IDOL

Software Version 12.11

Document Security Administration Guide

Document Release Date: February 2022
Software Release Date: February 2022

Legal notices

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for updated documentation, visit https://www.microfocus.com/support-and-services/documentation/.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that Micro Focus offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- View information about all services that Support offers
- Submit and track service requests
- Contact customer support
- Search for knowledge documents of interest
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts

Many areas of the portal require you to sign in. If you need an account, you can create one when prompted to sign in.

About this PDF version of online Help

This document is a PDF version of the online help, and is provided so you can easily print multiple topics or read the online help. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Part I: Overview

This section provides an overview of Micro Focus IDOL document security and describes how to configure the components involved in security.

Chapter 1: Introduction

This section provides an overview of Micro Focus IDOL document security.

Security Overview

Micro Focus provides the software infrastructure that automates operations on unstructured information. This software infrastructure is based on IDOL Server.

Document security protects the information that you index into IDOL Server.

Your organization is likely to store information in many repositories. Many of these repositories have security features that apply permissions to files, so that they only authorized personnel can view them. Document security ensures that when you index information into IDOL Server, IDOL continues to enforce these permissions. In response to a query, IDOL returns only documents that a user is permitted to view.

IDOL includes the following features to protect your data:

- **User authentication**. At the front end, users must log on before they can query IDOL. IDOL can authenticate users against an existing directory service, such as Microsoft Active Directory.
- **Document security**. IDOL checks each document that is returned in response to a query. If the user who submitted the query does not have permission to view the document, the document is removed from the query results before the results are returned.
- **Secure communications**. You can encrypt the communications between ACI servers and any applications that use the Micro Focus IDOL ACI API.

User Authentication

To access front-end applications, users must log on by providing their credentials. If you build your own front-end applications, you can use the IDOL API to customize the logon process.

IDOL supports authentication against industry standard systems, including:

- Windows NT logon.
- LDAP authentication.

- Other third-party authentication (for example, Lotus Notes).

A single logon allows a user to access all the systems for which they have permission. For example, if a user submits a query they receive all documents that they are permitted to view, even though these documents originated from different data repositories. The IDOL Server Community component enables you to store and update user security details for this purpose.

In addition, you can use IDOL in a single sign-on environment that uses Kerberos for authentication.

Document Security

Document security ensures that users can access only those documents for which they have the necessary permissions.

When a user logs on to a front-end application and is authenticated successfully, the IDOL Community component returns an encrypted security string to the front-end application. This string identifies the user and contains information about their group memberships. The front-end application must include this security string in every subsequent query it sends to IDOL.

When a user submits a query, the IDOL Content component compares the user's information with the permissions on each document. IDOL offers two ways of doing this, *Mapped Security* and *Unmapped Security*.

> **NOTE:** Micro Focus strongly recommends using Mapped Security, because IDOL can return query responses significantly faster than when Unmapped Security is used.

Mapped Security

In the *Mapped Security* architecture, IDOL determines whether a user is permitted to view a document by comparing the user's security details against an Access Control List (ACL) that has been added to the document.

With Mapped Security, when connectors fetch information from data repositories they add an encrypted Access Control List (ACL) to a metadata field in each document. The ACL contains information about which users and groups are permitted to access the document. The documents, and therefore the ACLs, are indexed into IDOL Server.

A user might be allowed or denied permission to view a document because they are a member of a security group. This means that IDOL must consider group memberships, in addition to permissions, before it can determine whether a user can view a document. OmniGroupServer collects user and group information, and stores it, so that IDOL Server can access this information.

When a user queries IDOL Content, IDOL sends the user's security string and the documents that match the query to the Mapped Security Plug-In. The Mapped Security Plug-In compares the user's details to the ACL in each document, and determines which documents the user is permitted to view. IDOL only returns those documents in its response.

The advantage of this process is that IDOL can quickly respond to a query, because it does not need to connect to the original data repository to check the ACL for each document. The disadvantage is

that there might be a delay between the security settings changing in the original data repository and the information being updated in the IDOL index.

Mapped Security is suitable for most environments, particularly where the security settings for documents do not change often.

Unmapped Security

In the *Unmapped Security* architecture, IDOL determines whether a user is permitted to view a document by connecting directly to data repositories and checking the user's security details against the entitlement information of the documents that matched the query.

The connection between IDOL and the data repositories is made possible by IDOL's unmapped security libraries.

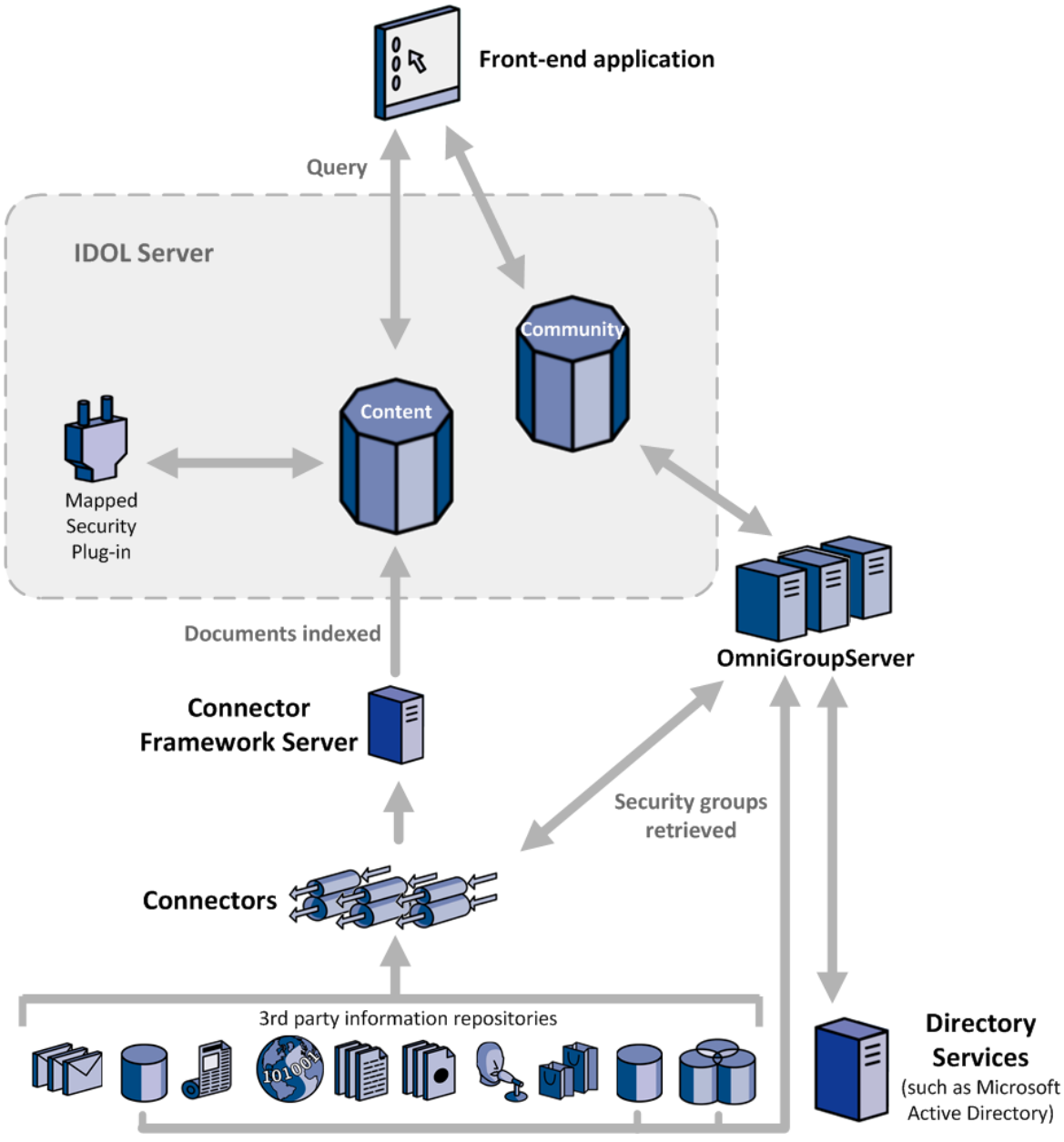
The advantage of unmapped security is that the security information is current. The disadvantage is that IDOL Server has to connect to the original data repositories to check permissions for each result document. This means that there can be a significant delay between a user submitting a query, and IDOL returning its response. For this reason, Micro Focus strongly recommends using Mapped Security.

Unmapped security is suitable for environments where the security settings for documents change frequently.

In some cases, you must also configure a group server. A group server is required because it is impossible for IDOL to retrieve group information from the repositories in a reasonable time. OmniGroupServer extracts user and group information and stores it so that it is available to IDOL immediately.

Mapped Security Example

The following diagram shows the components involved in a Mapped Security architecture:



Connectors extract information from third party repositories so that the information can be indexed into IDOL Server. The connector adds an Access Control List (ACL) to a metadata field in each document. The ACL describes which users and groups are permitted to view the document. CFS indexes the document into IDOL Server.

At the same time, OmniGroupServer retrieves group memberships from the third party repositories and from directories such as Active Directory. OmniGroupServer stores this information until it is needed. In some cases, where a repository uses its own system for storing users and groups, OmniGroupServer queries a connector to retrieve group information.

If the permissions set on a file in a repository are changed, the connector sends the update to CFS and the document's ACL is updated in IDOL Server. If a user's group memberships change, the group information is updated in OmniGroupServer the next time the group server synchronizes with the repository.

To use the front-end application, a user must log on. After authentication is successful, the front-end application sends a query to Community, to retrieve the user's security information. Community returns an encrypted `securityinfo` string that contains the names of the groups the user is a member of. The front-end stores the string, because it must be sent with all queries to IDOL server.

When a user does something in the front-end application that requires information from IDOL server (for example, starting a search), the front-end sends a query to IDOL server. IDOL Server (Content) runs the operation and sends the resulting documents and the user's security information to the Mapped Security plug-in. This compares the user's security information with the ACL of each document and returns the documents that the user is permitted to view. IDOL Server then returns these documents to the front-end.

Related Documentation

The following documents provide more details on IDOL and document security.

- *IDOL Server Administration Guide*

  IDOL Server is at the center of an IDOL infrastructure, storing and processing the data indexed into it. The *IDOL Server Administration Guide* describes the operations IDOL Server can perform with detailed descriptions of how to set them up.

- *IDOL Server Getting Started Guide*

  The *IDOL Server Getting Started Guide* describes how to install, configure, and run IDOL Server.

- *IDOL Expert*

  IDOL Expert provides conceptual overviews and expert knowledge of IDOL and its features and functionality. *IDOL Expert* contains information about setting up security across components.

- *IDOL Server Reference*

  The *IDOL Server Reference* describes the configuration parameters and actions that you can use with IDOL Server.

- *OmniGroupServer Reference*

  The *OmniGroupServer Reference* describes the configuration parameters and actions that you can use to set up OmniGroupServer.

- An appropriate connector guide (such as the *File System Connector Administration Guide* or

the *SharePoint Remote Connector Administration Guide*).

These guides explain in detail how to configure individual connectors.

Chapter 2: Configure IDOL Server

This section describes how to configure your IDOL Content and IDOL Community components to enable mapped security.

Introduction

To enable mapped security, configure content security and user security:

- **Content security**. Identifies the security type that applies to each document, and checks the Access Control List (ACL) of indexed documents against user security privileges.
- **User security**. Provides authentication and checks which security privileges users have in IDOL Server and third-party repositories. This includes the retrieval of user group information from a group server.

NOTE: This guide explains how to configure mapped security by editing the IDOL Content and IDOL Community component configuration files. If you have installed IDOL Server in unified mode, there is only one configuration file, named `IDOLServer.cfg`. The section names remain the same, except that:

- The `[Security]` section in the Community configuration file becomes `[UserSecurity]` in `IDOLServer.cfg`.
- The `[SecurityFields]` section in the Community configuration file becomes `[UserSecurityFields]` in `IDOLServer.cfg`.

Configure Content Security

To configure content security, complete the following steps:

- Configure Security Types, on the next page. Configure the types of security that you want to use.

- Identify the ACL Document Field, on the next page. Configure IDOL Content to read the access control list (ACL) from the correct document field, and ensure that the ACL does not appear in query results.
- Identify the Security Type, on page 17. Configure IDOL Content to identify the security type of documents that you have indexed.

Configure Security Types

To configure security types

1. Open the IDOL Content component configuration file.
2. In the `[Security]` section, set the `SecurityInfoKeys` configuration parameter to the path of your AES key file.

   ```
   [Security]
   SecurityInfoKeys=/path/to/aes.keyfile
   ```

   This is used to encrypt and decrypt the security strings that IDOL generates for each user. For this reason, the value of `SecurityInfoKeys` must be the same for each component that requires it (for example, Content, Community, and DAH must all use the same key file).

   If you need to generate a new key file, use the `autpassword` utility. Generate a key file in the same way as for encrypting passwords. For more information about using the `autpassword` utility, see Encrypt Passwords, on page 64.
3. List the security types that you want to use. For example:

   ```
   [Security]
   SecurityInfoKeys=/path/to/aes.keyfile
   0=NT_V4
   1=Notes_V4
   ...
   ```
4. Create a new configuration section for each of the security types. In each section, set the following configuration parameters:

Parameter	Value
`SecurityCode`	A unique number to use as an identifier for the security type.
`Library`	The name of the security library to use to check the security settings of documents that use this security type.
`Type`	The security type. Specify one of the security types listed in Security Types, on page 23.
`ReferenceField`	The name of the document field that stores the ACL for this security type. The default value of this parameter is `*/AUTONOMYMETADATA` (by default, connectors add the ACL to the `AUTONOMYMETADATA` field).

`EscapedEntries`	Non-alphanumeric characters in a security string that is passed to IDOL are usually percent-encoded, but the names in the ACL are not. If the user or group names in the security string can contain non-alphanumeric characters, set this parameter to **`true`**. This instructs IDOL Server to expect escaped information, and ensures that the names are correctly unescaped to perform security checks. The default for this parameter for most security types is `false`; it is usually necessary to set it to **`true`**.

