ECMA2

What is ECMA2?

One of the most important aspects of the sync engine is its ability to communicate with directories. There is some built-in connectivity to common directories, such as Active Directory, but there are more systems on the planet than Microsoft could possibly build connectors for.

In the sync engine, there is a public API that allows you to build your own custom connectors that work with the sync engine. Creating a connector is clearly a developer experience, but even as an IT-Pro you would need to understand these APIs when working with your developers to create custom connectors.

This section describes ECMA2 in detail from a theoretical detailed view and how certain scenarios should be solved with the framework. It does not contain sample code for connectors. There are several sites, for example on GitHub, where you can find sample code.

ECMA2 is available in FIM2010, MIM2016, and Azure AD Connect sync. The same connector can be used on all three systems.

The history and background

ECMA1/XMA

In MIIS2003 the ECMA1 (also known as XMA) framework was introduced. It was essentially a file-based connector that allowed adding code around the file processing. There was a call-based export feature as well, but since there was no corresponding call-based import feature, it was less useful. Many connectors were created on ECMA1, but it had obvious disadvantages. Many of the built-in connectors had access to data not available to an external connector developer. This made it hard or impossible to write connectors for some scenarios. For example, the initial Office 365 connector for DirSync (this was before the Azure AD rebranding) was re-written three times to work around limitations in ECMA1.

ECMA2

To address these issues, ECMA2 was developed. It was created with the goal:

> A team at Microsoft, a partner or a customer can create a connector that can have the same functionality, stability, and performance as a connector shipped out-of-box.

History

Almost as soon as FIM2010 had shipped, this was the main project the sync team worked on. ECMA2 was enhanced numerous times (2.1, 2.2, and so on) to complete the vision that all connectors should be possible to build on it. All the new connectors from Microsoft were built on ECMA2, verifying the design and add enhancements as needed.

During development, the project was called "EZMA" (pronounced "Easy-MA") and this acronym can still be seen in the sample code generated by the sync engine and on MSDN.

Principles

Some principles were defined when ECMA2 was first conceived. The vision and ultimate goal were that every connector shipped out-of-box should be possible to create with ECMA2. That would allow partners to create connectors to systems Microsoft would never go after. But it was also time to revamp all of the connectors that Microsoft shipped in FIM2010. There were numerous complaints about the feature-lacking out-of-box Lotus Notes connector. All of the database connectors were built on OLEDB, a technology slowly being deprecated by all database vendors.

Another principle was that an ECMA2 connector should be possible to run without the sync engine proper. So why would this be needed? As an example, there is a SharePoint UserProfile connector shipped from Microsoft. When you want to test a new release of SharePoint, you want to make sure it can be tested by the Office team without them knowing too much of FIM2010. By allowing a connector to be invoked outside the sync engine, you can test another product with a test harness, which is mimicking the sync engine. A common thought experiment that was used when designing ECMA2 was: "If Microsoft decided to build a virtual directory, could an ECMA2 connector be used to connect to the remote system". (Microsoft didn't plan to build a virtual directory. This was only used to try the concepts theoretically.)

Yet another reason to build ECMA2 and build connectors from Microsoft on it was so the connectors could be shipped independently from the sync engine. With MIIS2003 and ILM2007, when a bug fix for one of the built-in connectors had to be shipped, the entire sync engine was released. When FIM2010 was released, the FIM Service also had to be released for every connector bug fix. For many customers, it was a daunting task to upgrade both the sync engine and the FIM Service for a simple fix. By separating the connectors, these could be shipped independently of the sync engine.

As of today, the ECMA2 framework is complete and has delivered on its initial goal. It is a good platform for building your own custom connector.

Limitations

There are a few things an ECMA2 connector cannot do that still requires built-in connectors.

It cannot interact directly with PCNS; only the built-in ADMA can do that.

It does not have access to internal asynchronous interfaces. Only the FIM Service MA has access to these. The asynchronous interfaces allow a connector during export to receive data from the sync engine but not return the export result immediately. Instead, the connector comes back later with the result on another interface. The reason the FIM Service connector is using this is to allow it to be multi-threaded. One thread receives data and dispatches it, but multiple threads process the data and return the result to the sync engine.

Connectors and management agents

The "thing" that is connecting to remote systems used to be called management agents, but these are now called connectors. In the UI and in interfaces the old term MA can frequently be seen.

There were several reasons why the term was changed. The primary reason was that nobody actually called them MAs and referred to them as connectors. The industry standard was "connectors" and in all analyst reports, these were always referred to as connectors.

The Wikipedia definition also didn't help:

> A Management agent is a software agent that runs on a managed node (example: a router) and provides an interface to manage it.

All connectors have always tried to avoid installing anything on the remote system. As much as possible, the connectors use public remote APIs and try to not have a dependency on software created by Microsoft installed on a remote system.

To align with the industry, the term "connector" was introduced instead.

The Architecture

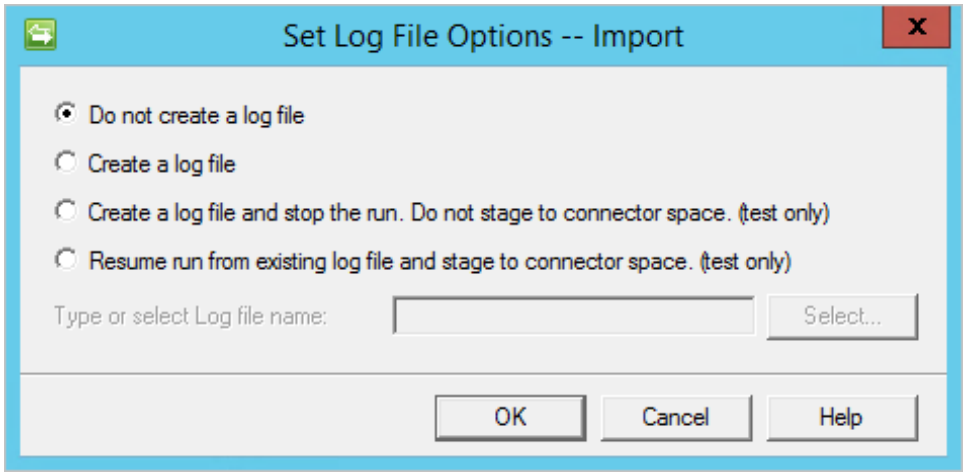
The controller

The component in the sync engine responsible for talking with the connectors, both the built-in connectors and ECMA1/ECMA2, is called the **controller**. It is responsible for reading and writing the connector space and present the data in the format the connector wants. For example, one connector might want all attributes for an object when only one has changed and another connector only wants the changed attribute. The controller is responsible for delivering the data in the format needed by each connector. The format is defined by capabilities in the connector code.

The data stored in the connector space is different depending on the defined capabilities. That is the reason why many capabilities can only be set during the creating of a connector. As soon as the first full import has completed, the connector space layout would be different if a capability has changed.

Drop files

You can get data directly from the controller by using drop files. A drop file is defined on the run profile and has these options:



Options for creating a drop file

A drop file gets its data directly from the controller so it is an excellent way to look at how the data would be presented from the controller to the connector (on export) and from the connector to the controller (on import).

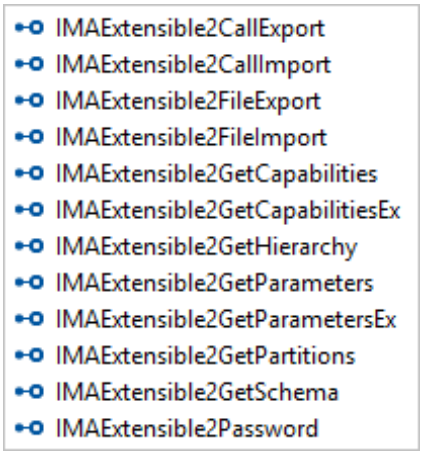
A drop file is not going to be an exact representation and interaction with the connector when you do not stage to the connector space. There are some scenarios, for example when an object is moved from an out-of-scope OU to an in-scope OU, where data from an earlier imported page affects what will happen later. When data is not staged to the connector space, the controller doesn't know that the data it needs is not complete. But for most scenarios, it is a good representation and used to understand how the sync engine works under the cover.

Microsoft.MetaDirectoryServicesEx.dll

All interfaces, methods, and properties for ECMA2 can be found in Microsoft.MetaDirectoryServicesEx.dll. It can be found in the ..\bin\assemblies folder. In the same folder, you also find a file named Microsoft.MetaDirectoryServices.dll. This latter file is only shipped for backward compatibility with MIIS2003. MIIS2003 did not ship with signed DLLs. If you compiled code using the unsigned DLL with MIIS2003, you cannot use a signed version without recompiling the code. The Ex version of the DLL should always be used for all new installations and in Azure AD Connect, the unsigned DLL is no longer available.

Interfaces

For configuring a connector, several interfaces are available.



ECMA2 interfaces in Microsoft.MetaDirectoryServicesEx

These can be broken down into some categories. There are interfaces used for configuring the connector, interfaces used at runtime, and finally password interfaces.

