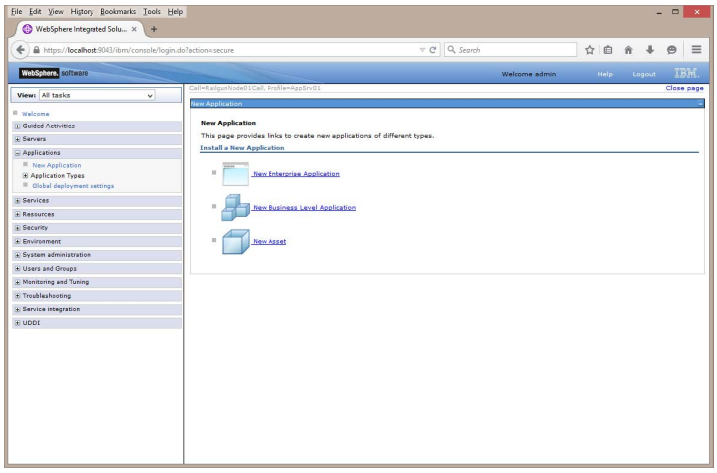


7. In the **Messages** box, click **Save**.

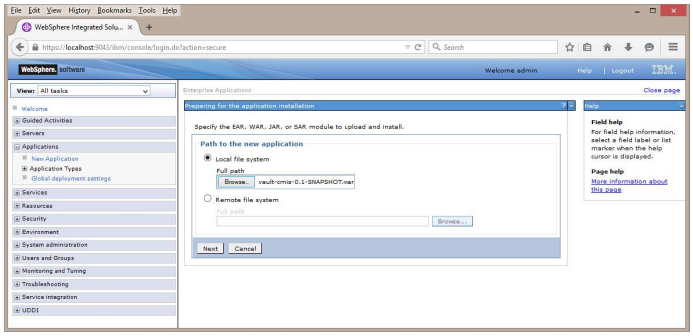
The next step is to install and configure the Vault CMIS application. Note that you may need to alter the web.xml of the installation war file to set up available roles. Under WebSphere, the web.xml can also be accessed and modified after the installation but this practice should be used with caution as changes will be lost upon and reinstallation of the Vault CMIS application.

Installing Vault CMIS Connector

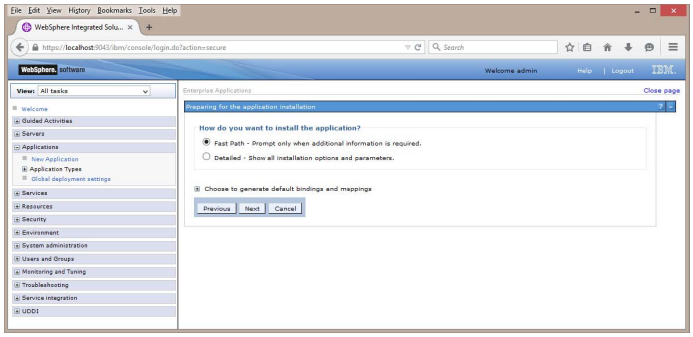
1. In the left hand pane, expand **Applications** and click **New Application**.
2. In the main pane select **New Enterprise Application**.



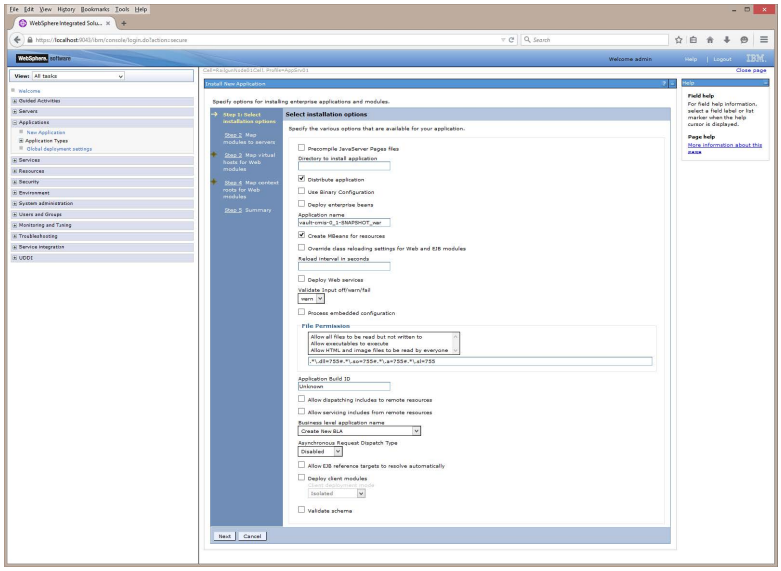
3. Browse to the location of the Vault CMIS.WAR file and select it.
4. Click **Next**.



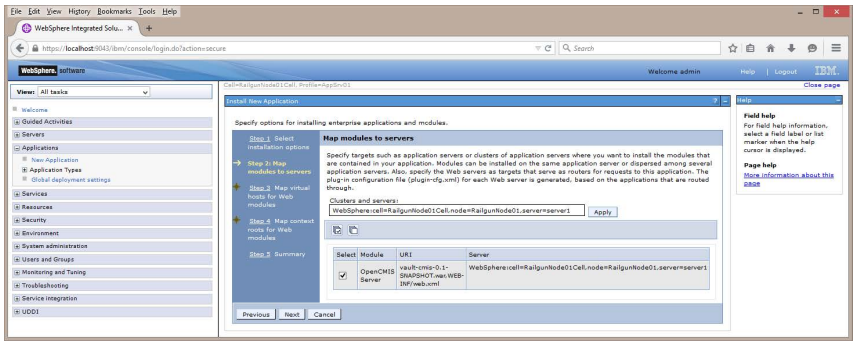
5. Select **Fast Path**, and click **Next**.