For example:

```
[NT_V4]
SecurityCode=1
Library=C:\Autonomy\IDOLServer/IDOL/modules/mapped_security
Type=AUTONOMY_SECURITY_V4_NT_MAPPED
ReferenceField=*/AUTONOMYMETADATA
EscapedEntries=true
```

For more information about the configuration parameters that you can set to customize security, refer to the *IDOL Server Reference*.

Identify the ACL Document Field

To identify the document field that contains the ACL

1. In the IDOL Content configuration file, find the `[FieldProcessing]` section.
2. Create a new field processing rule to identify the document field that contains the ACL. To do this, set the following parameters:

Parameter	Value
`PropertyFieldCSVs`	The document field that contains the document's ACL.
`Property`	The name of the section in the IDOL Content configuration file that specifies the property to apply to the document. You can specify any name, but when you create the property, you must use the same name.

For example:

```
[FieldProcessing]
0=SetAclFields

[SetAclFields]
PropertyFieldCSVs=*/AUTONOMYMETADATA
Property=AclFields
```

3. In the `//---Properties---//` section of the configuration file, create a new property. Configure the property so that the field is identified as containing the ACL. Set the `HiddenType` parameter

to **true** so that the field is not displayed in query results. You must create the property using the same name that you specified in the Property parameter. For example:

```
//---Properties---//

[AclFields]
HiddenType=TRUE
ACLType=TRUE
```

Identify the Security Type

To configure IDOL Content to identify the type of security used by a document, set up a field processing rule. The field processing rule should read a document field that specifies the security type used by the document.

To identify the security type by document

1. In the IDOL Content configuration file, find the [FieldProcessing] section.
2. Create a new field processing rule to read the document field that specifies the security type. Set the following parameters:

Parameter	Value
PropertyFieldCSVs	The names of document fields that the rule should read.
PropertyMatch	The values that the fields specified by PropertyFieldCSVs must have for the field process to be applied.
Property	The name of the property to apply to the document. You can specify any name, but when you create the property, you must use the same name.

In the following example, IDOL looks for the document field SECURITYTYPE. If the field exists and the contains the value NT, IDOL applies the property SecurityNT_V4:

```
[FieldProcessing]
0=SetAclFields
1=DetectNT_V4Security
...

[DetectNT_V4Security]
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=NT
Property=SecurityNT_V4
```

3. In the //---Properties---// section of the configuration file, create a new property using the name you specified with the Property parameter. Then, set the SecurityType parameter to the type of security used by the document. The value of this parameter must refer to the name of a section in the configuration file that contains settings for that security type. For information about configuring security types, see Configure Security Types, on page 15. The types that you

can specify are listed in the [Security] section. For example:

```
//---Properties---//

[SecurityNT_V4]
SecurityType=NT_V4
```

Configure User Security

IDOL user security provides user authentication, and checks which security privileges users have in third-party repositories. This includes the retrieval of group information from OmniGroupServer.

TIP: The IDOL Community component manages a database of users. You can populate this database manually, or configure IDOL to populate the database from a third-party directory. For more information about users in IDOL, refer to the *IDOL Server Administration Guide*.

To configure user security

1. Open the IDOL Community component configuration file.
2. In the [Server] section, set the following parameter.

DeferLogin	To automatically add users to IDOL the first time they log on to a front end, set **DeferLogin=True**. IDOL is populated with user information from your configured security repositories, for example your LDAP directory. If you want to add users to IDOL manually, set **DeferLogin=False** and add users with the UserAdd action. For more information about adding users manually, refer to the *IDOL Server Administration Guide*.

3. In the [Security] section, list the security types that you want to configure. Start numbering from 0 (zero), for example:

   ```
   [Security]
   0=NT
   1=LDAP
   2=Notes
   ```

4. In the [Security] section, set the following parameters.

SecurityInfoKeys	The path of your AES key file. This is used to encrypt and decrypt the security strings that IDOL generates for each user. For this reason, the value of SecurityInfoKeys must be the same for each component that requires it (for example, Content, Community, and DAH must all use the same key file).

	If you need to generate a new key file, use the `autpassword` utility. Generate a key file in the same way as for encrypting passwords. For more information about using the `autpassword` utility, see Encrypt Passwords, on page 64.
`CheckEntitlement`	To authenticate users before returning a `securityinfo` string, set this parameter to **`true`**. Be aware that the default value of this parameter is **`false`**, which means that a `securityinfo` string can be obtained without authentication.
`DefaultSecurityType`	An integer that specifies the security repository to use to authenticate users when the `Repository` action parameter is not set in the `Security` or `UserRead` action. Using the values from the example above, you would set `DefaultSecurityType=0` for NT authentication and `DefaultSecurityType=2` for Notes authentication.
`SyncRolesFromGroups`	Set this parameter to **`true`** to synchronize roles from NT groups. This ensures that a user's permissions and NT groups are always in sync. The default value for this parameter is `false`.
`GroupServerParentRole`	If you set `SyncRolesFromGroups` to **`true`**, `GroupServerParentRole` allows you to specify the parent role to which IDOL server adds new roles that it creates. If you don't specify a parent role with `GroupServerParentRole`, IDOL Server adds the new roles that it creates to the top role in the hierarchy.

For more information about the configuration parameters that you can use, refer to the *IDOL Community Component Reference*.

5. Create a section for each of the security types that you listed. For example:

```
[NT]
CaseSensitiveUserNames=FALSE
CaseSensitiveGroupNames=FALSE
Library=./modules/user_ntsecurity
DocumentSecurity=TRUE
DocumentSecurityType=NT_V4
v4=true
SecurityFieldsCSVs=username,domain
GroupServerHost=123.45.6.7
GroupServerPort=3057
Domain=Autonomy

[LDAP]
Library=./modules/user_ldapsecurity
DocumentSecurity=FALSE
LDAPServer=ldap
LDAPPort=389
```

```
RDNAttribute=uid
Group=o=Company,ou=Users
...

[Notes]
...
```

The parameters in each section depend on the type of repository. You can set the following parameters:

Parameter	Description
`Domain`	(NT security only) If you are configuring NT security, specify the name of the NT domain to use.
`Library`	The path of the library to use to authenticate users. The authentication libraries that Micro Focus currently supplies are: • `user_autnsecurity`. Autonomy authentication. • `user_ntsecurity`. NT authentication. • `user_notessecurity`. Lotus Notes authentication. • `user_ldapsecurity`. LDAP authentication. Specify the library you want to use without the file extension.
`v4`	Set this parameter to **`true`** if the security section defines security for NT or Exchange data and you are using a version 4 security type.
`GroupServerHost`	The IP address of the machine on which your group server is located.
`GroupServerPort`	The ACI port of the group server.
`GroupServerParameters`	One or more parameters to send to the group server in addition to `username`. Separate multiple parameters with a comma (there must be no space before or after a comma).
`GroupServerPrefixDomain`	Set this parameter to **`true`** to prefix domain information to the user name when contacting the group server, so that you can handle users in different domains who have the same user name.
`GroupServerUserField`	The field that IDOL Community must read the user name from. Use this parameter in cases where the group server contains multiple user name fields (for example, a field that contains the full name and another field that contains a shortened name).

`CaseSensitiveUsernames`	A Boolean value that specifies whether user names for this security type are case sensitive. If you set this parameter to `false`, IDOL Server returns upper case user names.
`CaseSensitiveGroupNames`	A Boolean value that specifies whether group names for this security type are case-sensitive. If you set this parameter to `false`, IDOL Server returns upper case group names.
`DocumentSecurity`	Set `DocumentSecurity` to `True` if this user security repository corresponds to a document security module that you configured in the `[Security]` section of your IDOL Content component configuration file. The IDOL Community component uses this information to generate an appropriate security string for the users in this repository. If you set this parameter to `TRUE` you must also specify the name of the security module with the `DocumentSecurityType` parameter. Otherwise, set this parameter to **`false`** (for example, to use `LDAP` or `autonomy` security).
`DocumentSecurityType`	(If you have set `DocumentSecurity` to **`true`**). The name of the corresponding security module, as listed in the `[Security]` section of the IDOL Content component configuration file.
`SecurityFieldCSVs`	Specify one or more security fields needed for the security type. All the fields you specify with `SecurityFieldCSVs` must be listed in the `[SecurityFields]` section. Separate multiple values with a comma (there must be no space before or after a comma). For more information on required fields for your security types, see SecurityInfo Parameters, on page 68.

6. Save and close the configuration file.

Example Configuration

The following is an extract from an example IDOL Content component configuration file with a single security type configured. The highlighting shows the relationships between the different sections in the configuration file.

```
[Security]
SecurityInfoKeys=/path/to/aes.keyfile
0=NT_V4

[NT_V4]
SecurityCode=1
```

```
Library=C:\Autonomy\IDOLServer/IDOL/modules/mapped_security
Type=AUTONOMY_SECURITY_V4_NT_MAPPED
ReferenceField=*/AUTONOMYMETADATA
EscapedEntries=true


[FieldProcessing]
0=SetAclFields
1=DetectNT_V4Security

[SetAclFields]
PropertyFieldCSVs=*/AUTONOMYMETADATA
Property=AclFields

[DetectNT_V4Security]
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=NT
Property=SecurityNT_V4


//---Properties---//

[AclFields]
HiddenType=TRUE
ACLType=TRUE

[SecurityNT_V4]
SecurityType=NT_V4
```

The following is the corresponding IDOL Community component configuration file.

```
[Server]
DeferLogin=True

[Security]
SecurityInfoKeys=/path/to/aes.keyfile
DefaultSecurityType=0
CheckEntitlement=TRUE
SyncRolesFromGroups=true
0=NT

[NT]
Library=./modules/user_ntsecurity
GroupServerHost=123.4.5.67
GroupServerPort=3057
Domain=AUTONOMY
SecurityFieldCSVs=username,domain
DocumentSecurity=true
DocumentSecurityType=NT_V4
```

```
v4=true
CaseSensitiveUserNames=FALSE
CaseSensitiveGroupNames=FALSE


[SecurityFields]
0=username
1=password
2=group
3=domain
```

Security Types

IDOL supports the following security types:

AUTONOMY_SECURITY_DOCUMENTUM_MAPPED

AUTONOMY_SECURITY_FILENET_MAPPED

AUTONOMY_SECURITY_NONE

AUTONOMY_SECURITY_NOTES

AUTONOMY_SECURITY_NOTES_MAPPED

AUTONOMY_SECURITY_NT

AUTONOMY_SECURITY_NT_MAPPED

AUTONOMY_SECURITY_ODBC

AUTONOMY_SECURITY_ODBC_MAPPED

AUTONOMY_SECURITY_OPENTEXT_MAPPED

AUTONOMY_SECURITY_ORACLE

AUTONOMY_SECURITY_PCDOCS_MAPPED

AUTONOMY_SECURITY_SHAREPOINT_MAPPED

AUTONOMY_SECURITY_UNIX

AUTONOMY_SECURITY_UNIX_MAPPED

AUTONOMY_SECURITY_UNSUPPORTED

AUTONOMY_SECURITY_WWW

AUTONOMY_SECURITY_V4_DOCUMENTUM_MAPPED

AUTONOMY_SECURITY_V4_EROOM_MAPPED

AUTONOMY_SECURITY_V4_EXCHANGE_GRPS_MAPPED

AUTONOMY_SECURITY_V4_EXCHANGE_MAPPED

AUTONOMY_SECURITY_V4_GENERIC_MAPPED

AUTONOMY_SECURITY_V4_IMANAGE_MAPPED

AUTONOMY_SECURITY_V4_MSCMS_MAPPED

AUTONOMY_SECURITY_V4_NETWARE_MAPPED

AUTONOMY_SECURITY_V4_NOTES_MAPPED

AUTONOMY_SECURITY_V4_NT_MAPPED

AUTONOMY_SECURITY_V4_OPENTEXT_MAPPED

AUTONOMY_SECURITY_V4_PCDOCS_MAPPED

AUTONOMY_SECURITY_V4_SCS_MAPPED

AUTONOMY_SECURITY_V4_TRIM_CONTEXT_MAPPED

AUTONOMY_SECURITY_V4_UNIX_MAPPED

AUTONOMY_SECURITY_V4_WORKSITE_MAPPED

AUTONOMY_SECURITY_V4_WORKSITE_MP_MAPPED

NOTE: Security types that include `V4` in the name are version 4 security types, which enable IDOL server to check whether a user is allowed to see certain documents without communication with the security library. This enables faster query response times.

Configure Client Authorization

You can configure IDOL to authorize different operations for different connections.

Authorization roles define a set of operations for a set of users. You define the operations by using the `StandardRoles` configuration parameter, or by explicitly defining a list of allowed actions in the `Actions` and `ServiceActions` parameters. You define the authorized users by using a client IP address, SSL identities, and GSS principals, depending on your security and system configuration.

For more information about the available parameters, see the *IDOL Reference*.

IMPORTANT: To ensure that IDOL allows only the options that you configure in `[AuthorizationRoles]`, make sure that you delete any deprecated *`Role`*`Clients` parameters from your configuration (where *`Role`* corresponds to a standard role name, for example `AdminClients`).

To configure authorization roles

1. Open your configuration file in a text editor.
2. Find the `[AuthorizationRoles]` section, or create one if it does not exist.
3. In the `[AuthorizationRoles]` section, list the user authorization roles that you want to create. For example:

   ```
   [AuthorizationRoles]
   0=AdminRole
   1=UserRole
   ```

4. Create a section for each authorization role that you listed. The section name must match the name that you set in the `[AuthorizationRoles]` list. For example:

   ```
   [AdminRole]
   ```

5. In the section for each role, define the operations that you want the role to be able to perform. You can set `StandardRoles` to a list of appropriate values, or specify an explicit list of allowed actions by using `Actions`, and `ServiceActions`. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus

   [UserRole]
   Actions=GetVersion
   ServiceActions=GetStatus
   ```

   > **NOTE:** The standard roles do not overlap. If you want a particular role to be able to perform all actions, you must include all the standard roles, or ensure that the clients, SSL identities, and so on, are assigned to all relevant roles.

6. In the section for each role, define the access permissions for the role, by setting `Clients`, `SSLIdentities`, and `GSSPrincipals`, as appropriate. If an incoming connection matches one of the allowed clients, principals, or SSL identities, the user has permission to perform the operations allowed by the role. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus
   Clients=localhost
   SSLIdentities=admin.example.com
   ```

7. Save and close the configuration file.
8. Restart IDOL for your changes to take effect.

> **IMPORTANT:** If you do not provide any authorization roles for a standard role, IDOL uses the default client authorization for the role (localhost for `Admin` and `ServiceControl`, all clients for `Query` and `ServiceStatus`). If you define authorization only by actions, Micro Focus recommends that you configure an authorization role that disallows all users for all roles by default. For example:
>
> ```
> [ForbidAllRoles]
> StandardRoles=*
> Clients=""
> ```
>
> This configuration ensures that IDOL uses only your action-based authorizations.