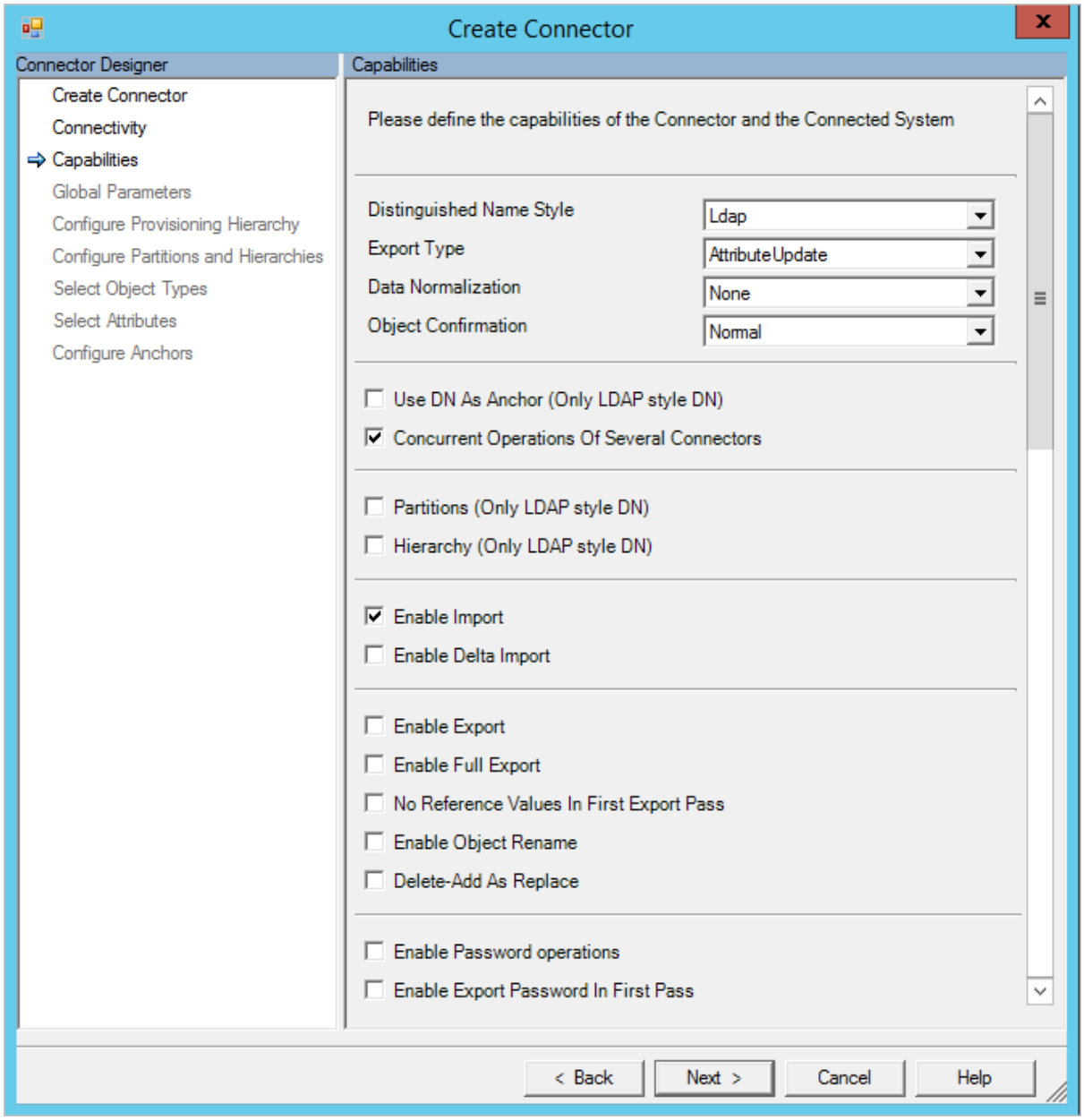
Configuration

Configuration interfaces are used during initial creation of the connector and for reconfiguring it.

There are two different interfaces for retrieving connector capabilities:

- IMAExtensible2GetCapabilities
- IMAExtensible2GetCapabilitiesEx

The first is used when you know exactly what your connector can support and you do not need any input from the administrator. The second interface provides information gathered from the administrator in the sync engine UI. As an example, this can be seen in the PowerShell connector from Microsoft. In this connector, all possible options can be configured by the administrator:

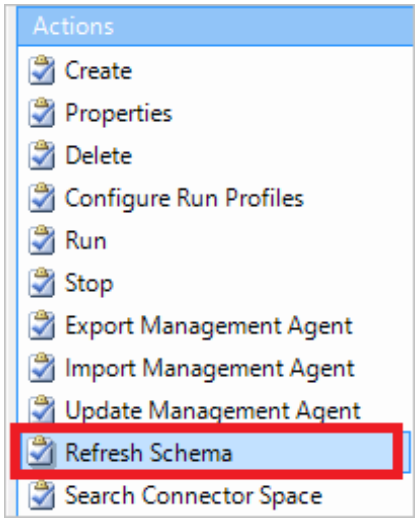


Capabilities as seen in the PowerShell connector

Then there are interfaces for defining the pages that appear during creation and reconfiguration. These are defined as parameter objects that should be drawn by the sync engine UI:

- IMAExtensible2GetParameters
- IMAExtensible2GetParametersEx

There is also an interface for returning the connected directory's schema to the sync engine. This interface is called on creation of the connector and when the administrator selects **refresh schema** in the sync engine UI.



The Refresh Schema action

- IMAExtensible2GetSchema

LDAP DN-style interfaces

When you configured the DN-style in the capabilities to be LDAP, then you have two additional interfaces you can implement and support:

- IMAExtensible2GetPartitions
- IMAExtensible2GetHierarchy

These interfaces are used to define the tree structure implied by LDAP. The partitions are the top-level structure and hierarchies are the children to partitions.

Import and export

There are four interfaces for working with import and export. Two file-based and two call-based. Of these, the call-based are preferred. The file-based interfaces were only added to allow backward compatibility with ECMA1. An ECMA1 connector (not using a hybrid approach of file-import and call-export) should be possible to recompile with ECMA2. For all other scenarios, the call-based interfaces should be used. File-based interfaces are not described any further - these are deprecated and not further developed.

- IMAExtensible2CallImport
- IMAExtensible2CallExport
- IMAExtensible2FileImport
- IMAExtensible2FileExport

The interfaces have three common methods:

- Open
- Get/Put
- Close

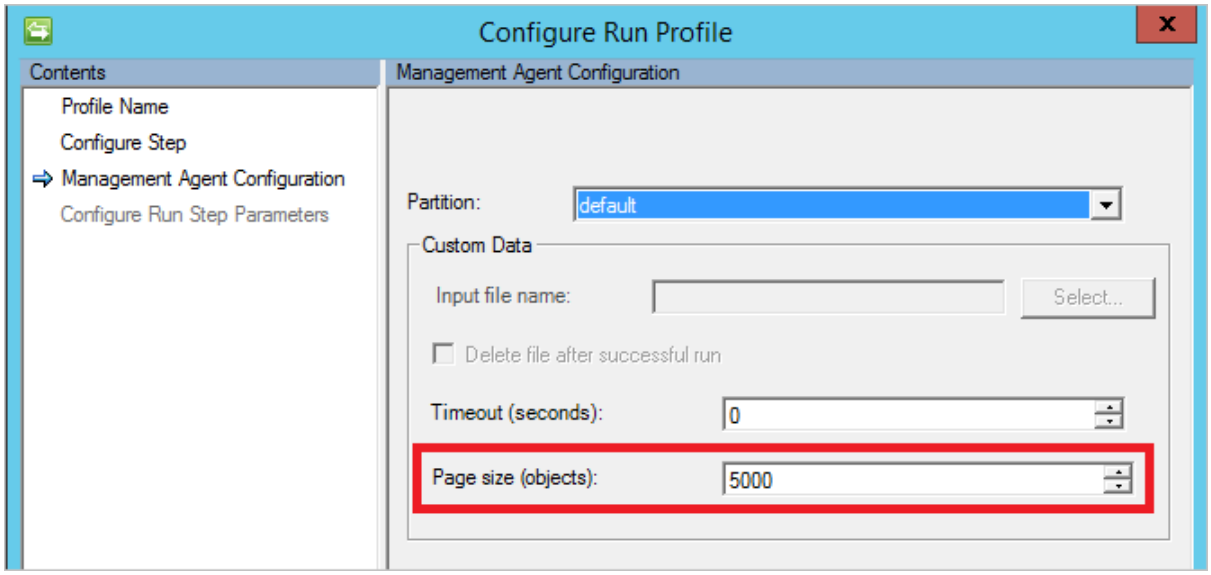
The purpose is to put everything you need to establish a connection in the Open method. Get/Put is used for the main processing. Close is used to clean-up and properly close the connection.

It might seem overly complicated and I have frequently seen code samples putting everything in the Get/Put method. The reason for why there are three is what happens when exceptions are thrown or the connector is forcefully stopped from the UI. In this case, the sync engine calls the Close method to give the connector the possibility to gracefully clean up and not leave open connections. That said, if your connector is only working with processing files and only supports full import, then there might not be anything to clean-up and easier to only implement a single method.

The interfaces also have these two properties:

- ImportDefaultPageSize/ExportDefaultPageSize
- ImportMaxPageSize/ExportMaxPageSize

These two properties determine the default value that should appear in the page (import)/batch (export) size property in the run profile and the max value the administrator can set it to:



The PageSize option in run profiles

Password management

The final category of interfaces is for password management. This interface is called to synchronize passwords from either PCNS or when set/changed using WMI.