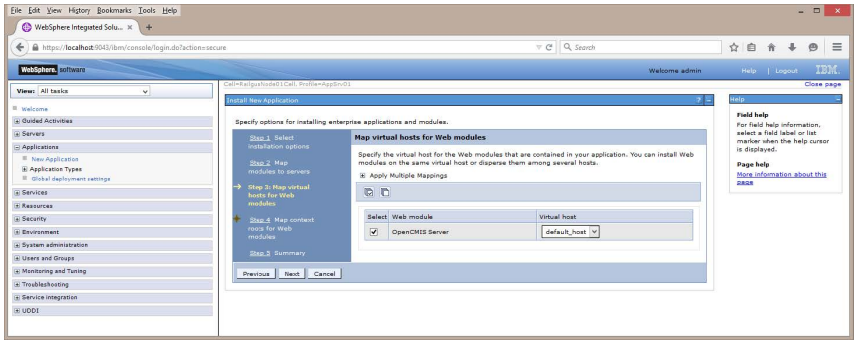
6. Click **Next**.



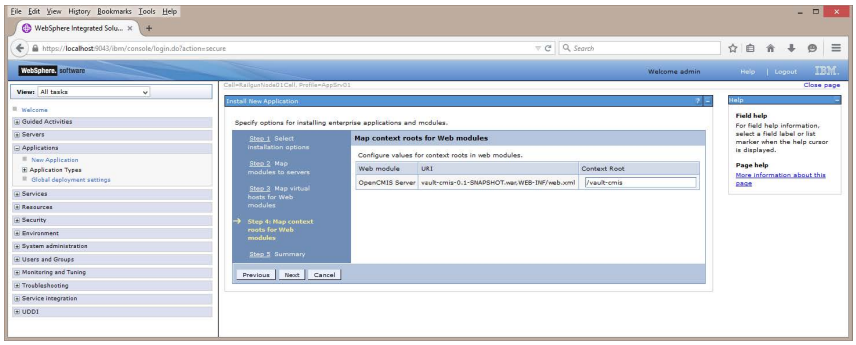
7. Select the CMIS module and click **Next**.



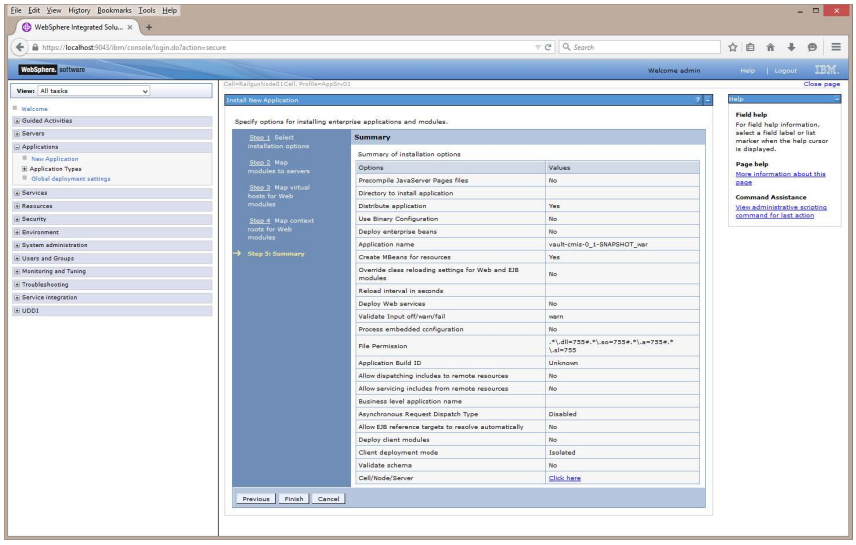
8. Select the CMIS module and click **Next** again.



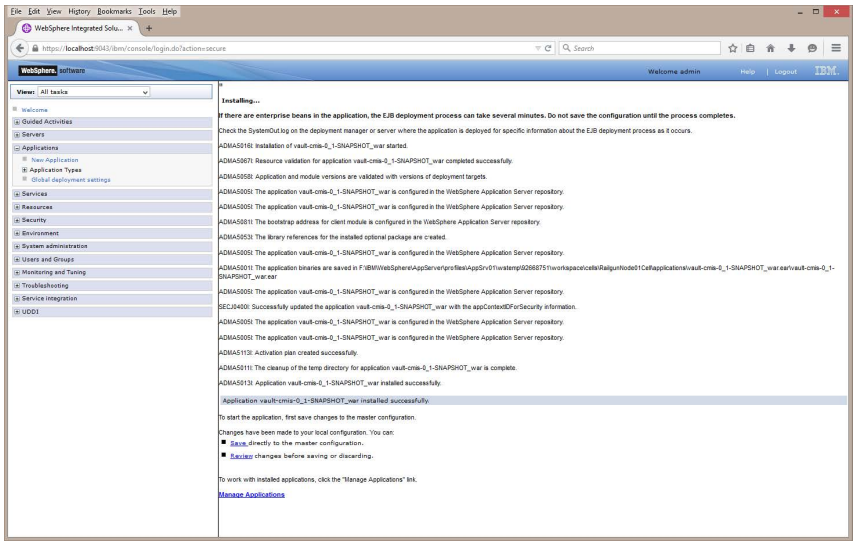
9. Enter the context root /vault-cmis and click **Next**.