Chapter 3: Configure Connectors

This section describes how to configure your connectors for mapped security.

Introduction

In a Mapped Security architecture, connectors perform the following tasks that are related to security:

- Connectors extract the permissions that are applied to files in a repository and add an access control list (ACL) to each document that is indexed into IDOL Server.
- Connectors add a field to each document to identify the security type. IDOL must know which security type applies to a document so that it can use the correct library to read the ACL.
- For some repositories, connectors extract group information and send it to OmniGroupServer. Connectors usually extract group information in cases where the repository uses its own system for storing users and groups. OmniGroupServer sends a `SynchronizeGroups` action to the connector when it needs group information, and the connector returns the information.

Retrieve Access Control Lists and Identify the Security Type

The following configuration settings are sufficient to configure most connectors for mapped security. However, some connectors require additional or different configuration, so Micro Focus recommends that you refer to the documentation for your connectors.

- To retrieve an ACL for each item in the repository and add the ACL to a document field, set the configuration parameter `MappedSecurity` to **`true`**. For example:

  ```
  [FetchTasks]
  MappedSecurity=True
  ```

- To add a field to each document that specifies the security type, create an ingest action or run a Lua script. For example:

  ```
  [Ingestion]
  IngestActions=META:SecurityType=LDAP
  ```

  > **NOTE:** The field name and value that you specify must match the field name (specified by

> `PropertyFieldCSVs`) and field value (specified by `PropertyMatch`) that you used to identify the security type in your IDOL Server configuration file.

Chapter 4: Configure OmniGroupServer

This section describes how to set up and use OmniGroupServer.

Introduction

OmniGroupServer is a generic group server that collects user and group security information from many types of repository.

OmniGroupServer can retrieve information from:

- Active Directory
- Documentum
- eRoom
- LDAP
- IBM/Lotus Notes
- Microsoft NT (though Micro Focus recommends retrieving these groups through LDAP).
- Netware
- OpenText LiveLink
- SharePoint (SharePoint uses a combination of SharePoint and NT security, so in addition to retrieving the SharePoint groups you must also retrieve NT groups through LDAP. For more information about setting up mapped security for documents retrieved from SharePoint, refer to the Administration Guide for your SharePoint Connector).
- Text files

OmniGroupServer stores the collected security information and checks for updates at regular intervals.

When a user starts a session with IDOL server, the client application requests security information from IDOL server. IDOL server retrieves the user's security details from OmniGroupServer and returns the information to the application. The client application then adds the security information to every subsequent query to IDOL. IDOL server can then compare the user's security details to a document's access control list (ACL) to determine whether to grant access to the document.

OmniGroupServer Configuration File

You can configure OmniGroupServer by editing the configuration file. The configuration file is located in the OmniGroupServer installation folder. You can modify the file with a text editor.

The parameters in the configuration file are divided into sections that represent OmniGroupServer functionality.

For information about the configuration parameters that you can set in the OmniGroupServer configuration file, refer to the *OmniGroupServer Reference*.

Server Section

The `[Server]` section specifies the ACI port of the OmniGroupServer. It also contains parameters that determine which clients are permitted to make requests to the server.

Service Section

The `[Service]` section determines which computers are permitted to use and control the OmniGroupServer service.

Logging Section

The `[Logging]` section specifies how messages are logged. You can log separate message types to different log files.

Default Section

The `[Default]` section contains default settings for the parameters that are set in other sections of the configuration file. For example, if you configure several different repository types, you can set the default value for a configuration parameter here.

Repositories Section

The `[Repositories]` section contains a list of *jobs* or *tasks* that you want to run to retrieve security information from repositories. This list also specifies the repositories to query when the `repository` parameter is not specified in an action.

If you want one instance of OmniGroupServer to retrieve information from multiple repositories of the same type, you must configure a separate job for each repository. For example, for one instance of OmniGroupServer to connect to multiple LDAP servers, you must configure a job for each LDAP server.

Start and Stop OmniGroupServer

The following section describes how to start and stop OmniGroupServer.

Start OmniGroupServer

- Double-click `OmniGroupServer.exe` in the OmniGroupServer installation directory (Windows only).
- If OmniGroupServer was installed as a Windows service, you can start the service from the Windows Services dialog box (Windows only).
- Use the start script, located in the installation folder (UNIX only).

Stop OmniGroupServer

- Send the following action to the OmniGroupServer's service port.

  `http://`*`host`*`:`*`ServicePort`*`/action=stop`

  where,

`host`	is the host name or IP address of the machine where OmniGroupServer is installed.
`ServicePort`	is the OmniGroupServer service port (specified in the `[Service]` section of the configuration file).

- If OmniGroupServer is running as a Windows service, stop OmniGroupServer from the Windows Services dialog box. (Windows only)
- Use the stop script (UNIX only).

Display OmniGroupServer Online Help

The OmniGroupServer online help describes the ACI actions and configuration parameters that you can use with OmniGroupServer.

When OmniGroupServer is running, you can display the online help using `action=help`.

To display the online help

- Send `action=help` to OmniGroupServer from your Web browser:

  `http://host:port/action=Help`

  where

`host`	The host name or IP address of the OmniGroupServer.
`port`	The OmniGroupServer ACI port (by default, `3057`).

Retrieve Security Information from Repositories

This section describes how to configure OmniGroupServer to extract group information from various repositories.

Retrieve Groups from Active Directory

If you have installed OmniGroupServer on a Windows machine, you can use the default configuration to retrieve group information from Active Directory using LDAP. In most cases the only configuration parameter that you should need to change is `ActiveDirectoryDomains`. Set this parameter to a list of domains from which you want to retrieve user and group information. For example:

```
ActiveDirectoryDomains=mydomain.com,anotherdomain.com
```

Related Topics

- Retrieve Groups from LDAP, on page 42

Retrieve Groups from Azure Active Directory

This section describes how to retrieve user and group information from an Azure Active Directory. OmniGroupServer retrieves information from Azure Active Directory through the Microsoft Graph API.

Set up an OAuth Service Application

To use the Microsoft Graph API, you must go to the Azure portal and register an application to represent OmniGroupServer. Full instructions about how to create an application are available in the Microsoft documentation.

To set up an OAuth Service Application

1. Go to the Microsoft Azure Portal.
2. Click **New registration**.
   a. Type a name for the new application.
   b. Specify a redirect URL. The "type" of the redirect URL should be "Web". The redirect URL must match the value of the `RedirectUrl` parameter in the OAuth tool configuration file, `oauth_tool.cfg`, that is supplied with OmniGroupServer. The default value is `http://localhost:7878/oauth`.
3. Click **Certificates and Secrets** and upload a certificate to use to authenticate the OmniGroupServer. You can use a self-signed certificate.

   To generate a self-signed certificate using OpenSSL

   The following commands demonstrate how to use OpenSSL to generate a self-signed certificate. The output file `certificate.cer` is the file to upload to Azure. The file `certificate.pfx` is a PKCS#12 file that includes the private key. You will need this file in a later step, to use the OAuth configuration tool.

   ```
   openssl req -x509 -days 365 -newkey rsa:2048 -keyout certificate.pem -out certificate.pem
   ```

   ```
   openssl pkcs12 -export -in certificate.pem -out certificate.pfx
   ```

   ```
   openssl x509 -pubkey -outform der -in certificate.pem -out certificate.cer
   ```
4. Click **API Permissions** > **Add a permission**.

   The Request API permissions dialog box opens.
5. Click **Microsoft Graph**, followed by **Application permissions** and select the following permissions.

Permissions
Directory.Read.All
Group.Read.All
GroupMember.Read.All
OrgContact.Read.All
User.Read
User.Read.All

6. Run the OAuth configuration tool that is supplied with OmniGroupServer.

a. Open the OAuth tool configuration file, `oauth_tool.cfg`, in a text editor.

b. In the `[Default]` section, set any SSL or proxy settings that are required to access the Graph API:

`SSLMethod`	The version of SSL/TLS to use.
`ProxyHost`	The host name or IP address of the proxy server to use.
`ProxyPort`	The port of the proxy server to use.

For example:

```
SSLMethod=NEGOTIATE
ProxyHost=10.0.0.1
ProxyPort=8080
```

c. In the `[Azure]` section, set the following parameters:

`AdminConsentUrl`	Replace the placeholder `{TenantId}` with your Microsoft 365 Tenant, for example `mydomain.onmicrosoft.com`.
`TokenUrl`	Replace the placeholder `{TenantId}` with your Microsoft 365 Tenant, for example `mydomain.onmicrosoft.com`.
`CustomValue0`	The path to the `.pfx` file that contains the certificate and private key to use to authenticate the OmniGroupServer.
`CustomValue1`	The password for the private key.
`AppKey`	The application (client) ID that was provided when you created the OAuth application.

d. Open a command-line window and run the following command:

```
oauth_tool.exe oauth_tool.cfg Azure
```

A web browser opens, asking you to log in and grant admin consent.

e. Log in and grant admin consent.

The web page displays a message stating that the OAuth details have been successfully stored, and the OAuth tool creates the files `oauth.cfg` and `oauth2_sites.bin`. When you configure OmniGroupServer, import the parameters from `oauth.cfg` into your task configuration.

Configure OmniGroupServer

This section describes how to configure an OmniGroupServer repository to store information from Azure Active Directory.

To retrieve security information from Azure Active Directory

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository. For example:

   ```
   [Repositories]
   Number=1
   0=AzureAD

   [AzureAD]
   ```

3. Include the OAuth configuration parameters necessary to authenticate with the Azure Active Directory. For example:

   ```
   [AzureAD] < "oauth.cfg" [OAUTH]
   ```

   For information about how to generate the `oauth.cfg` file, see Set up an OAuth Service Application, on page 31.

4. Set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the library `ogs_azure`.
`UseSystemProxy`	Specifies whether to obtain details about your HTTP proxy from the system. You might need to configure a proxy server to access the Microsoft Graph API.
`SSLMethod`	The SSL/TLS version to use.

   For example:

   ```
   [AzureAD] < "oauth.cfg" [OAUTH]
   GroupServerLibrary=ogs_azure
   UseSystemProxy=true
   SSLMethod=negotiate
   ```

   For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

5. Save and close the OmniGroupServer configuration file.

Retrieve Groups from Alfresco

This section describes how to retrieve group information from Alfresco.

To retrieve groups from Alfresco

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to contain the groups. For example:

```
[Repositories]
Number=1
0=Alfresco
```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The full path (including the file name) to the library that allows the group server to access the repository. Use the `ogs_alfresco` library.
`AlfrescoUrl`	The URL of the Alfresco instance to retrieve users and groups from. OmniGroupServer appends `/service/api/groups` to the value of this parameter, so do not include this part of the URL. For example, if the REST API for retrieving groups is located at `http://10.0.0.2/alfresco/service/api/groups`, set this parameter to `http://10.0.0.2/alfresco`.
`GroupServerAllUserGroups`	(Optional) The `ogs_alfresco` library does not automatically populate the `GROUP_EVERYONE` group. To add every user to the `GROUP_EVERYONE` group, set this parameter to **`GROUP_EVERYONE`**.
`BasicUsername`	The user name to use to access the Alfresco REST API using basic authentication. Micro Focus recommends encrypting the value of this parameter before entering it into the configuration file. For information about how to encrypt parameter values, see Encrypt Passwords, on page 64.
`BasicPassword`	The password to use to access the Alfresco REST API using basic authentication. Micro Focus recommends encrypting the value of this parameter before entering it into the configuration file. For information about how to encrypt parameter values, see Encrypt Passwords, on page 64.

For example:

```
[Alfresco]
GroupServerLibrary=ogs_alfresco
AlfrescoUrl=http://alfresco-host/alfresco
GroupServerAllUserGroups=GROUP_EVERYONE

// For basic authentication...
BasicUsername=admin
BasicPassword=password
```

For a complete list of configuration parameters that you can use to configure this task, refer to the *OmniGroupServer Reference*

4. Save and close the OmniGroupServer configuration file.

Retrieve Groups through a Connector

In some cases, you must use a connector to retrieve group information from a repository. This might be necessary when a repository has its own system for storing users and groups, and for describing which users are members of which groups.

In the OmniGroupServer configuration file, you can set up a repository and use the parameter `GroupServerJobType=Connector` to specify that a connector must retrieve the information. When `GroupServerJobType=Connector`, OmniGroupServer sends the `SynchronizeGroups` action to the connector. The connector then retrieves the group information and returns it to OmniGroupServer.

To retrieve security groups through a connector

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository. For example:

   ```
   [Repositories]
   Number=1
   0=GroupsFromConnector
   ```

3. Create a section to contain the task details and set the following configuration parameters.

`GroupServerJobType`	The type of task that OmniGroupServer must run. To retrieve groups through a connector, set this parameter to **`Connector`**. This instructs OmniGroupServer to send the `SynchronizeGroups` fetch action to the connector.
`ConnectorHost`	The host name or IP address of the machine that hosts the connector.
`ConnectorPort`	The ACI port of the connector.
`ConnectorTask`	The name of a task section in the connector's configuration file that contains the information and credentials required to connect to the repository.