- IMAExtensible2Password

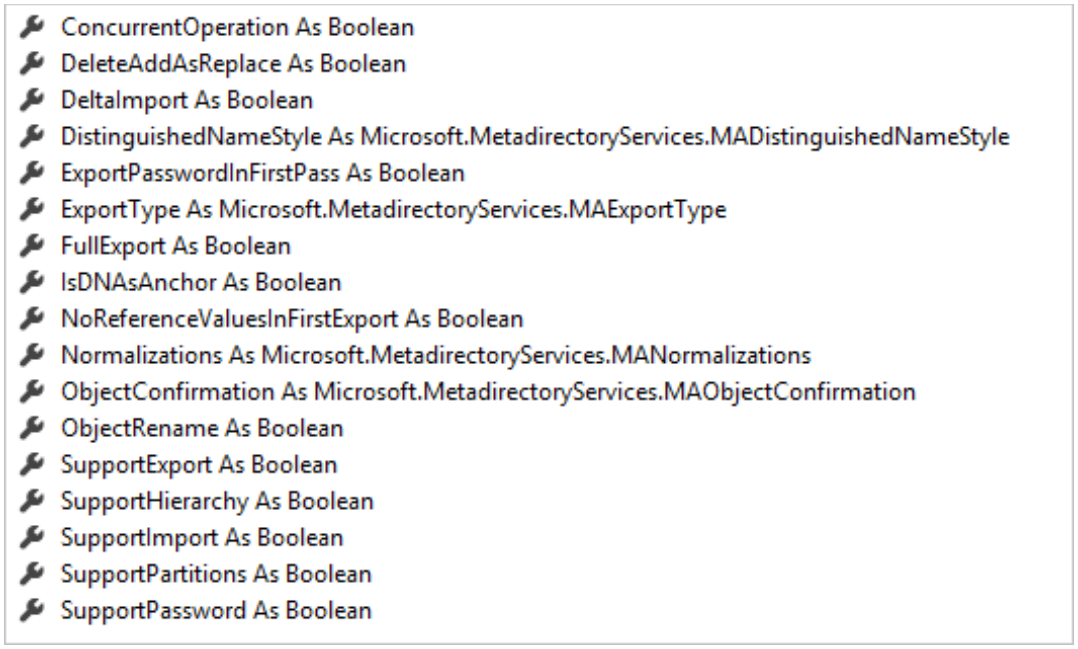
Capabilities

A connector informs how it wants data from the sync engine by defining capabilities. These are read when the connector is initially created.

You implement the capabilities in IMAExtensible2GetCapabilities or IMAExtensible2GetCapabilitiesEx. The latter is a later version and added to support more options without breaking backward compatibility with older ECMA2 implementations. It receives configuration parameters so it can take input from the administrator.

These interfaces are called when the connector is initially created. It should be assumed that the capabilities cannot be changed after the connector has been created, even though some actually would work. The capabilities page is not visible during reconfiguration of a connector, only during connector creation. But the method might still be called during connector reconfiguration so don't write your code assuming it is only called on creation.

These are the properties that can be returned from the methods:



Properties in MACapabilities

These properties can be grouped into categories.

- Basic properties that are core to how the connector should behave
- Import/export properties that defines details for how import and export should behave.

Basic properties

ConcurrentOperation

This Boolean instructs the sync engine if any other ECMA2 can run at the same time as this connector. It is intended to be used when you have a dependency on a library that could

only be invoked once. An example of this is the Lotus Domino connector. The Lotus Notes DLLs the connector is using can only be loaded once in memory and the sync engine can make sure only one Domino connector is running with this setting.

In ideal situations, this flag would only stop multiple instances of this ECMA2 connector, but it blocks all other ECMA2 connectors from starting.

The most common setting is going to be:

```
myCapabilities.ConcurrentOperation = True
```

DistinguishedNameStyle

The DN style can be set to the following possible values:

- None
- Generic
- Ldap

Read anchors and DNs for more information on the background of the concepts discussed in this section.

The value None indicates that there is only an immutable ID, the anchor, and that all other objects reference each other using this single value.

The value Generic indicates that the anchor and DN are different values. The DN is not hierarchical, but must follow the one-level DN standard `CN=value`. The connector might remove the CN= on export and add it on import if that is required. But in the sync engine, it should look like an LDAP style DN. The generic style implies only one partition (named "default") and no hierarchy.

The value LDAP indicates that the connector is having DN and anchor as different values. It also implies that the DN is hierarchical, allowing the connector to support partitions and hierarchies. Even though the name is LDAP, it does not have to follow any LDAP naming style, other than the generic `component name = Value, component name = Value`. The component name can be anything. It could be the well-known "CN, OU, DC" or something you make up. For example, the Generic SQL connector is using the LDAP style with the component name "OBJECT=". This allows it to use partitions (each object type is a partition).

As an example, this is how you might define this setting:

```
myCapabilities.DistinguishedNameStyle = MADistinguishedNameStyle.Ldap
```

IsDNAsAnchor

This Boolean is used together with DistinguishedNameStyle when the latter is set to LDAP. When the DN style is set to LDAP, it is assumed that the connected directory can return both an anchor value and a DN during delta import. Some LDAP directories cannot. For example,

Open LDAP has a GUID, which normally is a good candidate for an anchor, on each object but when you read the delta change log, the anchor attribute is not present.

By setting this property, it informs the sync engine that it should not assume an anchor attribute. The DN attribute is the only identifier.

The downside with no anchor attribute present is that an object rename cannot be supported. When an object is moved from one OU to another, then it looks like a delete of one object and an add of a new object to the sync engine.

Unless you need this functionality, then there is no need to specify this property. It is assumed that if not defined, you have a "proper" LDAP behavior with anchors and DNs.

Normalizations

This property can have the following values:

- None
- RemoveAccents
- Uppercase

If the connected directory cannot preserve upper/lowercase or accented characters, then this property can be set. For example, it used to be that mainframes always provided data in upper case. When you exported data with lowercase, it came back in the confirming import as upper case causing an export-not-imported warning.

A workaround would have been to use an uppercase function on every export attribute flow. Setting this property allows the sync engine to do the conversion for you. A connector cannot change data provided in an attribute flow, but the controller can.

The RemoveAccents option is probably even less useful. It removes accented characters and replaces these with unaccented characters on every attribute. It is quite common that you would want it for a single attribute, such as the email address, but not for every attribute.

I have not seen any connected directory in the past decade that does not support Unicode and where you have to set this property. All connectors I've seen have always used:

```
myCapabilities.Normalizations = MANormalizations.None
```

This property is in ECMA2 to provide backward compatibility with ECMA1.

Import/export properties

DeleteAddAsReplace

This is a setting telling the controller if you want a delete/add-operation sent as a single replace operation instead. You can set this setting to true when the connected directory does not store any history and it allows you to update the anchor attribute.

The default behavior should be to receive it as two operations - one delete followed by an add.

```
myCapabilities.DeleteAddAsReplace = false
```

DeltaImport

This Boolean indicates that your connector supports delta import and that it should be possible to add delta import run profiles to the connector. If import is supported, then it is required to support full import so there is no setting for that option.

```
myCapabilities.DeltaImport = true
```

ExportPasswordInFirstPass

This Boolean indicates if you want the password in the first pass or second pass. Most systems can and want the password to be set in the same operation as the create operation. But if you have a system where you first must create the object and then update the password in a second operation, then you can set this to false. The connector then gets one call with the add, without the password, and then an update with only the password.

The default behavior is:

```
myCapabilities.ExportPasswordInFirstPass = true
```

ExportType

This property can have the following values:

- ObjectReplace
- AttributeReplace
- AttributeUpdate
- MultivaluedReferenceAttributeUpdate

It is one of the more important properties to get right. This setting tells the controller how you want an update to an object presented to the connector.

ObjectReplace gives you every attribute on the object regardless if it has changed or not. You would use this option when you drop the entire object and recreate it every time. It is also used when the connected directory wants to calculate the difference. For example, when you interact with a stored procedure in SQL, you probably need to send in all attributes for the object or it is treated as the attribute should be removed.

AttributeReplace gives you only attributes that have changed. The important difference compared to the next option is that for a multi-valued attribute, all values are present even when only one has changed. That is, even if a single member has changed in the member attribute, you receive all members every time.

AttributeUpdate only gives you updated attributes. For multi-valued attributes, you get deletes and adds for each value that has changed. If your directory can support it, this is usually the most effective option to use.

MultivaluedReferenceAttributeUpdate is a hybrid of the two previous options and was introduced to support the behavior in Azure AD. It only provides changed attributes. For non-reference attributes it provides AttributeReplace and for reference attributes it provides AttributeUpdate. Except for Azure AD I have not seen any directory with this behavior so it is unlikely you will use this option.

```
myCapabilities.ExportType = MAExportType.AttributeReplace
```

FullExport

This Boolean indicates that your connected directory wants a full export, that is, it wants all objects every time and not only changed objects. When this option is enabled the Full Export step type appears in the run profiles.

Most connectors should keep this to its default value to false. When you enable this option, the connected directory is usually expecting a full flat text file and you only use Full Export and not the default (delta) export.

One scenario where this is useful is when you need to deliver data to a human for review. As an example, I once had a customer who wanted daily lists of all employees with their phone numbers so it could be delivered to the person handling the billing for all phones. This option works great with the PowerShell connector from Microsoft for these scenarios.

Full Export should only be used when the remote system cannot handle the default delta export. It should not be used for data reconsolidation when export and import is supported by the connected directory. That is, you should not use this option as a replacement for the normal import-sync-export process.

The default value should be:

```
myCapabilities.FullExport = false
```

NoReferenceValuesInFirstExport

This Boolean indicates that you do not want any optimistically references in the first pass and references should only be sent as updates, never as part of an add operation.

In most cases you want the references to be part of the add operation, which is also the default. But in some cases that would complicate the scenarios in the remote system. The FIM Service connector has this option set to true so administrators only have to deal with reference attributes in updates and not in both adds and updates. It simplifies the MPR configuration in FIM/MIM Service.