10. Review the summary and click **Finish**.

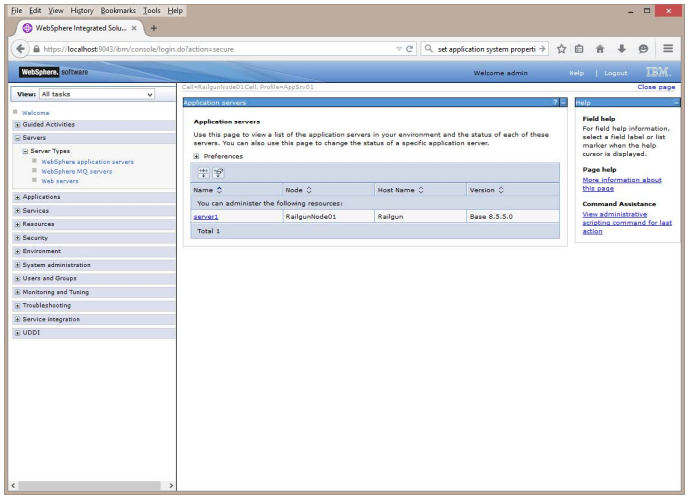


11. In the main pane, click **Save**.

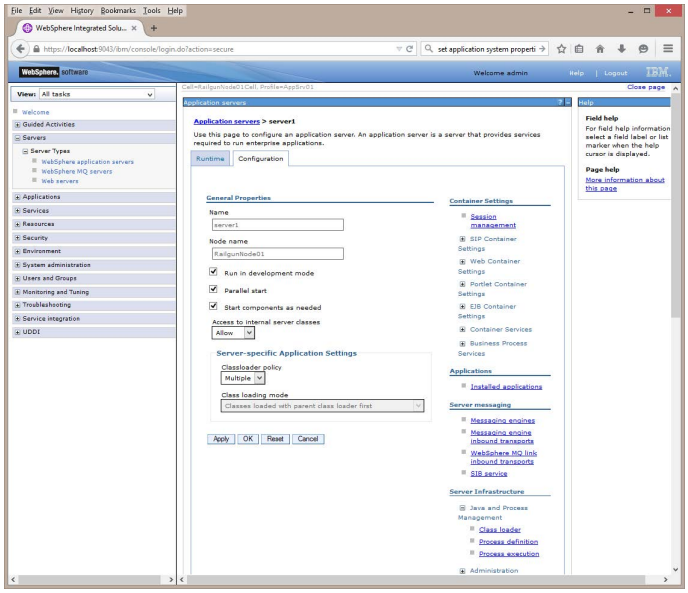


Set Vault CMIS Configuration location

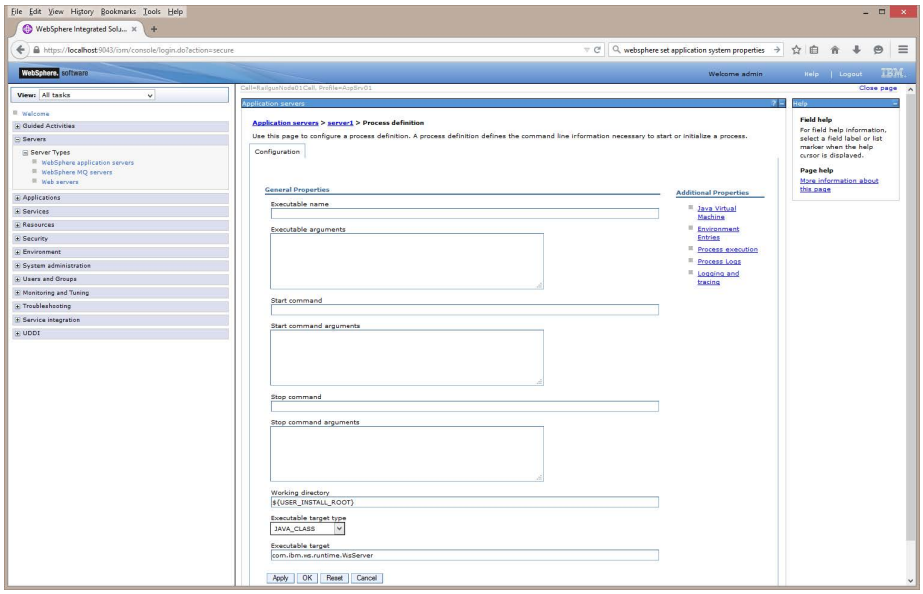
1. In the left pane, expand **Servers**.
2. Expand **WebSphere application servers**.
3. Select the server (server1 in this example).



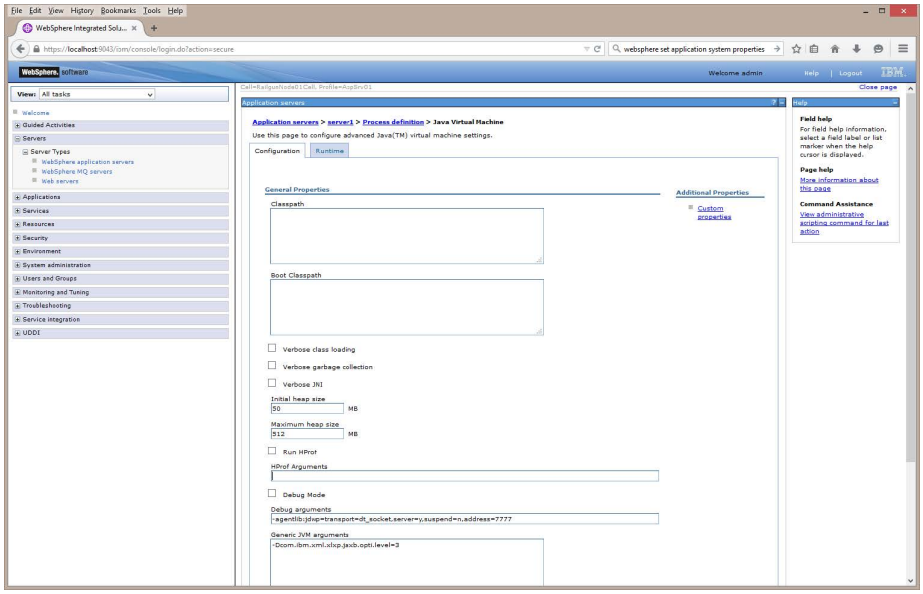
4. Under **Server Infrastructure**, expand **Java and Process Management**.
5. Click **Process definition**.



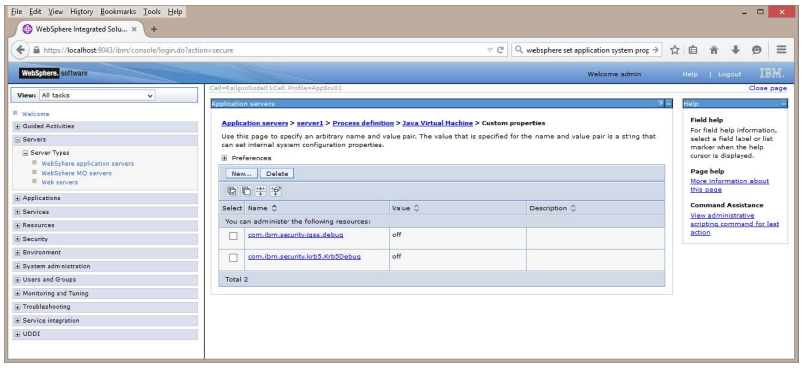
6. Under **Additional properties**, click **Java Virtual Machine**.