   For example:

   ```
   [GroupsFromConnector]
   GroupServerJobType=Connector
   ConnectorHost=localhost
   ConnectorPort=1234
   ConnectorTask=MyTask
   ```

   For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

4. Save and close the OmniGroupServer configuration file.

Retrieve Groups from Documentum

This section describes how to configure OmniGroupServer to retrieve user and group information from Documentum.

To retrieve groups from Documentum

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, set the parameter `JavaClassPath` to `ogs_java.jar`.

   ```
   [Repositories]
   JavaClassPath0=ogs_java.jar
   ```

3. In the `[Repositories]` section, create a repository to store the groups. For example:

   ```
   [Repositories]
   JavaClassPath0=ogs_java.jar
   Number=1
   0=Documentum
   ```

4. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the `ogs_java` library.
`JavaGroupServerClass`	The class that implements the group server for the Documentum repository. Set this parameter to: `com.autonomy.groupserver.documentum.DocumentumGroupServer`
`JavaClassPath`	Specify resources that are specific to this job.
`Docbase`	A comma-separated list of docbase names for which you want to retrieve user and group information.
`Username`	The user name that OmniGroupServer should use to authenticate with the Documentum API.
`Password`	The password that OmniGroupServer should use to authenticate with the Documentum API.

   For example:

   ```
   [Documentum]
   GroupServerLibrary=ogs_java
   JavaGroupServerClass=com.autonomy.groupserver.documentum.DocumentumGroupServer
   JavaClassPath0=ogs_documentum.jar
   // DFC location
   JavaClassPath1=DFC_INSTALL_PATH/*.jar
   // dfc.properties for this job
   ```

```
JavaClassPath2=./directory/

Docbase=
Username=
Password=
```

> **NOTE:** The DFC (Documentum Foundation Classes) must be installed on the server, and the path added to the `JavaClassPath` (replacing *`DFC_INSTALL_PATH`* in the example above).
>
> You must also have a `dfc.properties` file that specifies the `DocBroker` host and port as follows:
>
> ```
> dfc.docbroker.host[0]=
> dfc.docbroker.port[0]=
> ```

For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

5. Save and close the OmniGroupServer configuration file.

Retrieve Groups from eRoom

To retrieve groups from eRoom

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to contain the groups. For example:

   ```
   [Repositories]
   Number=1
   0=eRoom
   ```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The full path (including the file name) to the library that allows the group server to access the repository. Use the `ogs_eroom` library.
`eRoomBuiltInGroups`	A Boolean that specifies whether to include built-in groups.
`eRoomCommunityNames`	A comma-separated list of eRoom communities from which users and groups must be retrieved.
`eRoomServer`	The IP address or name of the machine that hosts the eRoom server.
`eRoomXmlUrl`	The URL used to send eRoom requests using the eRoom XML Query Language, eXQL.

   For example:

   ```
   [eRoom]
   GroupServerLibrary=ogs_eroom
   ```

```
eRoomBuiltInGroups=false
eRoomCommunityNames=eRoom Members,ACommunity
eRoomServer=12.34.56.78
eRoomXmlUrl=http://12.34.56.78/eRoomXML/
```

For a complete list of configuration parameters that you can use to configure this task, refer to the *OmniGroupServer Reference*

4. Save and close the OmniGroupServer configuration file.

Retrieve Groups from Google Directory

OmniGroupServer can retrieve group information from a Google Directory. You might want to do this if you are using the IDOL Google Drive Connector to retrieve information from the users in a G Suite team, and want to enable mapped security.

To retrieve group information from Google Directory complete the following steps:

- Configure Authentication (with a service account)
- Configure the OmniGroupServer Repository

Configure Authentication (with a service account)

This section describes how to create a new service account and configure OAuth authentication with the Google Directory.

> **NOTE:** This procedure is subject to change. Micro Focus recommends that you refer to the documentation for Google Directory.

To configure authentication (with a service account)

1. Go to https://console.developers.google.com/ and login.
2. Create a new project, and make sure the project is selected.
3. If not already selected, go to **APIs & Services** > **Library**.
4. Select **Admin SDK** and click **Enable**.
5. Select **APIs & Services** > **Credentials**.
6. Select **Create Credentials** > **Service Account Key**.
7. Select **New Service Account**.
8. Give the new account a name, and select **Role: Project** > **Viewer**.
9. Select **Key type JSON**, and then **Create**.

   A JSON file is saved to your machine. Store the JSON file to be used later.
10. Find the service account you created: **Menu button** > **IAM & Admin** > **Service accounts**.
11. In the table, select **Edit** for the new account.

12. Select the **Enable G Suite Domain-wide Delegation** check box, and click **Save**. (This requires the Product name to be set in the OAuth consent screen, if you haven't set it previously).
13. Click **View Client ID**, and copy the Client ID to be used later (this is also shown in **APIs & Services** > **Credentials**).
14. Go to https://admin.google.com/ and login using an administrator account.
15. On the **Admin Console** menu, select **Security**.
16. Select **Show more** > **Advanced settings** > **Manage API Client Access**.
17. Ensure that an entry is present for the Client ID (from step 13, above), and that the following scopes are included:
    - https://www.googleapis.com/auth/admin.directory.domain.readonly
    - https://www.googleapis.com/auth/admin.directory.user.readonly
    - https://www.googleapis.com/auth/admin.directory.group.readonly.

    When adding scopes, copy any existing scopes, then add the new scopes (comma separated, unquoted), and click **Authorize**.

To obtain OAuth tokens

1. In the OmniGroupServer installation directory, open `oauth_tool.cfg`.
2. Configure the following section, by replacing `<client_email>` with the e-mail address of the service account and `<private_key>` with the private key. The private key is available from the JSON file that was saved to your machine in the previous procedure.

   ```
   [GoogleDirectory]
   OAuthVersion=2.0
   SiteName=GoogleDirectory
   AuthorizationType=GoogleServiceAccountAuthorization
   CustomJson={"GoogleServiceAccount":"<client_
   email>","GoogleServiceAccountPrivateKey":"<private_key>"}
   ```

3. From the command line, run the following command:

   ```
   oauth_tool oauth_tool.cfg GoogleDirectory
   ```

   The OAuth configuration tool generates the files `oauth2_sites.bin` and `oauth.cfg`. The file `oauth.cfg` contains the configuration parameters that are required by OmniGroupServer to authenticate with the directory. For information about how to configure OmniGroupServer, see Configure the OmniGroupServer Repository, below.

Configure the OmniGroupServer Repository

This section describes how to create a new repository for retrieving group information from a Google Directory.

To retrieve security information from Google Directory

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository. For example:

   ```
   [Repositories]
   Number=1
   0=GoogleDirectory

   [GoogleDirectory]
   ```

3. Include the OAuth configuration parameters necessary to authenticate with the Google Directory. For example:

   ```
   [GoogleDirectory] < "oauth.cfg" [OAUTH]
   ```

   For information about how to generate this file, see Configure Authentication (with a service account), on page 39.

4. Set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the library `ogs_google`.
`SSLMethod`	The SSL method to use for communicating with the Google Directory. To use the latest method that is supported by both Google Directory and OGS, set this parameter to `Negotiate`.
`ProxyHost`	The host name or IP address of the proxy server to use to access the Google Directory.
`ProxyPort`	The port of the proxy server to use to access the Google Directory.
`UserAgent`	The value to use for the `User-Agent` header in requests sent to the repository. Micro Focus recommends that you include the string "gzip" in the user agent name.
`AdminEmail`	An administrator account email address to use for authorizing the API calls.
`CustomerId` or `DomainName`	To synchronize groups from only one domain, set `DomainName` to the name of your Google Apps domain. To synchronize groups from multi-domain accounts, set `CustomerId` to the customer ID for the Google Apps account. If you set both parameters, `CustomerId` is ignored.

   For example:

   ```
   [GoogleDirectory] < "oauth.cfg" [OAUTH]
   GroupServerLibrary=ogs_google
   SSLMethod=Negotiate
   ProxyHost=10.0.0.1
   ```

```
ProxyPort=8080
UserAgent=autn:netlib/2.0 gzip
AdminEmail=admin@example.com
// CustomerId=x01y34z45
DomainName=example.com
```

For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

5. Save and close the OmniGroupServer configuration file.

Retrieve Groups from LDAP

This section describes how to configure OmniGroupServer to retrieve user and group information from an LDAP directory.

To retrieve security information from an LDAP directory

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to store the LDAP groups. For example:

```
[Repositories]
Number=1
0=LDAP
```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the LDAP group server library, `ogs_ldap`.
`LDAPServer`	The host name or IP address of the machine that hosts the LDAP directory.
`LDAPPort`	The port to use to access the LDAP directory.
`LDAPBase`	The distinguished name of the search base.
`LDAPType`	The type of LDAP server (for example, **`MAD`** for Microsoft Active Directory).
`LDAPSecurityType`	The type of security to use when communicating with the LDAP server (for example, `SSL` or `TLS`).
`LDAPBindMethod`	The type of authentication to use to access the LDAP directory. To log on as the same user that is running OmniGroupServer, set this parameter to `NEGOTIATE`.

For example:

```
[LDAP]
GroupServerLibrary=ogs_ldap
```

```
LDAPServer=myLDAPserver
LDAPPort=636
LDAPBase=DC=DOMAIN,DC=COM
LDAPType=MAD
LDAPSecurityType=SSL
LDAPBindMethod=NEGOTIATE
```

For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

4. Save and close the OmniGroupServer configuration file.

Retrieve Groups from LDAP using Kerberos on Linux

This section describes how to retrieve user and group information from an LDAP directory (Microsoft Active Directory in this example), using Kerberos, when OmniGroupServer is installed on Linux.

To retrieve security information from an LDAP directory

1. Log on to the Windows Domain Controller and complete the following steps:
   a. Add a new user to the Active Directory (for example `Domain\OGSUser`).
   b. Generate a keytab file:

```
ktpass -out ogsuser_domain.keytab
       -princ ogsuser/linux_ogs_host.domain@kerberos_realm
       -mapUser DOMAIN\ogsuser
       -mapOp set
       -pass P4ssw0rd!
       -crypto all
       -ptype KRB5_NT_PRINCIPAL
```

   > **NOTE:** This command must be entered on one line.

   c. Move the `.keytab` file that is generated to the machine that hosts OmniGroupServer.
2. On the machine that hosts OmniGroupServer, run `kinit` using the keytab file that you generated.

```
kinit -k -t /path/to/ogsuser_domain.keytab -c /path/to/ogsuser_domain.krbcache
ogsuser/linux_ogs_host.domain@kerberos_realm
```

3. Open the OmniGroupServer configuration file.
4. In the `[Repositories]` section, create a repository to store the LDAP groups. For example:

```
[Repositories]
Number=1
0=LDAP
```

5. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the LDAP group server library, `ogs_ldap`.
`LDAPServer`	The host name or IP address of the machine that hosts the LDAP directory.
`LDAPPort`	The port to use to access the LDAP directory.
`LDAPBase`	The distinguished name of the search base.
`LDAPType`	The type of LDAP server (for example, **`MAD`** for Microsoft Active Directory).
`LDAPSecurityType`	The type of security to use when communicating with the LDAP server (for example, `SSL` or `TLS`).
`LDAPBindMethod`	The type of authentication to use to access the LDAP directory. Set this parameter to **`KERBEROS`**.

For example:

```
[Default]
GroupServerStartTime=now
GroupServerCycles=-1
GroupServerRepeatSecs=86400
GroupServerCaseInsensitive=TRUE
GroupServerShowAlternativeNames=TRUE
GroupServerMaxDatastoreQueue=100000

[Repositories]
...
GroupServerDefaultRepositories=LDAP
0=LDAP

[LDAP]
GroupServerLibrary=ogs_ldap
LDAPServer=ldap.mydomain.com
LDAPPort=389
LDAPBase=DC=mydomain,DC=com
LDAPType=MAD
LDAPBindMethod=KERBEROS
```

For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

6. Save and close the OmniGroupServer configuration file.
7. Run OmniGroupServer using the following command, replacing the paths with the correct paths for your system:

```
KRB5CCNAME=FILE:/path/to/ogsuser_domain.krbcache KRB5_
KTNAME=FILE:/path/to/ogsuser_domain.keytab nohup ./omnigroupserver.exe &
```

Retrieve Groups from Netware

This section describes how to configure OmniGroupServer to retrieve user and group information for Netware, through LDAP.

To retrieve security information from Netware using LDAP

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to store the Netware groups. For example:

   ```
   [Repositories]
   Number=1
   0=NetwareLDAP
   ```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the LDAP group server library, `ogs_ldap`.
`LDAPServer`	The host name or IP address of the machine that hosts the LDAP directory.
`LDAPType`	The type of LDAP server (for Netware/Novell eDirectory, set this parameter to **`NED`**).
`LDAPPort`	The port to use to access the LDAP directory.
`LDAPBase`	The distinguished name of the search base.
`LDAPUsername`	The user to use to access the directory.
`LDAPPassword`	The password to use to access the directory. Micro Focus recommends that you encrypt the password before adding it to the configuration file. For information about how to encrypt passwords, see Encrypt Passwords, on page 64.

   For example:

   ```
   [NetwareLDAP]
   GroupServerLibrary=ogs_ldap
   LDAPServer=myLDAPserver
   LDAPType=NED
   LDAPPort=389
   LDAPBase=O=org
   LDAPUsername=cn=admin,o=org
   LDAPPassword=passw0rd
   ```

   For a complete list of configuration parameters that you can use to configure the task, refer to the *OmniGroupServer Reference*.

4. Save and close the OmniGroupServer configuration file.

Retrieve Groups from Notes

This section describes how to configure OmniGroupServer to retrieve user and group information from IBM Notes.

To retrieve groups from Notes

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to store the groups. For example:

   ```
   [Repositories]
   Number=1
   0=Notes
   ```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the `ogs_notes` library.
`NotesServer`	The IP address or name of the Notes server in which the names database is stored.
`NotesUserIDFile`	The full path to the location where the user ID file is stored.
`NotesPassword`	The password for the user ID file specified by `NotesUserIDFile`. You can encrypt the password using the Autonomy password encryption utility.
`Database`	The name of the Notes database in which the security information is stored. This is usually `names.nsf`.