Usually you want to keep this to its default value:

```
myCapabilities.NoReferenceValuesInFirstExport = false
```


ObjectConfirmation

This property can have the following values:

- Normal
- NoDeleteConfirmation
- NoAddAndDeleteConfirmation

Usually, the sync engine expects that after an export, it can read the object again in a confirming delta import. That is the normal behavior for the sync engine. But that is not always the case for all connected directories. Setting this option correctly makes sure no exported-change-not-reimported warning occurs in the confirming import.

NoDeleteConfirmation indicates that a deleted object does not appear in a confirming import. An example would be a row in a SQL table using a column to track delta changes. When a row has been deleted in an export, there is nothing a confirming import can read in the SQL table.

NoAddAndDeleteConfirmation is more uncommon and indicates that the connected directory can not present added or deleted objects in the delta import stream, but it can provide updated objects. An example of this is the SharePoint UserProfile store connector. Only updated objects can be read in the delta import stream from SharePoint.

The default option would be:

```
myCapabilities.ObjectConfirmation = MAObjectConfirmation.Normal
```

ObjectRename

This Boolean indicates if your connector supports the DN being changed (when not using MADistinguishedNameStyle.None) in an update.

Most directories allow the DN to be changed, but in some cases, it needs to be treated as an immutable attribute. When this is needed, this property can be set to false.

Most connectors would use:

```
myCapabilities.ObjectRename = true
```

Schema

A schema is essential for the sync engine and connector to work.

There is a single function you need to implement and it is called on the creation of the connector and when the administrator selects refresh schema.

```
Function GetSchema(configParameters) As Schema
```

The returned schema is a collection of schemaType objects.

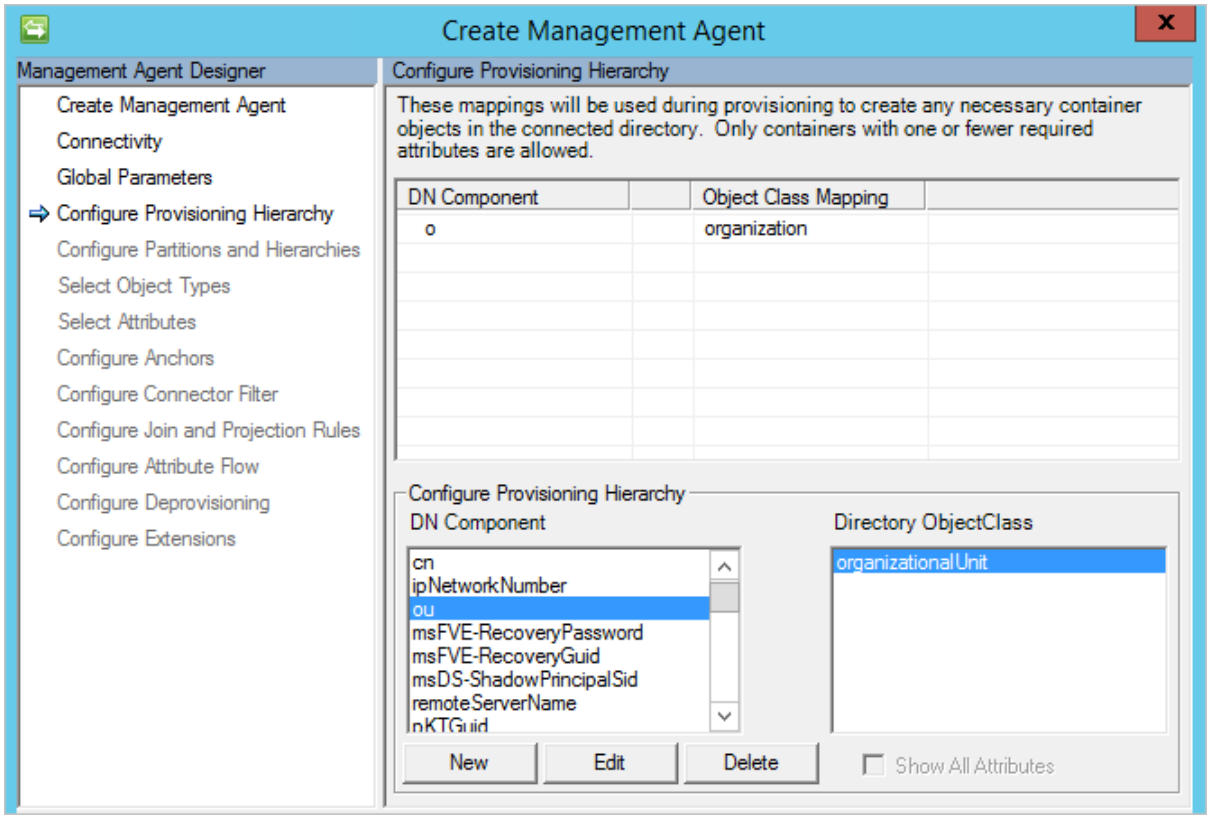
Objects

The objects are returned as a **schemaType** to the sync engine. On the object level, you return attributes and provisioning hierarchy information.

Provisioning hierarchy

This multivalued property, PossibleDNComponentsForProvisioning, works together with ExportActionProvisioningParent. With this structure configured, the connector can automatically create a structure in a target directory. For example, if the administrator provisions an object `cn=Bob,ou=Sweden,ou=Europe,o=Fabrikam` to the connector space, and an OU is not in the directory, the connector can create it.

For an object, you return the possible DN-components it can support, for example **ou** for the **organizationalUnit** object type. The administrator can then create a mapping between these two in the connector UI.



Provisioning hierarchy in Generic LDAP connector

During export, the connector receives requests to create OUs when needed. This is going to be a very bare-boned object with only a name, dn, and objectClass.

Attributes

In the returned structure, you indicate the attributes associated with an object. The primary key in the sync engine for the attribute is its name. When you return the same attribute for multiple different objects, you must return the same attribute type for all objects. This is

similar to an X.500 directory schema structure, which is how the schema is stored internally in the sync engine.

An attribute can be of the following types:

- Binary
- Boolean
- Integer
- Reference
- String

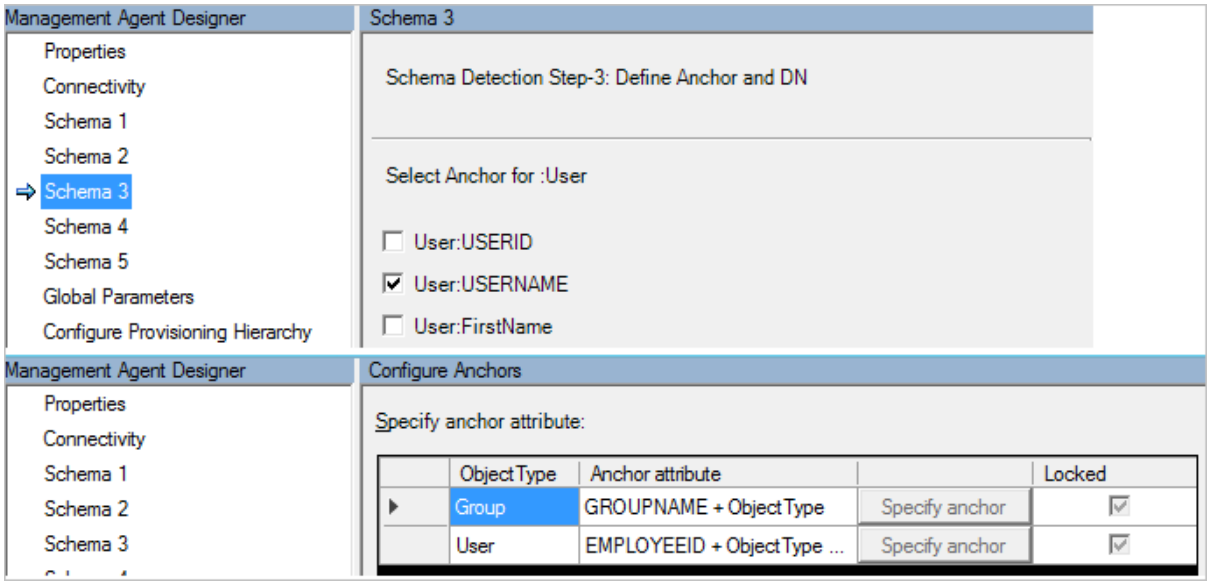
It can in addition to these types, it can be marked as multi-valued or as an anchor (anchors cannot be multi-valued). An attribute can also be marked HiddenByDefault. If this is set, then the administrator must select **show all** on the attributes page in the connector to see the attribute.

Anchor attributes

Anchor attributes are essential to the sync engine. You do not have to mark any attribute as an anchor attribute when you return the schema. If you do not, then the administrator is allowed to select anchor attributes at the end of the connector creation.

On the schemaType you can set the property **locked**. This option locks the anchor definition on the **configure anchors** page so it cannot be changed by the administrator. If you do not lock the definition, then the administrator can change the anchor attributes provided by GetSchema.

You can also implement a hybrid approach. In the Generic SQL Connector, the administrator is allowed to select which attributes should be part of the anchor. But the connector requires the objectType to be part of the anchor. For this reason, the administrator is asked for anchor attributes in the schema pages. The connector then adds its mandatory attribute and then on the **configure anchors** page, the option is locked.



Hybrid anchor approach in Generic SQL connector

Attribute operation type

An attribute can also be set with an operation type. If it is not defined, the sync engine assigns ImportExport operation type to the attribute.

- ExportOnly
- ImportOnly
- ImportExport

For example, connected directory system created attributes, such as whenCreated, should be marked as ImportOnly. Another example is userPassword, which should be marked as ExportOnly; no connected directory allows you to read and import this attribute. This setting affects the UI in how attribute flows are allowed to be configured.