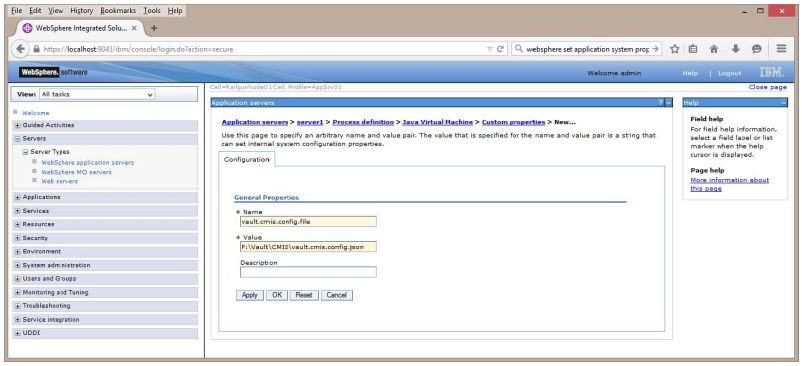
7. Under **Additional properties**, click **Custom properties**.



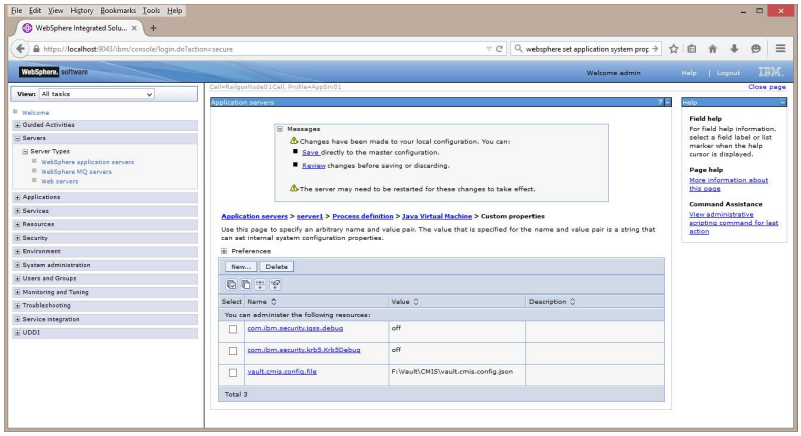
8. Click **New**.



9. Set the Name to **vault.cmis.config.file**.
10. Set the value to the full path of the Vault CMIS configuration file.
11. Click **OK**.



12. In the **Messages** area, click **Save**.



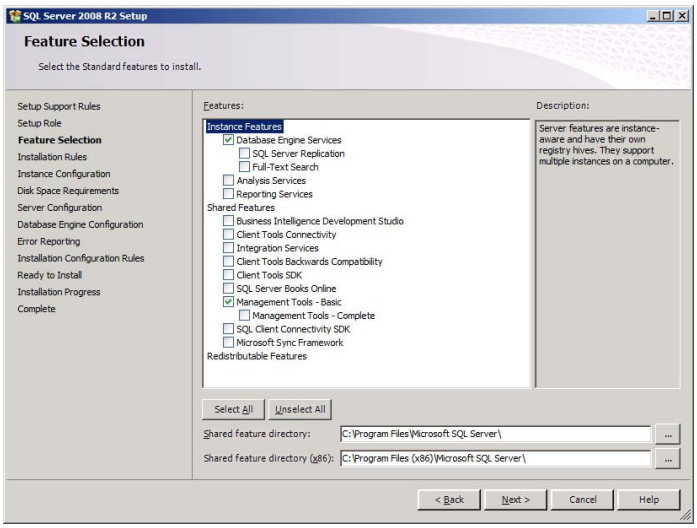
Note: If you are using an encrypted password, you will need to add a custom property for `vault.default.key`. See "Setting the default key file for Java environments" in the "Vault Customizing Guide" for further details.

7 - Configuring SQL Server for Vault ODBC Export

In this section

Installing SQL Server

Install the SQL server database engine and basic tools:



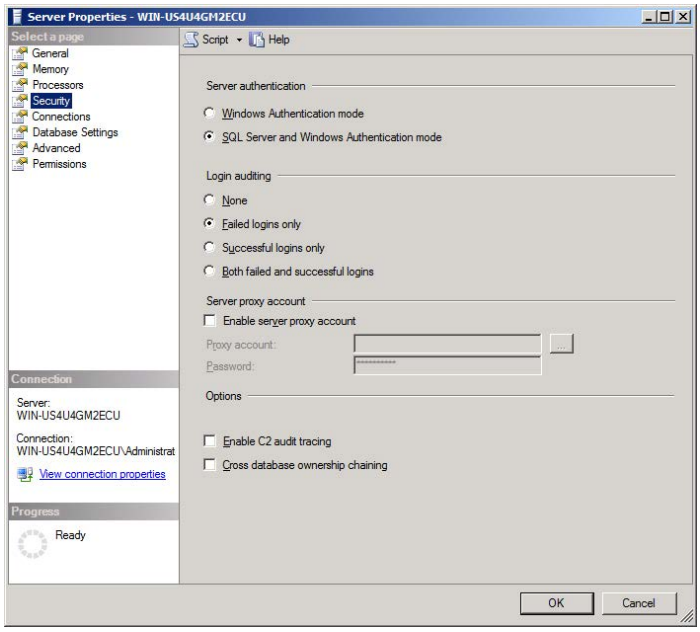
Afterwards, make sure the system and SQL server are up to date on patches.

It is recommended to enable Windows Update to check for updates for Microsoft products.

Enabling mixed accounts

The Vault CMIS Connector and ODBC export processes normally log into the database server using database rather than Windows logins. To permit this, you need to enable mixed mode authentication.

- Mixed mode can be enabled during the Database Engine Configuration stage of the install process.