   For example:

   ```
   [Notes]
   GroupServerLibrary=ogs_notes
   NotesServer=MyServer
   NotesUserIDFile=C:\Autonomy\admin.id
   NotesPassword=password
   Database=names.nsf
   ```

   For a complete list of configuration parameters that you can use to retrieve information from Notes, refer to the *OmniGroupServer Reference.*

4. Save and close the OmniGroupServer configuration file.

Retrieve NT Groups

This section describes how to configure OmniGroupServer to retrieve NT user and group information.

NOTE: This section describes how to retrieve NT user and group information from an NT server. Micro Focus recommends that you retrieve NT user and group information through LDAP instead. For more information, see Retrieve Groups from LDAP, on page 42.

To retrieve NT groups

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a repository to store the groups. For example:

   ```
   [Repositories]
   Number=1
   0=NT
   ```

3. Create a section to contain the task details and set the following configuration parameters:

`GroupServerLibrary`	The path (including the file name) to the library file that allows the group server to access the repository. Use the `ogs_nt` library.
`NtDomainName`	The name of one or more domains from which you want to obtain group information. Separate multiple domain names with commas (there must be no space before or after a comma).
`NtServerName`	The IP address or name of the NT server in which the names database is stored.

   For example:

   ```
   [NT]
   GroupServerLibrary=ogs_nt
   NtDomainName=DOMAIN
   NtServerName=localhost
   ```

   For a complete list of configuration parameters that you can use to configure this task, refer to the *OmniGroupServer Reference*.

4. Save and close the OmniGroupServer configuration file.

Retrieve Security Information from a Text File

To import user and group information from a text file, you can:

- use the `Import` action:

  ```
  /action=import&FileName=textfile.txt
  ```

- configure a task using the `ogs_text` library:
  - to replace all the information in the OmniGroupServer repository with the information in the text file, set `GroupServerIncremental=False`. In this case, you should also set the

`TextFile` parameter, to specify the path of the file that contains the information:

```
[Text]
GroupServerLibrary=ogs_text
GroupServerIncremental=false
TextFile=C:\Autonomy\OmniGroupServer\TextDir\mydata.txt
```

- to configure incremental updates from text files, set `GroupServerIncremental=True`. In this case, do not set the `TextFile` parameter. Instead, set the `TextDirectory` parameter to the path of the directory that contains the text files:

  ```
  [Text]
  GroupServerLibrary=ogs_text
  GroupServerIncremental=TRUE
  TextDirectory=C:\Autonomy\OmniGroupServer\TextDir
  ```

  Each text file written to the directory must have a file name in the format `<RepostoryName>_<Index>`, where

`<RepositoryName>`	is the name of the repository.
`<Index>`	is a number that counts up from 0, and specifies the order of the updates. For example, the changes listed in `Text_0` are processed first, and then the changes in `Text_1`, and so on. OmniGroupServer stores the index of the last processed file in a file named `<RepositoryName>_LastProcessed`.

Whether you use the `import` action or set up a task in the OmniGroupServer configuration file, the text files must be constructed as described in Construct the Text File, below

Construct the Text File

This section describes how to construct a text file containing user and group information in a format that can be imported into OmniGroupServer.

The structure of the file should follow these rules:

- Each line that is processed should have the following syntax:

  ```
  #<COMMAND> <CSV>
  ```

  where,

  `<COMMAND>` is one of the commands listed in Text Commands, on the next page.

  `<CSV>` is a comma-separated list of users or groups to which the command is applied.

  > **NOTE:** The commands `INITIALIZE`, `NESTEDGROUPS`, and `FINALIZE` do not require the `<CSV>`.

  For example:

  ```
  #USER User1,User2,User3
  ```

- Any line preceded by two slashes "`//`" is ignored.

- Blank lines are ignored.
- <CSV> entries must be either:
  - enclosed in "double quotes" with all "double quotes" in the string escaped (\)
  - listed without quotation marks, in which case commas must be escaped (\).

  For example:

<CSV> entry	Specifies the user/group/name
"ABCDE"	ABCDE
"AB\"CDE"	AB"CDE
"AB,CDE"	AB,CDE
\"AB"CDE	"AB"CDE
ABC"DE	ABC"DE
ABCDE	ABCDE
AB\,CDE	AB,CDE
AB\CDE	AB\CDE
AB\\\,CDE	AB\,CDE
AB\\\CDE	AB\\CDE
AB\\\\CDE	AB\\CDE

- When you are using the text file with `action=import`, the presence or absence of the INITIALIZE and FINALIZE commands determines whether the group server uses the information to perform an incremental update:
  - to replace all the information in the OmniGroupServer repository with the information in the text file, use the INITIALIZE and FINALIZE commands at the start and end of the text file.
  - to do an incremental update using the information in the text file, do not use the INITIALIZE and FINALIZE commands at the start and end of the text file.

Text Commands

Command	Description	Usage	Example
`INITIALIZE`	Initializes flags in the datastore so that deletions can be performed using the `#FINALIZE`	Used only with `action=import`. Must be the first command, if	`#INITIALIZE`

	command.	used.	
USER	Define a new user or select an existing user for the following commands.	Can appear anywhere. Ends any previous selection.	#USER NewUser
GROUP	Define a new group or select an existing group for the following commands.	Can appear anywhere. Ends any previous selection.	#GROUP NewGroup
ALTUSERNAME	Define an alternative name for a new user.	Must follow USER.	#USER NewUser #ALTUSERNAME AltName
MEMBERGROUP	Add a group as a member of the selected group.	Must follow GROUP.	#GROUP Group3 #MEMBERGROUP Group4 //Group 4 is added as a member of Group 3
MEMBEROF	Specifies that the selected user or group is a member of a group.	Must follow USER or GROUP.	#USER NewUser #MEMBEROF Group5 //NewUser becomes a member of Group5
MEMBER	Adds a user or group to the selected group.	Must follow GROUP.	#GROUP Group5,Group6 #MEMBER User5,User6,User7 //User5, User6, and User7 become members of Group5 and Group6
DELETEUSER	Deletes a user.	Can appear anywhere. Ends any previous selection.	#DELETEUSER User4
DELETEALTUSERNAME	Removes an alternative name from a user.	Must follow USER.	#USER User2 #DELETEALTUSERNAME AltName
DELETEGROUP	Delete a group.	Can appear anywhere. Ends any previous selection.	#DELETEGROUP Group5
DELETEMEMBERGRO	Remove a group	Must follow	#GROUP Group3

UP	from being a member of the selected group.	GROUP.	#DELETEMEMBERGROUP Group4 //Group4 is no longer a member of Group3
DELETEMEMBEROF	Remove the selected user from being a member of a group.	Must follow USER.	#USER User1 #DELETEMEMBEROF Group2 //User1 is no longer a member of Group2
DELETEMEMBER	Remove a user or group from being a member of the selected.	Must follow GROUP.	#GROUP Group3 #DELETEMEMBER User3 //User3 is no longer a member of Group3

FIELD	Adds a field to the selected user. You can add multiple values to a single field (see the example).	Must follow USER.	#USER User8
			#FIELD
			MyField=Value1,MyField=Value2

DELETEFIELD	Delete a field from the selected user.	Must follow USER.	#USER User8 #DELETEFIELD MyField
NESTEDGROUPS	Processes nested groups. For example, if Group1 is a member of Group2, and Group2 is a member of Group3, OmniGroupServer can consider Group1 and a member of Group3.	Used only with action=import. Typically the last command (except for FINALIZE, if used).	#NESTEDGROUPS
FINALIZE	Deletes any existing memberships that have not be updated since the last #INITIALIZE.	Used only with action=import. Must be the last command, if used.	#FINALIZE

Schedule Tasks

To schedule when OmniGroupServer checks repositories for updates, use the following configuration parameters:

`GroupServerStartTime`	The date and/or time (24 hour clock) that you want the task to start. You can specify **`now`** if you want the task to start immediately after OmniGroupServer starts. You can specify a comma-separated list of dates, with the formats `YYYY/MM/DD HH:NN:SS`, `HH:NN:SS`, or `HH:NN`.
`GroupServerRepeatSecs`	The number of seconds that should elapse between the start of each cycle. If a previous job has not finished when a job with the same name is scheduled, the scheduled job does not start until the unfinished job completes.
`GroupServerCycles`	The number of times you want the task to run. To run the task indefinitely, do not set this parameter, or set it to the default value of **`-1`**.

To configure a default schedule for all tasks, set these parameters in the `[Default]` section of the configuration file.

```
[Default]
GroupServerStartTime=01:00
GroupServerRepeatSecs=3600
GroupServerCycles=-1
```

Alternatively, to schedule a specific task, set these parameters in the task section:

```
[LDAP]
GroupServerLibrary=ogs_ldap
LDAPServer=myLDAPserver
LDAPPort=636
LDAPBase=DC=DOMAIN,DC=COM
LDAPType=MAD
LDAPSecurityType=SSL
LDAPBindMethod=NEGOTIATE
GroupServerStartTime=now
GroupServerRepeatSecs=3600
GroupServerCycles=-1
```

Combine Repositories

This section describes how to combine the user and group information stored in two different repositories. You might need to combine repositories when a document repository uses more than

one security type. For example, Microsoft SharePoint uses a combination of NT and SharePoint security.

To combine the information from two different repositories

1. Open the OmniGroupServer configuration file.
2. In the `[Repositories]` section, create a new repository to contain the combined information. For example:

   ```
   [Repositories]
   Number=3
   0=SharePoint
   1=LDAP
   2=Combine

   ...

   [Combine]
   ```

3. In the section that you created for combining the security information, use the following parameters to configure the combine task:

`GroupServerJobType`	The type of task that OmniGroupServer must run. Set this parameter to **`Combine`**.
`GroupServerSections`	The names of the repositories in the configuration file that you want to merge.
`GroupServerStartDelaySecs`	The number of seconds to wait before starting the task. It is important to set this parameter so that the combine operation does not start until the security groups have been retrieved by the other tasks. This ensures that the combine operation uses the latest information. The delay that you specify only has to ensure that the other jobs start first.

   For example, this configuration would combine the security information from the `SharePoint` and `LDAP` repositories into the `Combine` repository:

   ```
   [Combine]
   GroupServerJobType=Combine
   GroupServerSections=Sharepoint,LDAP
   GroupServerStartDelaySecs=10
   ```

   For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference.*
4. Save and close the configuration file.

Configure Redirection

It is possible for an OmniGroupServer repository to be updated and queried at the same time. However, in theory this could cause the results of queries to be incorrect. This is because OmniGroupServer could be part way through processing a new user or group as the query is processed.

OmniGroupServer can redirect queries from one repository to another. You can use this feature to prevent OmniGroupServer simultaneously updating and querying a repository. This section explains how to configure redirection.

TIP: An easier approach than setting up redirection is to configure one task to crawl a repository for updates, and then copy the information into another OmniGroupServer repository that can receive queries from IDOL. You can do this using `GroupServerJobType=combine`, for example:

```
[Repositories]
0=RepositoryThatGetsUpdates
1=RepositoryThatIsQueried

[RepositoryThatGetsUpdates]
// Task settings

[RepositoryThatIsQueried]
GroupServerJobType=combine
GroupServerSections=RepositoryThatGetsUpdates
GroupServerStartDelaySecs=10
```

To set up redirection, you must set up multiple OmniGroupServer repositories to retrieve user and group information from the same source. One of these repositories can be queried while the others are updated.

You can use redirection with any type of repository, however it is most useful in cases where OmniGroupServer takes a significant amount of time to update a repository.

The following example configuration shows how to configure redirection. Queries are directed to `MainRepository` and `SecondaryRepository`.

```
[Default]
GroupServerLibrary=ogs_example.dll
GroupServerCycles=-1
GroupServerRepeatSecs=86400
GroupServerSkipIfCanQuery=TRUE
GroupServerClearDatastoreOnStart=TRUE
GroupServerRedirectOnCompletion=MainRepository

[Repositories]
GroupServerDefaultRepositories=MainRepository
Number=2
0=MainRepository
```

```
1=SecondaryRepository

[MainRepository]
GroupServerStartTime=06:00
GroupServerRedirectedQueriesOnly=FALSE

[SecondaryRepository]
GroupServerStartTime=18:00
GroupServerRedirectedQueriesOnly=TRUE
```

The following steps explain how this configuration works:

1. When OmniGroupServer is started, both `MainRepository` and `SecondaryRepository` are empty. Queries are directed to `MainRepository`.
2. At 06:00 OmniGroupServer attempts to run the `MainRepository` job, but the job is skipped. This is because `GroupServerSkipIfCanQuery=true`, and the repository can be queried. Queries continue to be directed to `MainRepository`.
3. At 18:00 OmniGroupServer attempts to run the `SecondaryRepository` job. Although `GroupServerSkipIfCanQuery=true`, the repository cannot currently be queried (due to `GroupServerRedirectedQueriesOnly=TRUE`), so the job is allowed to run.
4. Some time later the `SecondaryRepository` job completes. Queries to `MainRepository` are now redirected to the datastore for `SecondaryRepository` (due to `GroupServerRedirectOnCompletion=MainRepository`).
5. The next day at 06:00, OmniGroupServer attempts to run the `MainRepository` job. The job runs this time because queries to it are currently redirected to `SecondaryRepository`.
6. Some time later, the `MainRepository` job completes. Queries to `MainRepository` are now redirected back to the datastore for `MainRepository` (due to `GroupServerRedirectOnCompletion=MainRepository`).
7. At 18:00, the `SecondaryRepository` job runs. Before the job starts the datastore is cleared (due to `GroupServerClearDatastoreOnStart=TRUE`). On completion queries to `MainRepository` are redirected to the `SecondaryRepository` datastore.
8. The next day at 06:00, the `MainRepository` job runs. Before starting the datastore is cleared (due to `GroupServerClearDatastoreOnStart=TRUE`). On completion, queries to `MainRepository` are redirected to the `MainRepository` datastore.
9. Steps 7 and 8 repeat until the service is stopped or an error occurs.

   If an error occurs which means that the update was not successful (for example the job could not connect to the data source), redirection does not occur. The next job does not run because its datastore is being used for queries. This means that there is no break in service, however until a job is successful OmniGroupServer will not have the latest user and group data.