Parameters

As a connector developer, you almost always need some information from the administrator to be able to read and write data from the connected directory. With the parameters interface, you can configure how these pre-defined pages are rendered so you can get required data.

These pages appear at pre-defined locations in the UI. There is usually an interface with a similar name the page is closely associated with. These pages are as follows:

- Capabilities
- Connectivity
- Schema (available with GetConfigParametersEx)
- Global
- Partition
- RunStep

There are two functions for returning configurable parameters to the sync engine.

```
Function GetConfigParameters(configParameters, page) As
ConfigParameterDefinition
Function GetConfigParametersEx(configParameters, page, pageNumber) As
ConfigParameterDefinition
```

The latter, with Ex, is used when you need a dynamic number of pages for schema discovery in the UI. In all other cases, the first method can be used.

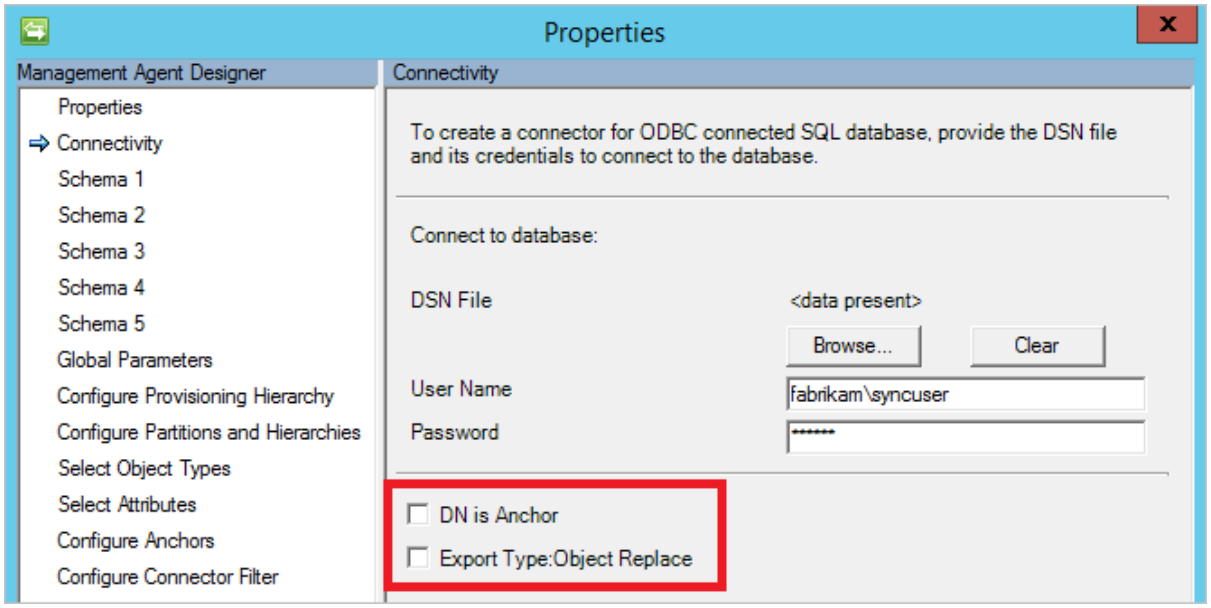
There are also two functions for validating the information.

```
Function ValidateConfigParameters(configParameters, page) As
ParameterValidationResult
Function ValidateConfigParametersEx(configParameters, page, pageNumber) As
ParameterValidationResult
```


Common scenarios would be to verify credentials and that required data is not empty.

Parameter pages

You don't need to have parameters on all pages and there is also some freedom for having a parameter on another page than the one with the same interface name. This freedom is mostly for the GetCapabilities interface. This is not called until after the Connectivity page. The reason is that in some cases you do not know what the remote directory can do until you have connected to it. In cases where you only need to ask the administrator for a single property, it might be better to put this on the connectivity page and not have a page with a single option. An example of this can be seen with the Generic SQL Connector.



Example of capabilities on the connectivity screen

Some pages have a special behavior.

The capabilities page only appears on connector creation. When editing or viewing an existing connector, this page is not visible.

The connectivity page appears when the connector needs to gather credentials in other places as well. As an example, when the administrator click refresh schema in the UI, the connectivity page appears to gather credentials.

Schema pages

The schema pages are different compared to the other pages. It is the only page that can appear as multiple pages. It is used when the configuration is complex, such as the Generic SQL connector. The Ex version of the interface was added to support the complexity of this connector.

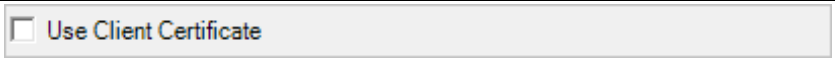
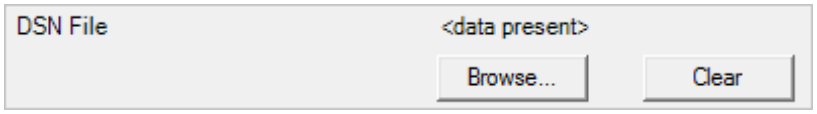
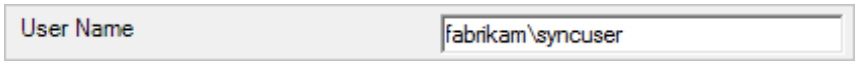
When the connectivity page has completed, the GetConfigParametersEx is called for schema with the pageNumber 1. The method can then gather some information from the

administrator, verify it, provide another page, and ask for more information. The number of pages is dynamic and one additional page is added until the method does not return any more parameters in a call.

Elements on a page

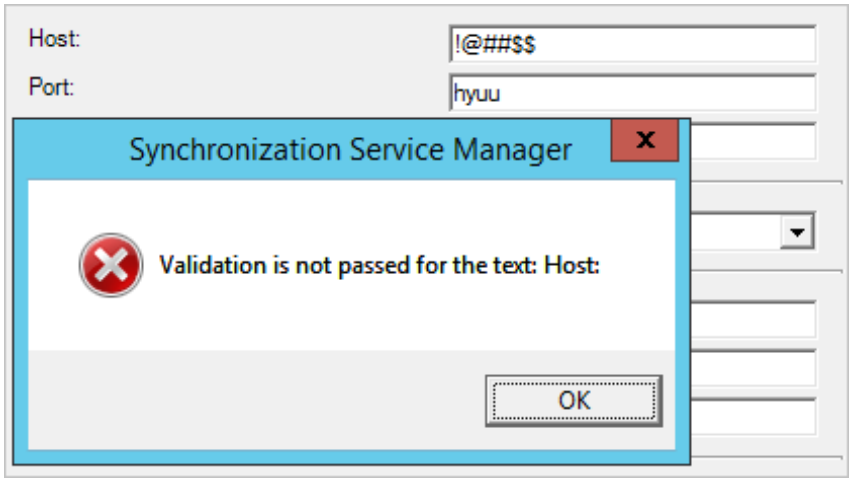
The method should return the elements it wants to be drawn in the sync engine UI in the correct order.

The following elements are available:

Name	Apperance
CheckBox	
Divider	
DropDown	
EncryptedString	
File	
Label	
String	
Text	

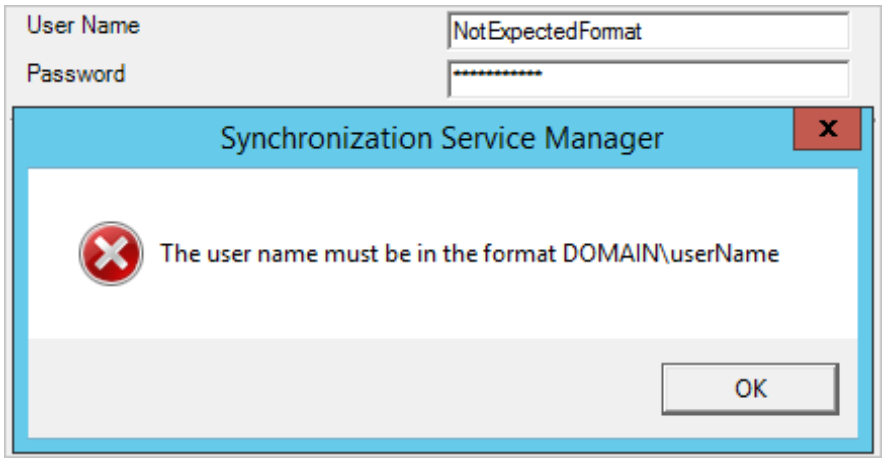
Most of these have an overloaded version so the default value can be passed in.

For string and encryptedstring, there is also a validation parameter. This allows for the sync engine being able to verify the format using the provided RegEx. This can be easy when you know the format for a field and the name of the field is obvious to the administrator. For example, a server name should not include special characters and port should only be numeric.



Bad data caught by validation

It is also common to just pass in empty in the validation parameter and do it all in the ValidateConfigParameters method instead. This would allow you to pass back a custom error message to the administrator.



Message from ValidateConfigParameters

Validate parameters

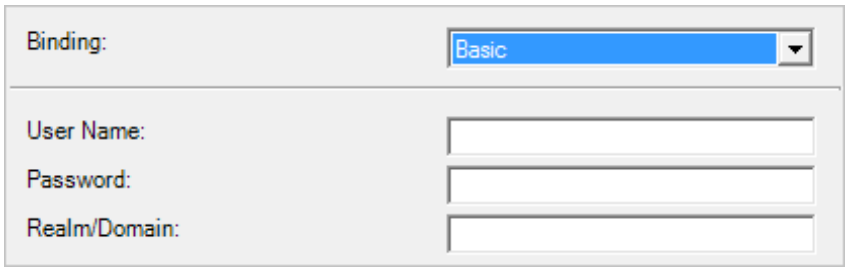
The validate parameters method allows you to verify the parameters provided by the administrator. If there is a problem with the data, then you set ErrorMessage on ParameterValidationResult. You can also set ErrorParameter indicating which parameter caused the problem, but that is optional. If everything is ok, then return success on the Code property.