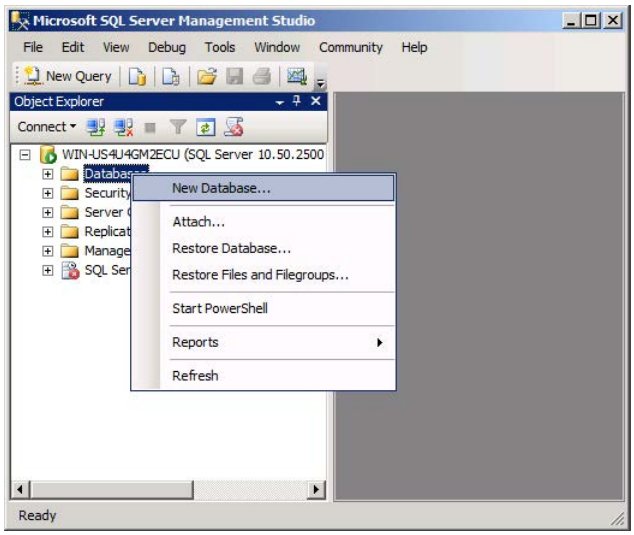


- It can also be enabled after installation using the Microsoft SQL Server Management Studio.
- To enable mixed mode, right click the server in the object explorer and choose **Properties**. Then select the **Security** page.

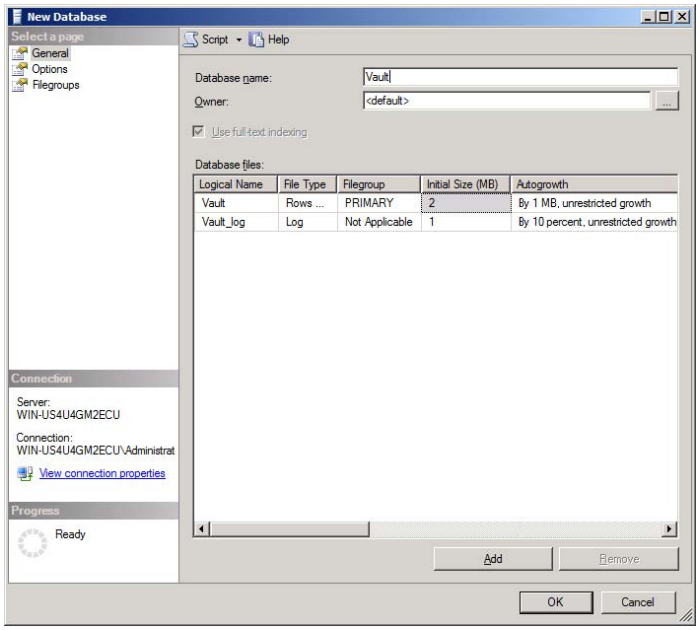
Create database

You will need to create a database to hold the table used to link the CMIS Connector and Vault.

1. In Microsoft SQL Server Management Studio, expand the server in the object explorer, right click **Databases** and choose **New Database**.



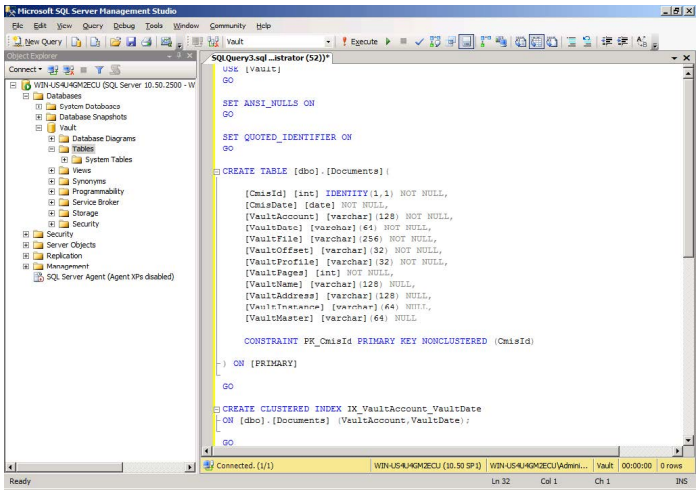
2. Give the new database a name, and click **OK**.



Create document table

Once the database has been created, you'll need to create the document table that links Vault and CMIS. This can be done via the Microsoft SQL Server Management Studio or from the command line.

1. In Microsoft SQL Server Management Studio, click **New Query**. Copy and paste the table creation script into the new window, and click **Execute**.

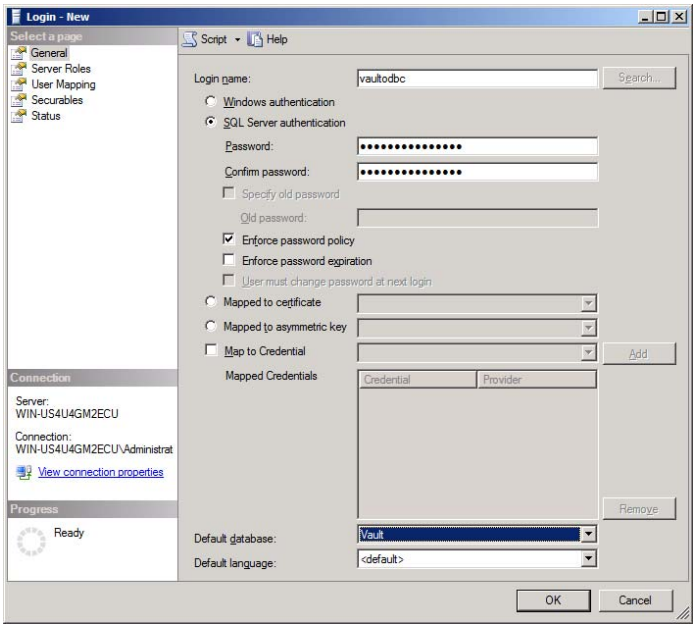


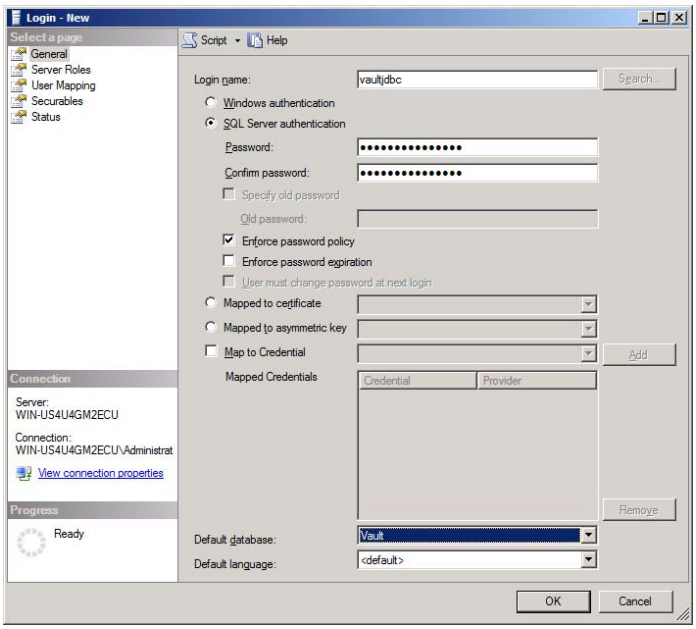
2. Alternately you can use sqlcmd to run the creation script directly from the command line or a batch file.

```
sqlcmd -i create-table.sql
```