   If OmniGroupServer is stopped, when it restarts it begins again from step 1. However, the `MainRepository` datastore is already populated so can immediately return results (though the information does not include the latest changes).

The example configuration above could be extended to more than two repositories by creating copies of the `SecondaryRepository` job and adjusting the start times accordingly.

Chapter 5: Configure a Front End

This section describes how to configure your front end application to enable security.

Introduction

You can configure Micro Focus and third-party front-end applications to enable authentication and add user security details to queries sent to IDOL Server, ensuring users access only those documents for which they have permission.

For information about how to configure an Micro Focus front-end application, refer to the documentation for that application.

Security for Third-Party Interfaces

You can set up security for a third-party interface, and use IDOL Server to ensure that result documents are displayed only to people who have the appropriate privileges.

To set up security for a third-party interface

1. In the IDOL Server configuration file, create the user security types for the repositories from which data is indexed (see Configure User Security, on page 18). If you want IDOL Server to perform authentication, you must include a security type that specifies the security library that IDOL Server uses for authentication.
2. In the front-end application, define the user's security details for the user security types you have set up in IDOL server. You can do this by creating the user in IDOL Server using the `UserAdd` action, and specifying the user's security details for the repositories. For example:

   ```
   http://localhost:9000/action=UserAdd
                          &UserName=JSmith
                          &Password=secret123
                          &SecurityNTUserusername=JohnS
                          &SecurityNTUserDomain=MyCompany
   ```

   This defines a user whose `autonomy` user name and password are `JSmith` and `secret123`, and whose user name and domain in the repository for which the `NTUser` section sets up security are `JohnS` and `MyCompany`.

Refer to the *IDOL Server Reference* for full details of how to use actions to define and edit users in IDOL Server.

3. When a user logs on to the system, your front-end application must communicate with IDOL Server to retrieve an encrypted string that contains the user's security details for your repositories.

   Send a `UserRead` action to IDOL Server, with the `SecurityInfo` action parameter set to **`true`**. You must include the user's user name and password for the repository that IDOL Server authenticates against. You must also include the domain if you are authenticating against an NT repository. For example:

   ```
   http://localhost:9000/action=UserRead
                                    &UserName=JSmith
                                    &Password=secret123
                                    &SecurityInfo=true
   ```

   If the `CheckEntitlement` configuration parameter is set to **`true`**, the user is also implicitly authenticated before the `securityinfo` string is returned.

   IDOL Server returns XML details of the user's settings, including an encrypted security string that includes the details for all the repositories for which you have set up IDOL Server user security types.

4. Configure the front-end application to specify the encrypted security string returned in Step 3 as the value of the `SecurityInfo` parameter when the front-end application sends queries to IDOL Server (for example, using the `Agent`, `Profile`, `Suggest` and `Query` actions).

   For example:

   ```
   http://localhost:9000/action=Query
                           &Text=accounts
                           &SecurityInfo=encrypted_string
   ```

   Query IDOL Content with Security Information, on page 68 includes an example of how to use the `SecurityInfo` parameter.

   Refer to the *IDOL Server Reference* for full details of the actions that you can send to IDOL Server.

Instead of sending actions to generate the security string in the steps outlined above, you can use the ACI API to create the encrypted strings. For more information, refer to the *ACI API Programming Guide*.

> **NOTE:** Actions issued through a browser must be percent encoded to allow unreserved alphanumeric characters. For example, the user name `us\jsmith` is a valid format for IAS, but an action issued through a browser to IDOL server must percent-encode the unreserved URL character: `us%5cjsmith`.

Security through the Java ACI API NG

This section contains example code that demonstrates how to use the Java ACI API NG to obtain a `securityinfo` string for a user and perform queries against IDOL.

The sample code makes the following assumptions:

- The IDOL Server has been deployed in a Kerberos environment.
- You have configured your JVM to work in a Kerberos environment. For information about how to do this, refer to the Java documentation, for example: http://docs.oracle.com/javase/7/docs/technotes/guides/security/.
- The sample code requests the details required, for example a user name and the query text, on the command line. You can modify this so that the information is obtained in a suitable way.

The sample code first creates an `AciService` to use for communicating with IDOL Server. For more information about creating an ACI service, refer to the *ACI API Programming Guide*.

```
// Create the AciService to use when communicating with IDOL Server...
final AciService aciService = new AciServiceImpl(
        new GssAciHttpClientImpl(new HttpClientFactory().createInstance()),
        new GssAciServerDetails(serviceName, host, port)
);
```

The sample code then obtains a `securityinfo` string:

```
final Document document = aciService.executeAction(
        new AciParameters(
              new AciParameter(AciConstants.PARAM_ACTION, "UserRead"),
              new AciParameter("UserName", userName),
              new AciParameter("SecurityInfo", true)
        ),
        new DocumentProcessor()
);
```

After extracting the `securityinfo` string from the XML response, the sample code then sends a query to IDOL:

```
final Document results = aciService.executeAction(
       new AciParameters(
             new AciParameter(AciConstants.PARAM_ACTION, "Query"),
             new AciParameter("Text", queryText),
             new AciParameter("combine", "simple"),
             new AciParameter("maxResults", 10),
             new AciParameter("totalResults", true),
             new AciParameter("print", "none"),
             new AciParameter("summary", "ParagraphConcept"),
             new AciParameter("characters", 400),
             new AciParameter("anyLanguage", true),
```

```
                new AciParameter("outputEncoding", "utf8"),
                new AciParameter("SecurityInfo", securityInfo)
        ),
        new DocumentProcessor()
    );
```

The sample code prints information about the result documents to the command line. You can modify this to suit your requirements.

Example

```
package com.autonomy.examples.aci.ng;

import com.autonomy.aci.client.services.AciConstants;
import com.autonomy.aci.client.services.AciService;
import com.autonomy.aci.client.services.AciServiceException;
import com.autonomy.aci.client.services.ProcessorException;
import com.autonomy.aci.client.services.impl.AciServiceImpl;
import com.autonomy.aci.client.services.impl.DocumentProcessor;
import com.autonomy.aci.client.transport.AciParameter;
import com.autonomy.aci.client.transport.gss.GssAciHttpClientImpl;
import com.autonomy.aci.client.transport.gss.GssAciServerDetails;
import com.autonomy.aci.client.transport.impl.HttpClientFactory;
import com.autonomy.aci.client.util.AciParameters;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.math.NumberUtils;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * This example uses the GSS-API extensions to communicate with ACI Servers
 * secured with Kerberos. It will first retrieve the users Securityinfo
 * string from IDOL Server and then send this as part of a standard query.
 */

public class GssApiExample {

 // We'll use XPAth in this example, although you could easily use the
 // processor and classes from the QueryExample...
 private XPath xpath = XPathFactory.newInstance().newXPath();
```

```
 private void query(final String serviceName, final String host, final int port,
final String userName, final String queryText) {
    // Create the AciService to use when communicating with IDOL Server...
    final AciService aciService = new AciServiceImpl(
            new GssAciHttpClientImpl(new HttpClientFactory().createInstance()),
            new GssAciServerDetails(serviceName, host, port)
    );

    // Get the security info string...
    final String securityInfo = getSecurityInfoString(aciService, userName);

    // Send a query to IDOL Server the contains the security info string...
    final Document results = aciService.executeAction(
          new AciParameters(
                 new AciParameter(AciConstants.PARAM_ACTION, "Query"),
                 new AciParameter("Text", queryText),
                 new AciParameter("combine", "simple"),
                 new AciParameter("maxResults", 10),
                 new AciParameter("totalResults", true),
                 new AciParameter("print", "none"),
                 new AciParameter("summary", "ParagraphConcept"),
                 new AciParameter("characters", 400),
                 new AciParameter("anyLanguage", true),
                 new AciParameter("outputEncoding", "utf8"),
                 new AciParameter("SecurityInfo", securityInfo)
          ),
          new DocumentProcessor()
    );

    try {
       // Get the individual documents in the response...
       final NodeList nodeList = (NodeList) xpath.evaluate
("/autnresponse/responsedata/hit", results, XPathConstants.NODESET);
       final int documents = nodeList.getLength();

       System.out.println("Displaying " + documents + " results from a total of " +
xpath.evaluate("/autnresponse/responsedata/totalhits", results) + '\n');

       for(int ii = 0; ii < documents; ii++) {
          // Get an individual document in the response...
          final Node node = nodeList.item(ii);

          // Output it's details to the console...
          System.out.println(xpath.evaluate("reference", node));

          System.out.println("\nTitle     : " + xpath.evaluate("title", node));
          System.out.println("Relevance : " + xpath.evaluate("weight", node) + '%');
          System.out.println("Database  : " + xpath.evaluate("database", node));
```

```
            final String links = xpath.evaluate("links", node);
            if(StringUtils.isNotBlank(links)) {
                System.out.println("Matched on: " + links);
            }
            System.out.println("Summary   : " + xpath.evaluate("summary",
node).replaceAll("\n", " ")); // Remove line breaks for clarity...
        }
    }
    catch(XPathExpressionException e) {
       throw new ProcessorException("Unable to parse the query response.", e);
    }
 }

 private String getSecurityInfoString(final AciService aciService, final String
userName) {
    // Get the users Security Info string...
    final Document document = aciService.executeAction(
            new AciParameters(
                  new AciParameter(AciConstants.PARAM_ACTION, "UserRead"),
                  new AciParameter("UserName", userName),
                  new AciParameter("SecurityInfo", true)
            ),
            new DocumentProcessor()
    );

    try {
       // Use XPath to pull out the users SecurityInfo string, you could
       // alternatively do this with a simple processor...
       return (String) xpath.evaluate("/autnresponse/responsedata/securityinfo",
document, XPathConstants.STRING);
    }
    catch(XPathExpressionException e) {
       throw new ProcessorException("Unable to obtain the SecurityInfo string from
the response.", e);
    }
 }

 public static void main(final String[] args) throws IOException {
    // Read three things from stdin, the host, port and the query text...
    final BufferedReader reader = new BufferedReader(new InputStreamReader
(System.in));

    System.out.print("Please enter the Kr5b service name: ");
    final String serviceName = reader.readLine();

    System.out.print("Please enter the host [localhost]: ");
    final String host = StringUtils.defaultIfBlank(reader.readLine(), "localhost");

    System.out.print("Please enter the port [9000]: ");
```

```
    final int port = NumberUtils.toInt(reader.readLine(), 9000);

    System.out.print("Please enter your username: ");
    final String userName = reader.readLine();

    System.out.print("Please enter the query text [*]: ");
    final String queryText = StringUtils.defaultIfBlank(reader.readLine(), "*");

    try {
      // Display the query response...
      new GssApiExample().query(serviceName, host, port, userName, queryText);
    }
    catch(AciServiceException ase) {
      System.err.println("An error occurred while trying to output the IDOL query
response.");
      ase.printStackTrace();
    }
 }

}
```

Chapter 6: Secure Communications

You can configure IDOL Server and other ACI servers to communicate using Secure Socket Layer (SSL) communications.

SSL Communications

You can configure Secure Socket Layer (SSL) communications for IDOL Server, as well as for other ACI servers, front-end applications, and connectors.

The exact configuration you use depends on the component you are configuring. For more information, refer to the topics about Secure Communications in *IDOL Expert*, and the relevant component Administration Guides.

Chapter 7: Encrypt Passwords

Micro Focus recommends encrypting all passwords before you enter them into a configuration file.

Encrypt Passwords

Micro Focus recommends that you encrypt all passwords that you enter into a configuration file.

> **NOTE:** The AES encryption method has been hardened in version 12.9.0 and later. Micro Focus strongly recommends that you reencrypt all passwords in configuration files by using the updated tool.
>
> The older AES encryption format and basic encryption methods are now deprecated. Passwords that you have encrypted with older versions continue to work, but IDOL logs a warning. Support for these older encryption methods will be removed in future.

Create a Key File

A key file is required to use AES encryption.

To create a new key file

1. Open a command-line window and change directory to the IDOL installation folder.
2. At the command line, type:

   `autpassword -x -tAES -oKeyFile=./MyKeyFile.ky`

   A new key file is created with the name `MyKeyFile.ky`

> **CAUTION:** To keep your passwords secure, you must protect the key file. Set the permissions on the key file so that only authorized users and processes can read it. IDOL must be able to read the key file to decrypt passwords, so do not move or rename it.

Encrypt a Password

The following procedure describes how to encrypt a password.

To encrypt a password

1. Open a command-line window and change directory to the IDOL installation folder.
2. At the command line, type:

```
autpassword -e -tEncryptionType [-oKeyFile] [-cFILE -sSECTION -pPARAMETER]
PasswordString
```

where:

Option	Description
`-tEncryptionType`	The type of encryption to use: • **`AES`** - AES256 • **`Basic`** **DEPRECATED:** The basic encryption type is deprecated in version 12.9.0 and later. Use the more secure AES encryption instead. Passwords that you have encrypted with older versions continue to work, but IDOL logs a warning. Support for this older encryption method will be removed in future. For example: **`-tAES`**
`-oKeyFile`	AES encryption requires a key file. This option specifies the path and file name of a key file. The key file must contain 64 hexadecimal characters. For example: **`-oKeyFile=./key.ky`** **NOTE:** The full (absolute) path of the key file is included in the encrypted value, because IDOL requires the key to decrypt the password. If you move or rename the key file, this path becomes invalid and you must update the encrypted value.
`-cFILE -sSECTION -pPARAMETER`	(Optional) You can use these options to write the password directly into a configuration file. You must specify all three options. • **`-c`**. The configuration file in which to write the encrypted password. • **`-s`**. The name of the section in the configuration file in which to write the password. • **`-p`**. The name of the parameter in which to write the encrypted password. For example: **`-c./Config.cfg -sMyTask -pPassword`**
`PasswordString`	The password to encrypt.

For example:

```
autpassword -e -tBASIC MyPassword
```

```
autpassword -e -tAES -oKeyFile=./key.ky MyPassword
```

```
autpassword -e -tAES -oKeyFile=./key.ky -c./Config.cfg -sDefault -pPassword
MyPassword
```

The password is returned, or written to the configuration file.