Limitations with parameters

Some more advanced configuration options are not possible. You would have to work around these limitations or live with a non-optimal experience.

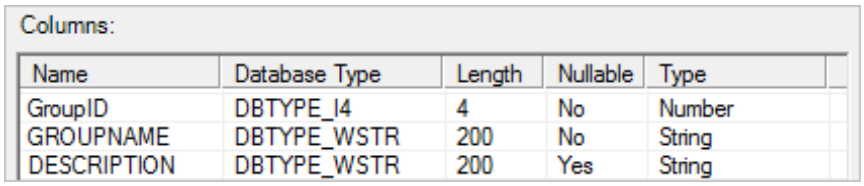
The most common feedback is that a page cannot be updated based on what the administrator selects. For example, in Generic LDAP there is a binding drop-down. If you

select anonymous, then you should not provide any credentials, if you select basic, you need username and password, and if you select Kerberos, you also need to provide the realm. But the page always has all three parameters and you have to document which to use for different scenarios.



All parameters are visible

The elements are also a fixed list and you cannot add your own element to the sync engine UI. For example, in the built-in SQL connector there is a grid that allows attribute configuration. In the Generic SQL connector, the same configuration is instead a long list of drop-downs, one for each attribute.



A grid from the built-in SQL connector not available in ECMA2

Partitions

Partitions are only available when using LDAP as the DN-style. In all other cases, a single **default** partition is used.

Partitions and hierarchies represent the tree structure found in LDAP DN naming. A partition is the highest level in the structure. It commonly represents a replication scope or object scope. The partitions should be considered to on the same level and with no dependency on each other.

What is a partition?

A partition is defined by its behaviors:

- All import and export operation should be possible to complete by communicating with a single server and a single set of credentials.
- When a full import is run, the remote server should be able to provide **all attributes** for **all objects** for the partition. If the server does return an object, the sync engine can treat it as an implicitly deleted object (known as obsoletion).
- When a delta import is run, the delta change log has in scope all objects provided by the full import.

- Shares the same object/attribute schema with all other partitions in the connector.

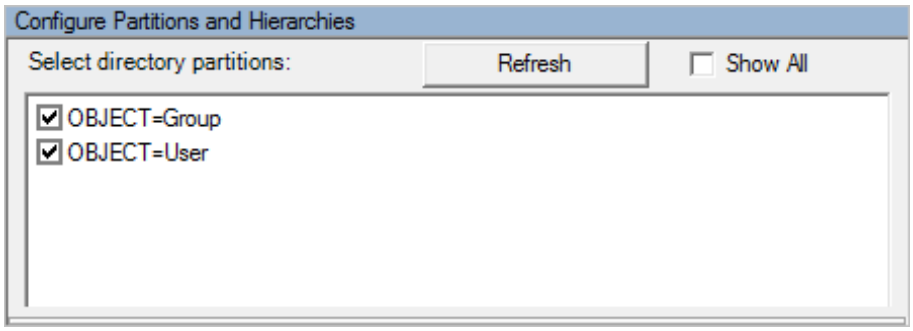
A partition is **not** defined by:

- The scope of reference attributes. Reference attributes can target any object in the connector space.

Some examples of partitions:

- In Active Directory, each domain is a partition. When talking to a domain controller, it can only provide all attributes for the domain it hosts. There is a Global Catalog on each domain controller, but it does not have a complete copy of all attributes. The delta change log is stored per domain and one domain controller cannot provide delta import for another domain.
- In the Generic SQL connector, each object type is a partition. When interacting with a SQL database, the expectation is that each object type is represented by a table.

The partitions show up in the connector on the **Configure Partitions and Hierarchies** page.



Partitions in the Generic SQL connector

GetPartitions

The signature for the single method is as follows:

```
Function GetPartitions(configParameters) As Partition
```

The return should be a structure of partitions. This structure should be self-explanatory. The only non-obvious property is HiddenByDefault. When set to true, it indicates that the administrator must select **Show All** to see this particular partition. This should be used for partitions that you do not expect should be imported/exported in normal situations. For example, in the ADMA, the configuration namespace/partition is hidden by default. The configuration does not contain any identity data so it is uncommon to include this particular namespace in an ADMA.

Partitions and DNs

Every partition is defined by its DN. The partitions can have names that imply a hierarchy. For example, in Active Directory you might have an empty forest root domain named **fabrikam.com** and then multiple domains following the structure **europe.fabrikam.com**.

When an object is provisioned into the connector space from the MV, the DN is used to determine which partition the object should be put in. When the partitions have names that are part of each other, an object is put in the partition that is deepest.

When running an import, all objects must match the DN in the partition that is currently running. The exception is reference attributes, that can reference objects in other partitions. These objects might not yet have been imported. In that case, a placeholder is created until the object has been imported.

Hierarchies

Hierarchies is only available when you use an LDAP DN-style. Hierarchies are the substructures to partitions.

GetHierarchy

The signature of the single method is as follows:

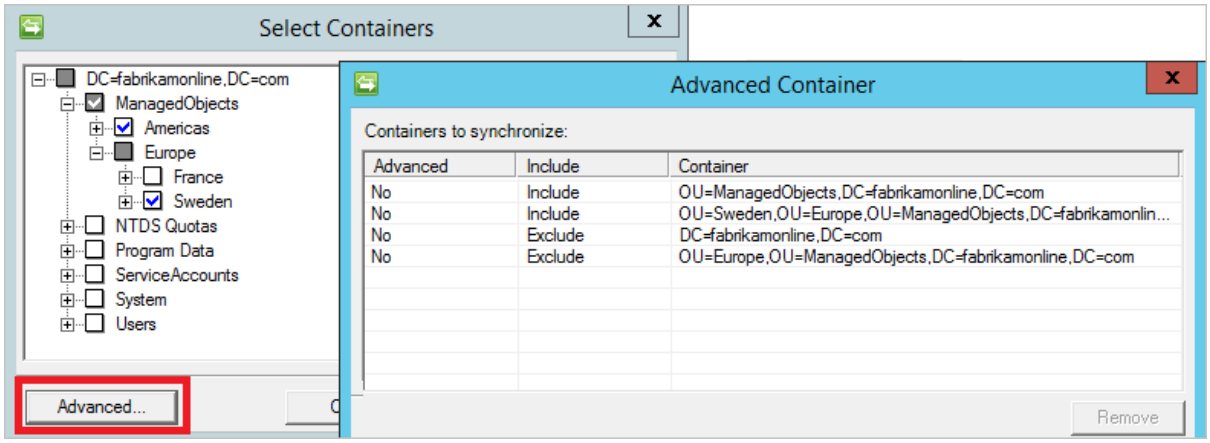
```
Function GetHierarchy(configParameters, parent) As HierarchyNode
```

This method is called from the sync engine UI to draw the content of Select Containers. As input parameters, you receive the usual configParameters and the DN of the parent. The method is then supposed to return the direct descendants of that node in the target directory. Every time the administrator clicks one of the plus-signs to expand a node, this method is called again. If there are no children, simply return nothing back and the UI removes the plus, indicating no children.

How hierarchies are represented

The hierarchies are stored as a number of included and excluded nodes. The easiest way to understand is by looking at an example. You can see the same view using the built-in ADMA and select and unselect OUs. Then click on Advanced to see how the configuration is stored in the sync engine.

As you can see, the entire structure is a list of included and excluded nodes.



Hierarchies in the Active Directory connector

Notice that the checkboxes in the UI have four states:

- Unselected and white background (for example "Users")
- Unselected and grey background (for example "Europe")
- Selected and white background (for example "Americas")
- Selected and grey background(for example "ManagedObjects")

These different states represent how new OUs should be handled.

- When selected, a new OU (for example "Canada" under "Americas") would be included.
- When unselected, a new OU (for example "Germany" under "Europe") would be excluded.

The OU "ManagedObjects" is included, but it has OUs under it marked as excluded. The OU "Europe" is excluded, but it has OUs under it marked as included. That is why these two have a grey background.

This structure of included and excluded hierarchy nodes is what is going to be passed into the OpenImportConnection method. You can then adjust your import code to use this information.

Import

To signal to the sync engine that you support import, you implement the IMAExtensible2CallImport interface. If you do not implement the interface, it indicates that you have an export-only connector.

You must support full import. Support for delta import is optional.

OpenImportConnection

The open connection method is used to establish a connection and prepare any configuration. It receives more information about the configuration than the other methods. Its signature is as follows:

```
Function OpenImportConnection(configParameters, types, importRunStep) As
OpenImportConnectionResults
```

ConfigParameters is a collection of all parameters configured on the Connector. These can be Global, Partition, or defined on a run step.

Types contains the schema, including the attributes that are selected as included by the administrator. In some cases, the target directory does not have many attributes so there is no performance problem in always asking for all of them. But in other cases that could be problematic and the connector should then only query for the attributes selected by the administrator.

ImportRunStep contains several important concepts a connector might need to worry about.

The ImportRunStep contains information about the hierarchy nodes selected. A connector can select to use this information or ignore it. When data during import is passed to the sync engine and it represents an object in an out-of-scope OU, the sync engine ignores it and does not stage it. If it makes sense to look at the hierarchy information depends on the target directory. As an example, the built-in ADMA does not look at this information and during a full import, it reads all information from AD. The technical reason is because it can quickly be a performance problem in AD to pass in too many OU filters to the AD APIs. The solution in avoiding performance issues is to read everything from AD and let the sync engine figure out what to keep and what to throw away.