Create logins

Create users for the connections from Vault (via ODBC) and the connector (via JDBC). Make sure to disable password expiration.

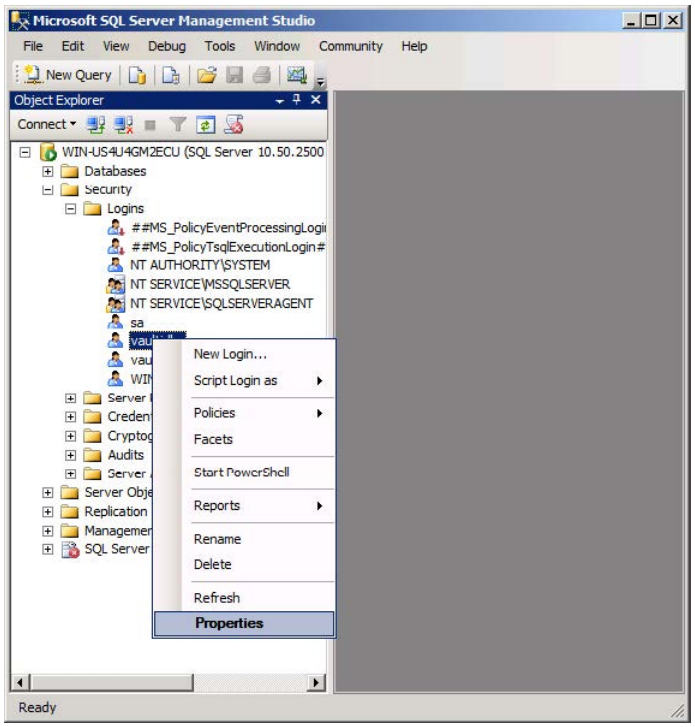




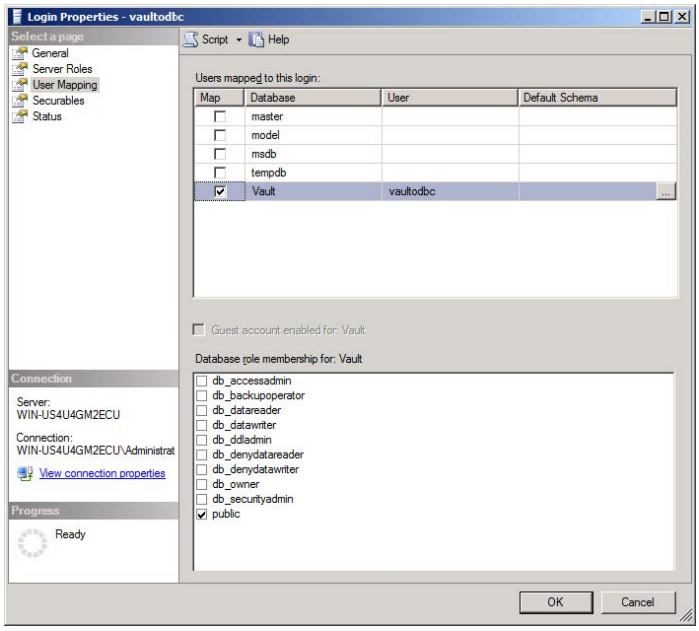
User mapping

For each of the new users, enable access by mapping the users to the new database:

1. In Microsoft SQL Server Management Studio, navigate to **Security\Logins** in the object explorer. Right click the user, and choose **Properties**.



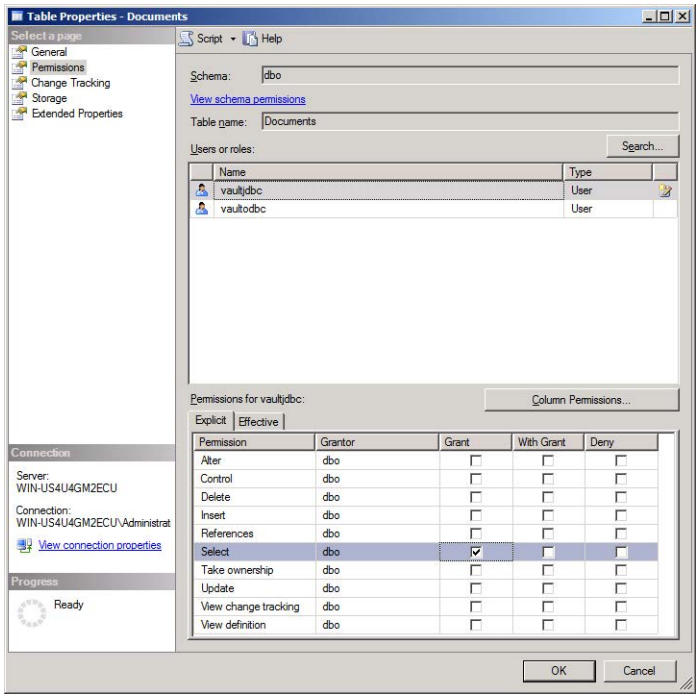
2. Select the **User Mapping** page. Click the **Map** check box for the new database, and click **OK**.