Decrypt a Password

The following procedure describes how to decrypt a password.

To decrypt a password

1. Open a command-line window and change directory to the IDOL installation folder.
2. At the command line, type:

   ```
   autpassword -d -tEncryptionType PasswordString
   ```

   where:

Option	Description
`-tEncryptionType`	The type of encryption: • **`Basic`** • **`AES`** - AES256 For example: **`-tAES`**
`PasswordString`	The password to decrypt.

   For example:

   ```
   autpassword -d -tBASIC 9t3M3t7awt/J8A
   ```

   ```
   autpassword -d -tAES PasswordString
   ```

   The password is returned in plain text.

Chapter 8: Query Servers

This section describes how to include security information in queries sent to IDOL server. It also describes how to log query security information, and how to view security details from a group server.

View Security Details on a Group Server

You can send the following HTTP commands from your Web browser to a group server to view a user's security details:

`http://`*`host`*`:`*`port`*`/action=GetGroups&username=`*`user`*

where,

`host`	The IP address or name of the machine on which the group server is installed.
`port`	The ACI port of the group server.
`user`	The user name of the user whose security details you want to view. The name is case-sensitive.

For example:

```
http://12.3.4.56:4000/action=GetGroups&username=John Smith
```

This command uses port `4000` to request security details for the user `John Smith` from the group server located on a machine with the IP address `12.3.4.56`.

In response to this command, the group server returns a list of groups to which this user belongs. For example:

```
<?xml version='1.0' encoding='UTF-8' ?>
<autnresponse>
<action>GETGROUPS</action>
<response>SUCCESS</response>
   <responsedata>
      <Groups>admingroup</Groups>
      <Groups>testgrp2</Groups>
      <Groups>testgrp16</Groups>
   </responsedata>
</autnresponse>
```

Query IDOL Content with Security Information

When you use document security, your front-end application must use the `SecurityInfo` parameter to include the encrypted user security details in queries. For example:

```
/action=Query&Text=accounts&SecurityInfo=[encrypted string]
```

The `SecurityInfo` parameter specifies security details for the user who sends the query. When IDOL processes the query, it compares the SecurityInfo details with the ACL field in result documents to determine which documents the user has permission to view.

In the previous example, a user sends a query to find documents containing the text *accounts*. The query returns only those documents that the user is permitted to view.

> **TIP:** For information about which actions accept the `SecurityInfo` parameter, refer to the *IDOL Server Reference*.
>
> You cannot use `SecurityInfo` for administrative actions, which have authorization restrictions by IP addresses, SSL identity, or GSS principal in the `[AuthorizationRoles]` section of your IDOL component configuration files.

You can obtain the encrypted `SecurityInfo` string in the following ways:

- your front-end application can send a `UserRead` action to the IDOL Community component.
- You can use the Autonomy ACI API to encrypt the user details.

Obtain the SecurityInfo String

You can obtain a `SecurityInfo` string by sending the `UserRead` action to the IDOL Community component, for example:

```
action=UserRead&UserName=DOMAIN\USER&SecurityInfo=true
```

You might need to specify information for authentication, for example a password:

```
action=UserRead&UserName=DOMAIN\USER&SecurityInfo=True&Password=mypassword
```

SecurityInfo Parameters

The following table lists the parameters that must be included in the `SecurityInfo` string when you send a query to an IDOL Server that has document security enabled.

> **TIP:** To troubleshoot issues with the security information, you can use the `UserDecryptSecurityInfo` action in your IDOL Community component to decrypt a security string, for example to check that it contains the right permissions and restrictions.

Security Type	Mapped Security	Unmapped Security
Documentum	• user: Documentum user name • group: comma-separated list of groups to which the user belongs	• user: Documentum user name • pass: Documentum password
eRoom	• user: eRoom user name • group: comma-separated list of groups to which the user belongs	N/A
Exchange	• user: Exchange user name • domain: Exchange domain name	• user: Exchange user profile name • domain: Exchange server domain name
FileNet	• user: FileNet user name • group: comma-separated list of groups to which the user belongs	N/A
iManage	• user: iManage user name • group: comma-separated list of groups to which the user belongs	N/A
Lotus Notes	• user: Lotus Notes user name • group: comma-separated list of groups that the user belongs to	• user: Lotus Notes user name • group: comma-separated list of groups that the user belongs to • domain: Lotus Notes domain name
Microsoft NT	If your security type is `AUTONOMY_SECURITY_V4_NT_MAPPED`: • user: `NT_domain_name\user_name` • group: `NT_domain_name\group` • Note: the security string must be percent encoded. If your security type is `AUTONOMY_SECURITY_NT_MAPPED`: • user: NT user name • domain: NT domain name • group: comma-separated list of groups to which the user belongs	• user: NT user name • pass: NT password • domain: NT domain name
NetWare	• user: NetWare user name	• user: NetWare user name

	• domain: NetWare domain name • group: comma-separated list of groups to which the user belongs	• pass: NetWare password • domain: NetWare domain name
ODBC	N/A	Dependent on the stored function/procedure
Open Text	• user: Open Text user name • group: comma-separated list of groups to which the user belongs	• user: Open Text user name • pass: Open Text password
Oracle	N/A	Dependent on the stored function/procedure
PCDocs	• user: PCDocs user name • group: comma-separated list of groups to which the user belongs	N/A
SharePoint	• user: SharePoint user name • group: comma-separated list of groups to which the user belongs	N/A
UNIX	• user: UNIX user name • group: comma-separated list of groups to which the user belongs.	N/A

Log Query Security Information

If you want to log query security information, set the `SecurityDebugLogging` parameter in the `[Server]` section of the IDOL Server configuration file to **`true`**:

```
[Server]
SecurityDebugLogging=true
```

On this setting, if the query log level is set to `FULL`, then decrypted details of the security information supplied with the `SecurityInfo` parameter are logged into the query log.

> **NOTE:** Although this setting is useful for troubleshooting purposes when you set up security, you should disable it afterward.

Part II: Appendixes

This section contains the following appendixes.

- Available Security Libraries
- Custom Mapped Security
- Additional Restrictions on Document Access

Appendix A: Available Security Libraries

Mapped Security

Use the Generic Mapped Security plug-in library that is supplied with IDOL.

There are two exceptions: ODBC and Oracle. For mapped security in these two cases, create a custom mapped security type.

Unmapped Security

Micro Focus supplies unmapped security libraries for specific repositories. Micro Focus currently supplies unmapped security libraries for:

- Documentum
- Exchange
- Lotus Notes
- NetWare
- NT
- ODBC
- Open Text
- Oracle

Appendix B: Custom Mapped Security

This section describes how to set up mapped security for a custom security type.

Introduction

There might be cases when you retrieve information from a repository that does not use one of the supported security models.

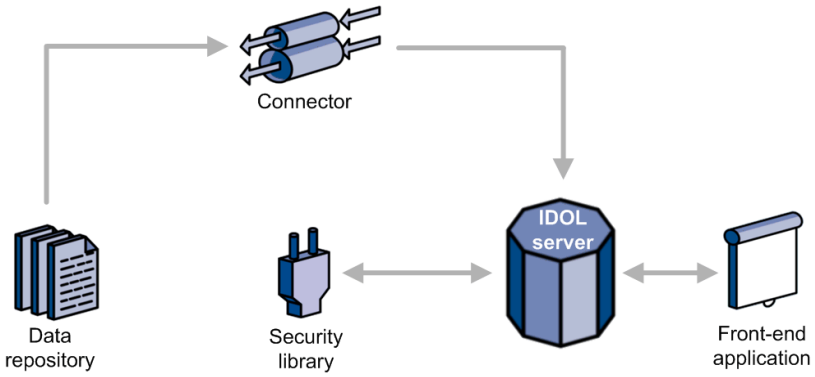
So that IDOL Server can process custom security types, it includes a *Generic Security Module*. The Generic Security Module is part of the Mapped Security plug-in that performs security checks.

The Generic Security Module uses the security type `AUTONOMY_SECURITY_V4_GENERIC_MAPPED`. The difference between this and any other security type is that it provides a method of configuring the format of the ACL and the sequence of security checks that are performed.

System Architecture

To configure custom mapped security, you must set up the following components:

- A connector to extract information from your repository. The connector must add an ACL to each document, and add a field that identifies the custom security type.
- Micro Focus IDOL server. You must specify the format of the ACL and the sequence of security checks to perform.



The connector retrieves information from your data repository and sends it to CFS so that it can be indexed it into IDOL server. The connector adds an encrypted ACL to each document. The ACL is in a custom format.

When IDOL receives a query, it sends the user's `securityinfo` token and the result documents to the Generic Security Module, part of the Mapped Security Plug-in. The Generic Security Module determines whether a user is allowed to see documents retrieved as query results. The structure of the ACL and the sequence of security checks that the Generic Security Module must perform are specified by configuration parameters in the IDOL Server configuration file.

Set Up IDOL Server

The following sections describe how to configure IDOL Server to process documents that use a custom security type.

Set Up a Custom Mapped Security Type

To set up a custom mapped security type

1. Open the IDOL Server configuration file.
2. Create a new section to configure the custom security type. For example:

```
[Security]
0=NT_V4
1=CustomSecurity

[CustomSecurity]
```

3. In the new section, define the security type. You will need to set at least the following parameters:

`SecurityCode`	A unique number to use as an identifier for the security type.
`Library`	The path to the security library that IDOL must use to check the ACL of documents that use this security type. Set this parameter to the path of the mapped security library that is included by default with IDOL Server.
`Type`	The security type. Set this parameter to **`AUTONOMY_SECURITY_V4_GENERIC_MAPPED`**.
`SecurityACLFormat`	The format of the custom ACL. For information about how to set this parameter, see Specify the ACL Format and Security Checks, on the next page.
`SecurityACLCheck`	A comma-separated list of security checks that the Generic Security Library must perform, using the information in the ACL and the

information provided in the user's `securityinfo` token. For information about how to set this parameter, see Specify the ACL Format and Security Checks, below. For examples that show how to set this parameter, see Example - NT Security, on page 78, and Example - Custom Security, on page 80.

4. For example:

```
[CustomSecurity]
SecurityCode=2
Library=./modules/mapped_security.dll
Type=AUTONOMY_SECURITY_V4_GENERIC_MAPPED
SecurityACLFormat=<E=B!>:U:<U=SLE+>:G:<G=SLE+>:OG:<OG=SLE+>:NU:<NU=SLE-
>:NG:<NG=SLE->
SecurityACLCheck=OG=[DG]?P:-,NU=[DU]?F:-,NG=[DG]?F:-,E=1?P:-,U=[DU]?P:-,G=
[DG]?P:F
EscapedEntries=True
...
```

5. Save and close the configuration file.

Related Topics

- Configure IDOL Server, on page 14
- Configure a Front End, on page 56

Specify the ACL Format and Security Checks

This section describes how to construct the values for the configuration parameters `SecurityACLFormat` and `SecurityACLCheck`.

Specify the ACL Format

`SecurityACLFormat=`*`ACLFormatString`*

where,

Variable	Format
ACLFormatString	*String ACLFormatFields String*
ACLFormatFields	*ACLField* | *ACLField NonEmptyString ACLFormatFields*
ACLField	"<" *ACLFieldName* "=" *Properties* ">"
ACLFieldName	*NonEmptyString*
Properties	*Property* | *Property Properties*
Property	"B" | "D" | "S" | "L" | "E" | "X" | "C" | "+" | "-" | "!"
String	"" | *NonEmptyString*
NonEmptyString	Character *String*

Specify the Security Checks

`SecurityACLCheck=`*`ACLCheckString`*

where,

Variable	Format
ACLCheckString	*CheckString* | *CheckString* "," *ACLCheckString*
CheckString	*ACLValue Operator UserValue* "?" *MatchAction* ":" *NoMatchAction*
ACLValue	"'" *String* "'" | *ACLFieldName*
Operator	"=" | "~=" | "&=" | "~&=" | "=~" | "=&" | "=&~"
UserValue	*String* | "[" *ValueType* "]"
ValueType	"U" | "USER" | "G" | "GROUP" | "D" | "DOMAIN" | "DU" | "DOMAINUSER" | "DG" | "DOMAINGROUP" | "P" | "PASSWORD"
MatchAction	*Action*
NoMatchAction	*Action*
Action	"P" | "PASS" | "F" | "FAIL" | "C" | "-" | "CONTINUE" | PositiveInteger

Syntax

The following table defines the property types used in the `SecurityACLFormat` configuration parameter. Acceptable types appear in parentheses.

Property	Definition
B	Boolean type (equivalent to Digit type)
D	Digit type
S	String type
L	Comma-separated list (S)
E	Encrypted (S)
X	Escaped (S)
C	Case insensitive (S)
+	Positive terms (S)
-	Negative terms (S)
!	Everyone flag (B | D)

The following table describes the operators that you can use between the *ACLValue* and *UserValue* in the SecurityACLCheck configuration parameter:

Operator	Definition	Usage
=	Returns true if there is at least one match.	
~=	Returns true if there is at least one match, or if there is nothing in the *ACLValue* to check.	Valid only when the *ACLValue* field has the L (list) property.
&=	Returns true only if every value in *ACLValue* matches a value in *UserValue*. There must be at least one match.	Valid only when the *UserValue* is [G], [GROUP], [DG], or [DOMAINGROUP].
~&=	Returns true if every value in *ACLValue* matches a value in *UserValue*, or if there is nothing in the *ACLValue* to check.	Valid only when the *UserValue* is [G], [GROUP], [DG], or [DOMAINGROUP], and the *ACLValue* field has the L (list) property.
=~	Returns true if there is at least one match, or if there is nothing in the *UserValue* to check.	Valid only when the *UserValue* is [G], [GROUP], [DG], or [DOMAINGROUP].
=&	Returns true only if every value in *UserValue* matches a value in *ACLValue*. There must be at least one match.	Valid only when the *UserValue* is [G], [GROUP], [DG], or [DOMAINGROUP].
=&~	Returns true if every value in *UserValue* matches a value in *ACLValue*, or if there is nothing in the *UserValue* to check.	Valid only when the *UserValue* is [G], [GROUP], [DG], or [DOMAINGROUP].