The ImportRunStep also contains OperationType. The OperationType is defined so it can have the following values:

- Delta
- Full
- FullObject

Delta and Full are representing a delta or full import. The FullObject option requires a longer explanation.

OperationType FullObject

In OpenImportConnection this property can only have the values Delta or Full. This information was also supposed to be in GetImportEntriesRunStep but is unfortunately missing and requires a workaround.

First, we must understand the scenario. This can only happen when you use the LDAP DN-style and you have implemented hierarchies. In this case, during a full import, it is possible that some entries are thrown away by the sync engine since these represent objects in out-of-scope OUs. Then during delta import, a rename of an object from an out-of-scope to an in-scope OU is seen and passed to the sync engine. When this happens, the sync engine does not have any other attribute for the object than the DN. The object is put in the sync engines FullObject-needed list.

In the next call to GetImportEntries the FullObjectEntries structure is populated with objects where the sync engine needs all attributes.

Here comes the issue. Since the OperationType isn't present in the GetImportEntries structure, the connector must instead look at FullObjectEntries and if it is populated, then it must provide a full import (that is, all attributes) of these objects. The intention of the interface is that if FullObjects should be returned, then only the objects asked for should be returned and nothing else.

In GetImportEntries, you should think of your code working like this:

```
if (this.importType == OperationType.Full)
{code here}
else if (this.importType == OperationType.Delta)
{code here}
else if (this.importType == OperationType.FullObject)
{code here}
else
{throw new NotSupportedException("Unexpected import type: " +
this.importType);}
```

But rather than looking at OperationType.FullObject, because it doesn't occur in real life, you need to look at the presence of FullObjectEntries instead.

GetImportEntries

The workhorse for import is the function GetImportEntries. Its signature is as follows:

```
Function GetImportEntries(GetImportEntriesRunStep) As GetImportEntriesResults
```

The page size property is indicating the maximum number of objects you can return. You can return fewer objects. You do not have to fill up the pages to the size of a page. This can be convenient when you have to read objects by querying for all objects starting with A, then with all starting with B, and so on.

The entries are returned as a collection of CSEntryChange objects.

In this structure, you should return the ObjectModification type. In full import, this should always be "Add" indicating a full object. In delta import this can be "Add", "Delete", or "Update". But you don't have to use all of these in delta import. In some cases, you can only tell that the object has changed, but not which attributes. In that case, return all objects as "Add" with all attributes and the sync engine calculates the changed attributes.

On each object, you can also return an object-level error. If you do not set anything, it is assumed that the object was successfully read from the connected directory.

With each page, you should return moreToImport to indicate if there is more data in the connected directory to read. When set to true, the sync engine calls the method again to get more data.

If you support delta import, at the end of both full import and delta import you should also set customData with information about the watermark. You might store other information in this property, but the intended use is only to store the watermark and make sure you understand how errors affects it.

CloseImportConnection

This method is called after GetImportEntries has indicated that there is nothing more to import. It is also called when the run is stopped from the UI or an exception is thrown from the other methods. In this method, you should close the connection to the connected directory.


```
Function CloseImportConnection(importRunStep) As CloseImportConnectionResults
```

Export

Export is the process of writing data to the target system. If you do not implement this interface, then it indicates you have an import-only connector.

There are two kind of export: export and full export. The first "export" is delta export where only changed objects are processed.

OpenExportConnection

The OpenExportConnection is used to establish connection with the remote system. In many cases this code is going to be identical to the code in OpenImportConnection.

```
Sub OpenExportConnection(configParameters, types, exportRunStep)
```

You have access to configParameters and types (the schema). The exportRunStep contains similar information as importRunStep, but it is not likely you need to do anything in particular. The sync engine does not give the connector any unselected attributes or objects in out-of-scope OUs.

PutExportEntries

This method is called multiple times during an export. As described in the export passes, the same object can appear multiple times in the same export.

```
Function PutExportEntries(csentries) As PutExportEntriesResults
```

The information found in the csentries is different depending on how you configured the macapability ExportType. The object can have the action "Add", "Update", or "Delete". You also find information about deleted and added attributes, but when your connector always do a replace, then you would ignore the deleted attributes list.

In the results returned back, you can provide an anchor as received from the target system. In many cases you have a GUID from the target system that should be used as the anchor for the object.

On each object, you can also return an object-level error errors that tells the sync engine that it should present the data in some other way to the connector.

CloseExportConnection

This method is called after PutExportEntries has processed all objects. It is also called when the run is stopped from the UI or an exception is thrown from the other methods. In this method, you should close the connection to the connected directory.

```
Sub CloseExportConnection(exportRunStep)
```


Password

The password interface is similar to the export interface, but it has more methods to implement. This interface is used when PCNS synchronizes passwords to directories or when called directly through the WMI interfaces.

If your connected directory does not store its own passwords, then you should not implement this method. This interface is only relevant when your directory stores a copy of the user's password.

Open and Close password connection

These are similar to the open and close methods for import and export. One key difference is that the sync engine keeps a connection open for while in case another password is received. When a password is received, the open connection method is called. But after the password has been set in the target, the sync engine does not call close immediately. Instead, it keeps the connection open and when another password is received, it calls put password without doing an open first. It is only if it has not received another password for a while it calls the close connection method.

For very large environments and some systems, reusing a connection improves the performance significantly. In fact, this behavior was introduced for the Generic WebService connector and password sync to SAP. For very large customers (100,000 and more users), SAP was not able to keep up when the connection was opened and closed for every password.

For this reason, you should not try to implement the open/set/close logic in a single method.

If there is a problem establishing the connection, an exception should be thrown.

```
Sub OpenPasswordConnection(configParameters, partition) Sub ClosePasswordConnection()
```

Set and change password

These two methods are used to actually set the password in the target system.

```
Sub SetPassword(csentry, newPassword, options)

Sub ChangePassword(csentry, oldPassword, newPassword)
```

Set password is the most common method that is used. This is the method PCNS calls when a password should be synchronized to another system.

Change password is not used by anything delivered from Microsoft. It is exposed through WMI and you would have to create your own solution if you want to use it. In MIIS2003 there was a sample web portal that used this method to allow a user to change passwords in different target directories and where the password policy was different so PCNS couldn't be

used. Requiring users to remember different passwords for different systems is quite uncommon today and I have not seen this method used in real life for a very long time.

These methods can also get options passed in. These are:

- None
- ForceChangeAtLogOn
- UnlockAccount
- ValidatePassword

These can be passed in on the WMI interface. The main reason these are here is that the ADMA uses these. The FIM/MIM SSPR (self-service password reset) portal is setting the password using this interface on the built-in ADMA and it sets UnlockAccount and ValidatePassword.

These options are supposed to implement the following logic:

- ForceChangeAtLogOn The password must be changed by the user on first sign in. If the password has been set by an administrator, then this is set so the user is forced to change it.
- UnlockAccount If the account in the target directory is locked out due to many failed password attempts, then also unlock the account when the password is reset.
- ValidatePassword In AD when an administrator sets a new password, password history and some of the password policies are ignored. When this option is set, the connector should treat the password operation as if it was an end user request (think of SSPR). In this case, the password policy should be enforced. Otherwise, SSPR could be a way around the requirement to change passwords to a new password periodically.

If there is a problem setting the password, then you should throw an exception. There are several built-in exceptions to use for this situation:

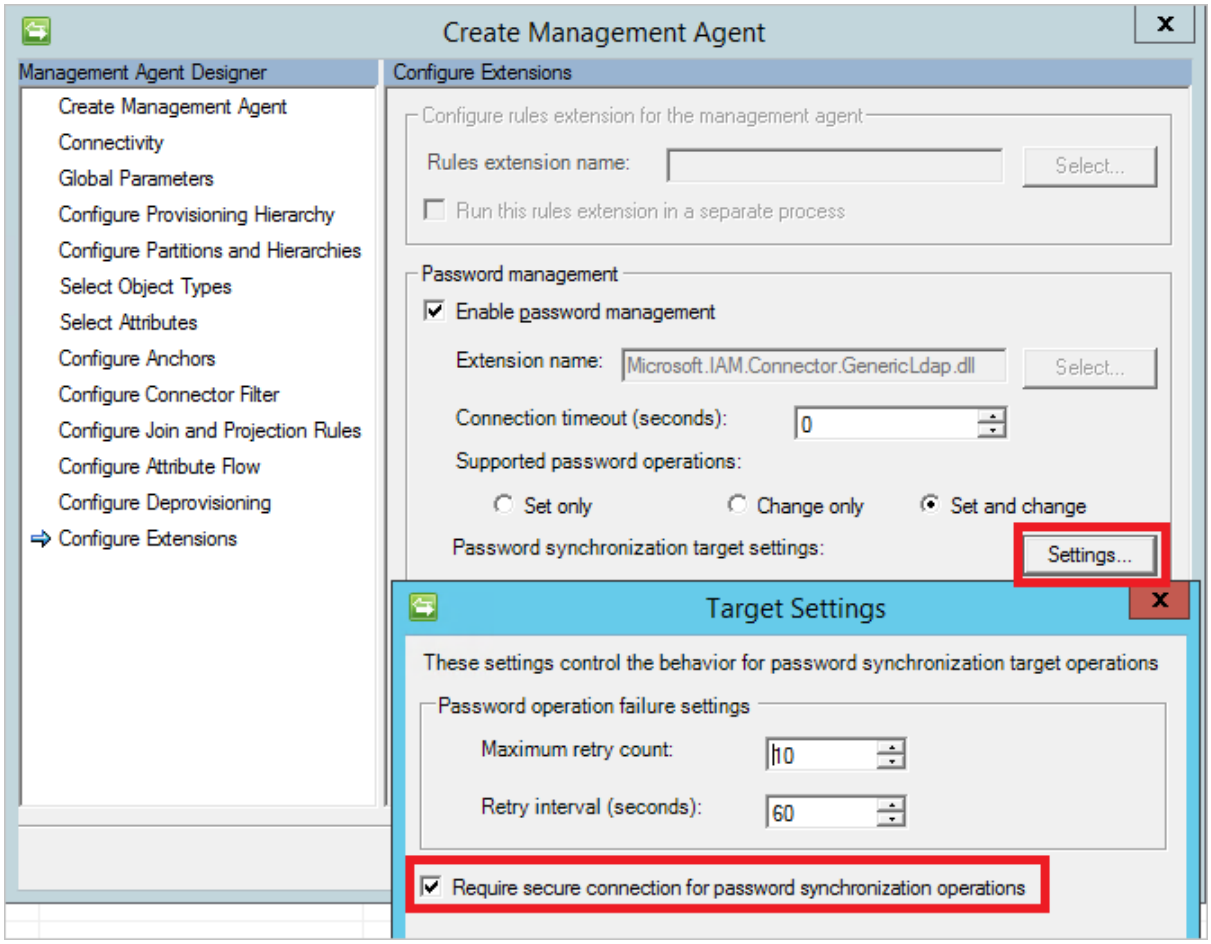
- AccessDeniedException
- BadServerCredentialsException
- DroppedConnectionException
- OldPasswordIncorrectException (for change password)
- PasswordExtensionException
- PasswordIllFormedException
- PasswordPolicyViolationException