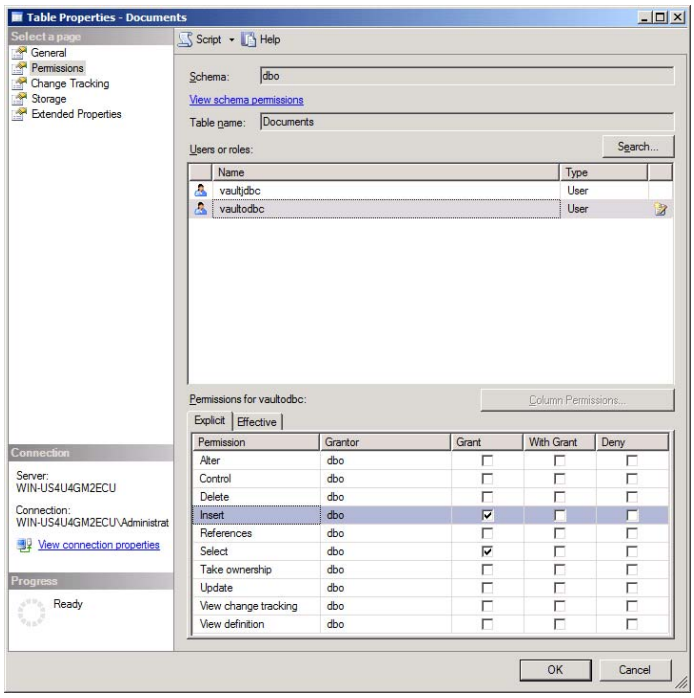
Permissions

Set the Vault user permissions.

1. On the **Permissions** page, click **Search** to find and add users.
   - Vault CMIS Connector user needs select access to the document table.
   - The user may also need Delete access if the ODBC driver is set up to delete records when documents are removed from Vault.



2. Vault OBDC Export user needs select and insert access to the document table as a minimum. You may also have to enable delete if you are using the Vault Export Delete feature.



Enable TCP access

In some cases the default SQL Server installation will not have TCP/IP enabled.

For example, SQL Server Express typically has networking disabled.

Open SQL Server Configuration Manager and expand the **SQL Server Network Configuration** item. Select the **Protocols** item. Verify that TCP/IP is set to Enabled.

> **Note:** You can right click **TCP/IP** and choose **Properties**. This configuration will show you the ports being used, normally 1433.

Firewall configuration

Windows Firewall may block port 1433 if Vault or the CMIS Connector are on other machines. Windows Firewall may also block port 8080 (or 8443 if HTTPS is enabled). You may have to configure new firewall rules for these ports.

8 - Vault Configuration for ODBC Export to a SQL Server

In this section

Creating a data source

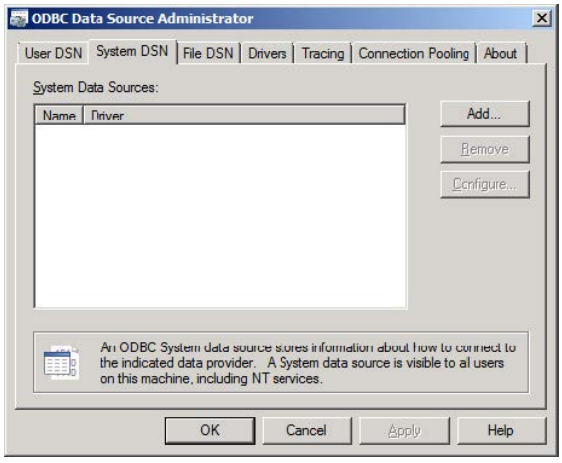
The Vault loader process is responsible for exporting data to an external data source via ODBC. Typically this is done by defining a System Data Source Name (DSN) on the machine running Vault. Since the loader process runs as a 32-bit process, you need to make sure to create the DSN with the 32-bit ODBC Data Source Administrator tool.

If the SQL Server is installed on a different machine, you may need to install the SQL Server ODBC driver on the Vault machine.

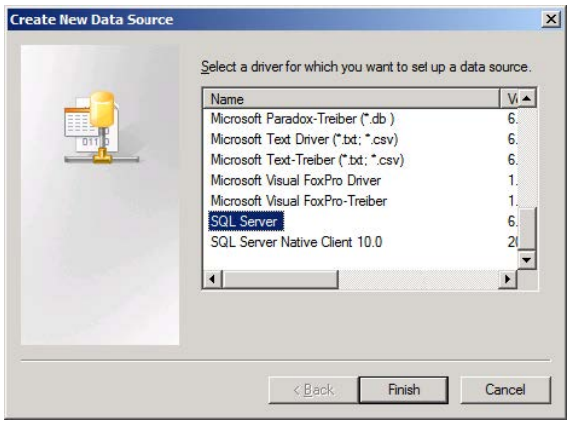
```
URL: https://www.microsoft.com/en-ca/download/details.aspx?id=36434
File: 1033\x86\msodbcsql.msi
```

To configure the data source, start by running Export:

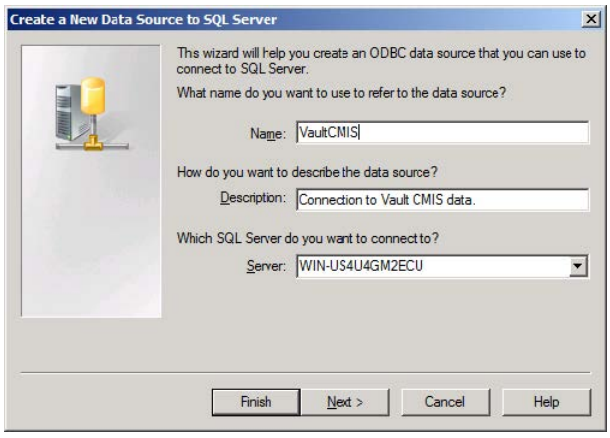
1. Start by switching to the **System DSN** tab and clicking **Add...**



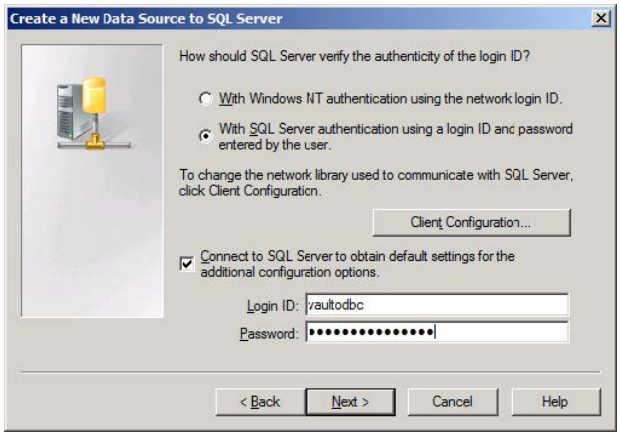
2. Select the SQL Server driver from the list.