The following table describes the possible values of *ValueType* in the SecurityACLCheck configuration parameter:

Value Type	Definition
[U], [USER]	User name only
[DU], [DOMAINUSER]	Domain\User name or User name if \ exists
[G], [GROUP]	Group only
[DG], [DOMAINGROUP]	Domain\Group or Group if \ exists
[D], [DOMAIN]	Domain only
[P], [PASSWORD]	Password only

The following table describes the possible actions that can be used in the `SecurityACLCheck` configuration parameter:

Action	Definition
`F, FAIL`	Fail
`P, PASS`	Pass
`C, -, CONTINUE`	Continue
`Number N`	Skip the next *N* checks

Example - NT Security

This example explains the format of an NT-style ACL, describes the security checks performed against the ACL, and then shows how NT security would be configured using the Generic security type.

The ACL is a string of text that specifies who is allowed to view a document. An NT ACL looks like this:

`0:U:<users>:G:<groups>:NU:<nousers>:NG:<nogroups>`

The ACL begins with a 0 or a 1 (the Everyone flag) and is followed by four sections:

U Allowed users

G Allowed groups

NU Disallowed users

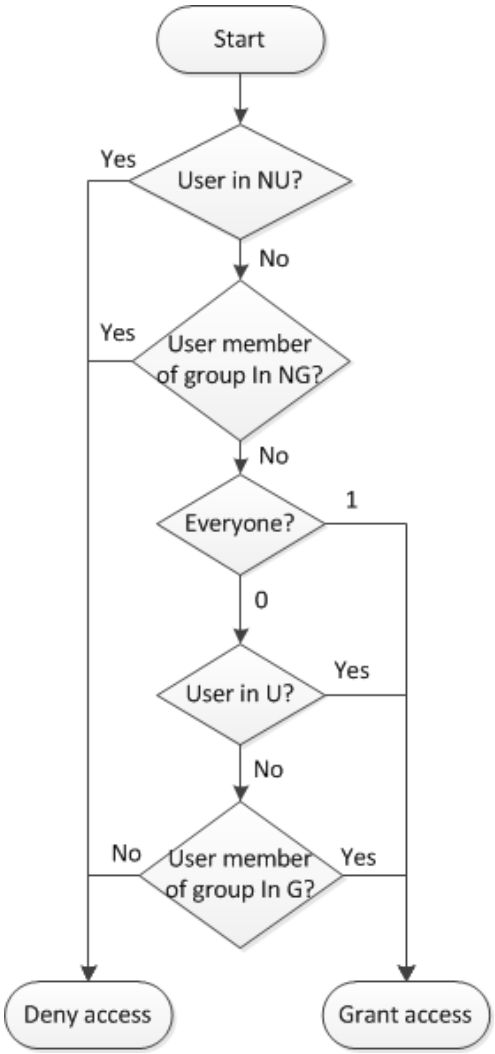
NG Disallowed groups

The `<users>`, `<groups>`, `<nousers>`, and `<nogroups>` sections each hold comma separated lists of encrypted strings. Each encrypted string holds a username or the name of a group.

When the ACL is processed by IDOL, the disallowed users and groups are always given priority. A user is not allowed to view a document if they are in the list of disallowed users (NU) or if they belong to a group in the list of disallowed groups (NG).

If the user has not been explicitly denied access, the rest of the ACL determines whether they are granted access to a document. If the Everyone flag is 0 they are granted access only if their user name appears in the list of allowed users (U), or if they are in a group that appears in the list of allowed groups (G). If the Everyone flag is 1 the user is granted access, regardless of whether they appear in those lists (U or G).

This process is represented by the following diagram:



In the following example ACL, `user1` and `user2` are permitted to view the document; `user3` is not permitted to view the document. No access rights for any groups are specified:

```
0:U:user1,user2:G::NU:user3:NG:
```

The ACL string must be complete and well-formed. For example, even when no allowed groups are specified the string element `G::` must appear in the correct place.

In the IDOL Server configuration file, to configure NT security through the Generic Security Module, the content security section would contain:

```
Type=AUTONOMY_SECURITY_V4_GENERIC_MAPPED

SecurityACLFormat=<E=B!>:U:<U=SLE+>:G:<G=SLE+>:NU:<NU=SLE->:NG:<NG=SLE->

SecurityACLCheck=NU=[DU]?F:-,NG=[DG]?F:-,E=1?P:-,U=[DU]?P:-,G=[DG]?P:F
```

NOTE: `SLE+` in the example indicates that ACL field `U` is an Encrypted String List of Positive terms.

The comma-separated values in the SecurityACLCheck parameter are explained in the following table:

NU=[DU]?F:-	Compare ACL field disallowed users (NU) to the user (DU). Deny access (F) if there is a match, otherwise continue (-).
NG=[DG]?F:-	Compare ACL field disallowed groups (NG) to each of the users group memberships (DG). Deny access (F) if there is a match, otherwise continue (-).
E=1?P:-	Compare ACL field everyone (E) to 1. Allow access (P) if matched, otherwise continue (-).
U=[DU]?P:-	Compare ACL field allowed users (U) to the user (DU). Allow access (P) if matched, otherwise continue (-).
G=[DG]?P:F	Compare ACL field allowed groups (G) to each of the users group memberships (DG). Allow access (P) if matched, otherwise deny access (F).

Example - Custom Security

A repository might use a security model that resembles NT security. For example, the ACL might specify a list of groups that automatically grant access to the document if the user is a member. This would require a different ACL, and an additional check to be performed.

The following example shows how to modify the value of the SecurityACLFormat parameter. When compared to the standard NT ACL described in Example - NT Security, on page 78, you can see that the underlined section has been added:

```
SecurityACLFormat=<E=B!>:U:<U=SLE+=>:G:<G=SLE+>:OG:<OG=SLE+>:NU:<NU=SLE->:NG:<NG=SLE->
```

If the ACL is modified for custom security, and has an additional ACL field with positive groups that have priority over the negative terms, the SecurityACLCheck could be modified as follows:

```
SecurityACLCheck=OG=[DG]?P:-,NU=[DU]?F:-,NG=[DG]?F:-,E=1?P:-,U=[DU]?P:-,G=[DG]?P:F
```

The new value in the SecurityACLCheck parameter is explained in the following table:

OG=[DG]?P:-	Compare ACL field OG to each of the users group memberships. If the user is a member of one of the groups, allow access, otherwise continue. The remaining checks are the same as for the NT security type described in Example - NT Security, on page 78.

Set Up the Connector

The connector you use to retrieve information from your repository must be configured to add an ACL to each document. For example, if you are implementing mapped security for data extracted from an ODBC data source, you must ensure that the template that ODBC Connector uses to index data includes a field for the ACL.

ACLs are typically written to a document field named `AUTONOMYMETADATA`. The format of the ACL is usually controlled by the connector, so that it never needs configuring manually. In the case of custom mapped security, the format must match the format you specified using the `SecurityACLFormat` parameter in the IDOL Server configuration file.

To add an ACL to each document, the connector must first construct the ACL. You can encrypt the ACL using the Lua function `encrypt_security_field`.

The connector must also add a field to each document that identifies the security type. For example, using Lua:

```
document:addField("SecurityType","CustomMapped")
```

The value that you specify for this field must match the value that you specified in the IDOL Server configuration file to identify the security type (see Identify the Security Type, on page 17).

Appendix C: Additional Restrictions on Document Access

This appendix describes additional ways to restrict access to data in IDOL server.

Restrict IDOL Server Database Access

You can restrict access to individual IDOL server databases by creating privileges and associating them with IDOL server roles. Only users who belong to a role with the necessary privileges can query an IDOL server database.

To restrict a role's access to specific databases

1. Add a privilege to IDOL server by sending a `RoleAddPrivilege` action from a browser.

   For example:

   ```
   http://localhost:9000/action=RoleAddPrivilege
                                    &Name=Databases
                                    &SingleValue=false
   ```

   This example creates a multivalued `Databases` privilege in the IDOL server.

2. Add the privilege to a role, and specify a value for it.

   For example:

   ```
   http://localhost:9000/action=RoleSetPrivilegeForRole
                                    &RoleName=Marketing
                                    &Privilege=Databases
                                    &Value=News,Markets,Sales
   ```

   This example adds the `Databases` privilege to the `Marketing` role. The `Databases` privilege has the value `News,Markets,Sales`.

3. Use a SecurityInfo string in your queries as usual. See Query IDOL Content with Security Information, on page 68.

   > **TIP:** If you do not have connector-generated ACLs for your documents, you can use an ACL of the form `1:U::G::NU::NG:` in all documents to ensure that Content requires a valid security string.

> **NOTE:** You can use the `DatabasePrivilege` parameter in the `[Roles]` section of the Community configuration file to specify a privilege that defines which databases all roles can access. The role to which all users belong by default is specified by the `DefaultRolename` parameter in the

`[Roles]` section of the Community configuration file. This parameter is set to the `everyone` role by default.

NOTE: Every time you restart Community, it resets the databases that the default role can query (specified by the `DefaultRolename` configuration parameter) to include all databases. You can override this behavior and persist the databases that can be queried by the default role by setting `AutoSetDatabases=False`.

The SecurityInfo string approach is the most secure way to ensure that only authorized users can view the documents in a database. However, you can also identify the privileges directly in the front end if you do not want to use SecurityInfo strings.

To restrict a role's access to specific databases without using SecurityInfo

1. Set up the privilege and role as described in steps 1 and 2 of the previous procedure.
2. Set up the front-end application to identify a user's privileges when the user logs on to the system.

   For example:

   ```
   http://localhost:9000/action=RoleGetUserPrivilegeValueList
                                    &UserName=JSmith
                                    &Privilege=Databases
   ```

   If the user `JSmith` has been added only to the `Marketing` role, the result this action returns specifies that the user's `Databases` privilege has the value `News,Markets,Sales`.
3. The front-end application can now include the `Databases` privilege's values in queries this user sends to IDOL server. To specify databases to which a query is restricted, for example, the front end should add the `DatabaseMatch` parameter to the query it sends, and set it with the value that the `RoleGetUserPrivilegeValueList` action returned.

   For example:

   ```
   http://localhost:9000/action=Query
                                   &Text=2003 marketing campaigns in Europe
                                   &DatabaseMatch=News,Markets,Sales
   ```

   In this example, the query specified is applied to the News, Markets and Sales databases.

Glossary

A

ACI (Autonomy Content Infrastructure)
A technology layer that automates operations on unstructured information for cross-enterprise applications. ACI enables an automated and compatible business-to-business, peer-to-peer infrastructure. The ACI allows enterprise applications to understand and process content that exists in unstructured formats, such as email, Web pages, Microsoft Office documents, and IBM Notes.

ACI Server
A server component that runs on the Autonomy Content Infrastructure (ACI).

ACL (access control list)
An ACL is metadata associated with a document that defines which users and groups are permitted to access the document.

action
A request sent to an ACI server.

active directory
A domain controller for the Microsoft Windows operating system, which uses LDAP to authenticate users and computers on a network.

C

Category component
The IDOL Server component that manages categorization and clustering.

Community component
The IDOL Server component that manages users and communities.

connector
An IDOL component (for example File System Connector) that retrieves information from a local or remote repository (for example, a file system, database, or Web site).

Connector Framework Server (CFS)
Connector Framework Server processes the information that is retrieved by connectors. Connector Framework Server uses KeyView to extract document content and metadata from over 1,000 different file types. When the information has been processed, it is sent to an IDOL Server or Distributed Index Handler (DIH).

Content component
The IDOL Server component that manages the data index and performs most of the search and retrieval operations from the index.

D

DAH (Distributed Action Handler)
DAH distributes actions to multiple copies of IDOL Server or a component. It allows you to use failover, load balancing, or distributed content.

DIH (Distributed Index Handler)
DIH allows you to efficiently split and index extremely large quantities of data into multiple copies of IDOL Server or the Content component. DIH allows you to create a scalable solution that delivers high performance and high availability. It provides a flexible way to batch, route, and categorize the indexing of internal and external content into IDOL Server.

I

IDOL
The Intelligent Data Operating Layer (IDOL) Server, which integrates unstructured, semi-structured and structured information from multiple repositories through an understanding of the content. It delivers a real-time environment in which operations across applications and content are automated.

IDOL Proxy component
An IDOL Server component that accepts incoming actions and distributes them to the appropriate subcomponent. IDOL Proxy also performs some maintenance operations to make sure that the subcomponents are running, and to start and stop them when necessary.

Intellectual Asset Protection System (IAS)
An integrated security solution to protect your data. At the front end, authentication checks that users are allowed to access the system that contains the result data. At the back end, entitlement checking and authentication combine to ensure that query results contain only documents that the user is allowed to see, from repositories that the user has permission to access. For more information, refer to the IDOL Document Security Administration Guide.

K

KeyView
The IDOL component that extracts data, including text, metadata, and subfiles from over 1,000 different file types. KeyView can also convert documents to HTML format for viewing in a Web browser.

L

LDAP
Lightweight Directory Access Protocol. Applications can use LDAP to retrieve information from a server. LDAP is used for directory services (such as corporate email and telephone directories) and user authentication. See also: active directory, primary domain controller.

License Server
License Server enables you to license and run multiple IDOL solutions. You must have a License Server on a machine with a known, static IP address.

O

OmniGroupServer (OGS)
A server that manages access permissions for your users. It communicates with your repositories and IDOL Server to apply access permissions to documents.

P

primary domain controller
A server computer in a Microsoft Windows domain that controls various computer resources. See also: active directory, LDAP.

V

View
An IDOL component that converts files in a repository to HTML formats for viewing in a Web browser.

W

Wildcard
A character that stands in for any character or group of characters in a query.

X

XML
Extensible Markup Language. XML is a language that defines the different attributes of document content in a format that can be read by humans and machines. In IDOL Server, you can index documents in XML format. IDOL Server also returns action responses in XML format.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Micro Focus IDOL 12.11 Document Security Administration Guide

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to swpdl.idoldocsfeedback@microfocus.com.

We appreciate your feedback!