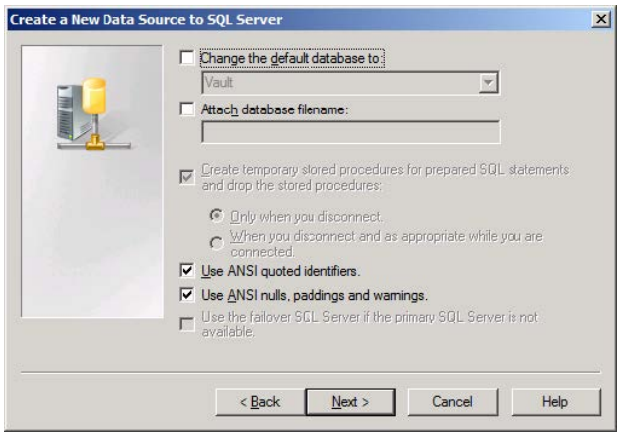
3. Enter a name and description for the Vault CMIS database source.
4. Select the SQL Server hosting the Vault connector table.



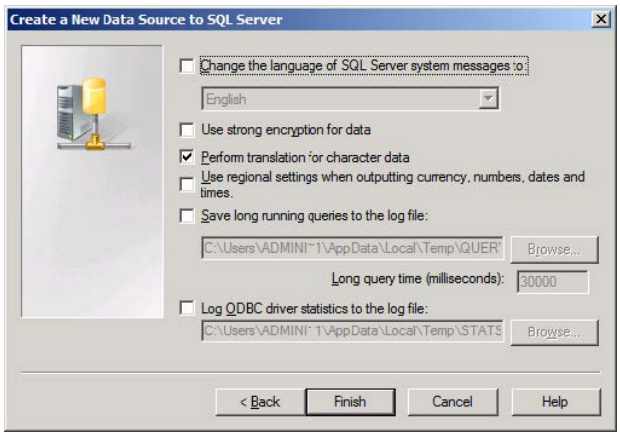
5. Configure the data source to use mixed authentication.
6. Connect to the SQL Server using the ODBC export user created earlier.



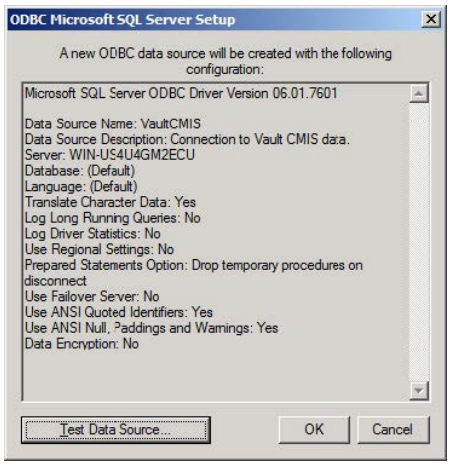
7. Click **Next…**



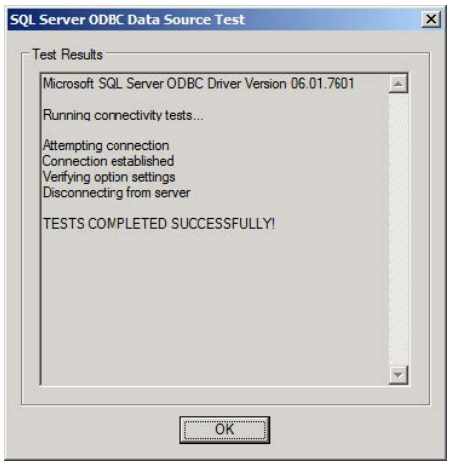
8. Click **Next…**



9. Click the **Test Data Source** button to verify the connection.



10. Click **OK** when done.



9 - Enabling ODBC export

In this section

Profile export settings

Within Vault, the properties that should be exported to the connector table need to be defined in the profile export settings. This uses the existing ODBC export feature which is available when running Vault on Windows. The "Vault Customization Guide" covers the settings in more detail.

The following example demonstrates how the connection and property settings are configured:

```
server\profiles.ini

[DemoAFP]
…

ExportEnable=1
ExportSource=VaultCMIS
ExportUser=vaultodbc
ExportPassword=<snip>
ExportTable=Vault.dbo.Documents
ExportField1=CmisDate,doc.date,d
ExportField2=VaultAccount,doc.account,s
ExportField3=VaultDate,doc.date,s
ExportField4=VaultFile,int.file,s
ExportField5=VaultOffset,int.pointer,s
ExportField6=VaultProfile,int.profile,s
ExportField7=VaultPages,doc.pages,n
ExportField8=VaultName,doc.name,s
ExportField9=VaultAddress,doc.address,s
ExportField10=VaultInstance,docInstanceID,s
ExportField11=VaultMaster,doc.guid,s
```

The ExportSource is the name of the System DSN created earlier. The ExportUser and ExportPassword are for the ODBC export login created earlier.

The ExportField settings list the target column name and the Vault property name it should be populated with. The s/d/n indicate the type of the field: string, number or date.

Note that you may want to use profile inheritance to move export settings into its own named section. That would allow multiple profiles to share the same settings for example.

When the ODBC export process runs, the e2loaderd process will log messages along the lines of:

```
06:05:30 <sql1> begin export, file [20150429013130], source [VaultCMIS],
 table [Vault.dbo.Documents]
         0    10   20   30   40   50   60   70   80   90   100
         |    |    |    |    |    |    |    |    |    |    |
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
06:05:41 <sql1> end export, added [37000], skipped [0]
```

Manually triggering ODBC export

You can manually trigger SQL for jobs using a script provided in the server\tools directory. It takes file pattern for jobs to trigger.

```
server
tools\vaultflags.bat sql 2015*
```

Note that the script would need to be modified if paths have been redirected to non-default locations.

Notices

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com