Password security level

This method should return either secure or NotSecure to indicate if the password transport is over an encrypted channel to the target directory.

```
Function GetConnectionSecurityLevel() As ConnectionSecurityLevel
```

This setting works together with the password management setting on configure extension. The default setting is that unless you say that your connection is secure, then your connector does not receive any passwords from PCNS.



Require secure connection

Return values, Errors, and Exceptions

During import and export, the methods can provide error information to the sync engine. These errors can be returned on an object level or as a fatal exception.

There are some well-known exceptions that can be used, such as ServerDownException, to indicate a fatal problem that should stop the run from continuing. Unless the problem is one of the well-known exceptions, throwing one should be avoided. A custom exception shows up in the sync engine UI as a general failure and only the stack trace in the eventlog would provide additional information. It is instead expected that the error is returned on the object that caused the problem. Even an object-level error can stop a run.

An object-level error is returned by creating a `Microsoft.MetadirectoryServices.CSEntryChangeResult` in response to the object.

For password sync export operations, all errors are returned as exceptions.

The errors can be categorized into the following:

- Well-known exceptions

Export errors, defined in `Microsoft.MetadirectoryServices.MAExportError`

- Export errors, object-level
- Export actions

Import errors, defined in `Microsoft.MetadirectoryServices.MAImportError`

- Import errors, fatal
- Import errors, object-level

In the sync engine, there are both errors and warnings. A run in the sync engine can have 5000 errors before the sync engine terminates the run. There can be 100,000 warnings before the same happens. The connector can only return errors. Warnings are currently only available for built-in functionality in the sync engine.

Well-known exceptions

This is a list of the most common well-known exceptions that could be used in a connector to indicate a fatal problem.

- ServerDownException The server does not exist. Cannot even establish a connection to the remote server. This is the equal of a failed ping.
- BadServerCredentialsException The credentials provided does not work. The expectation is that the administrator changes the credentials, like resetting the password, before the connector is able to work again.
- FailedConnectionException The server is up (responds to ping), the credentials work, but for some other reason a connection cannot be established.
- DroppedConnectionException Used when the connector could earlier in the run connect to the remote server, but for some reason is no longer able to connect to it.

There are many other exceptions, but these have been superseded by object-level errors.

Export errors, object level

What used to exceptions in earlier versions of the sync engine, is in ECMA2 an error returned on the object that caused the problem.

This list can be found on MSDN

In addition, there are also two special errors that allow you to return a custom error text to the administrator:

- ExportErrorCustomContinueRun An error on the object, but the sync engine should continue to process other objects.

- ExportErrorCustomStopRun An error on the object. It is fatal and no other object should be tried to be exported.

When using these errors, you should also use the overloaded version of CSEntryChangeResult that allows you to pass back the errorName and errorDescription. The errorName is a short text that would appear in the operations log in the sync engine UI. It can be anything, but tradition says that spaces are replaced by dashes so it looks something like "invalid-characters-error". The errorDescription is additional information that allows the administrator to understand the problem in detail. This information appears as a pop-up in the sync engine UI when the administrator clicks the error link.

Export actions

There are some return values that do not indicate an error, but that the connector has detected a situation in the target directory that wasn't expected. But the sync engine should be able to fix the problem without involving the administrator to take an action.

These actions also have a corresponding object level error when an action should not be taken by the sync engine.

- ExportActionConvertUpdateToAdd The connector received an update to an object, but when looking in the target, it is no longer there. It might have been deleted manually in the target or lost in a replication conflict. An import and sync would stage the object to be recreated, but the connector can return this action instead. It would change it from an update to an add and in the next export batch, the object comes back with all attributes.
- ExportActionProvisioningParent This action is useful when the DN-style is LDAP. When the connector tries to write an object, it detects that the parent OU isn't present. With this action, the sync engine comes back in the next batch with information on how to provision the parent OU. When the parent OU has been created in the target directory, the original object comes back to the connector.
- ExportActionRetryReferenceAttribute This indicates that at least one reference attribute couldn't be written, but all non-reference attributes succeeded. It tells the sync engine that the object should be retried in the next [export pass].

Import errors, fatal

There are two import errors that are supposed to only be used in delta import and to indicate that the delta change log is broken. Using one of these would be similar to an exception and would stop the sync engine from trying to continue delta import. These errors were initially introduced because there is one LDAP server (not naming any names here) that under load would corrupt its own delta change log. The server would then return these particular errors back on an object when trying to get the delta change. The expectation is that a full import is needed to get a new watermark so the bad section in the delta change log can be skipped.

- ImportErrorParseError

- ImportErrorReadError

Import errors, object-level

An error on an object-level is also known as a discovery error.

This list can be found on MSDN

In addition, there are also two custom errors that can be used. These work like the corresponding export error.

- ImportErrorCustomContinueRun
- ImportErrorCustomStopRun

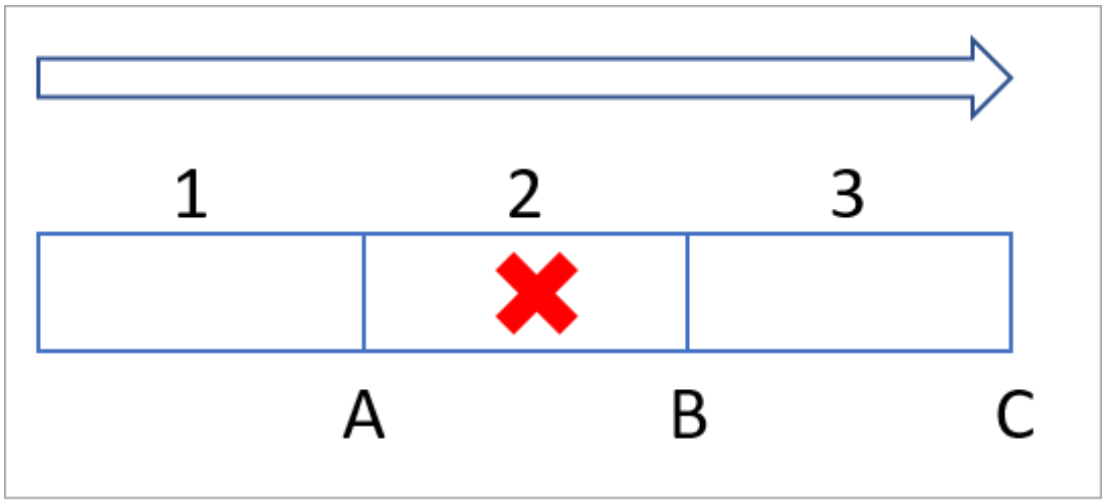
When an error is returned on an object during import, it also implies a few other things to the sync engine.

- During full import, object obsoletion should not occur at the end of the import.
- During delta import, the watermark should not be preserved beyond the object with an error.

Watermarks

An error during delta import needs some additional explanation.

To understand the scenario, look at this picture.



Watermark processing

During import, page 1 is read and watermark A is returned to the sync engine. The connector then reads page 2 from watermark A, but in it there is a problem with an object and a discovery error is returned on the object. The connector returns watermark B to the sync engine. Then the connector reads page 3 and is given watermark B from the sync engine to continue from, it is successful, and returns watermark C to the sync engine.

But, when the delta import operation has finished, the sync engine commits watermark A to the database. When the next delta import run occurs, it will again try page 2, fail, and continue with page 3 and the new page 4. This allows it to be able to continue to process new changes in the target directory. If this continues without the admin taking any interaction, the delta import process gets longer and longer over time. If the target directory is one where the delta change log is pruned over time, then delta import stops working when that happens and watermark A is no longer valid.

The most likely solution to the problem is to run a full import. After the full import has completed successfully, watermark C is committed to the database.

Sync engine behaviors

Sync engine behaviors

This section is currently a collection of small topics that didn't fit any other location.

Anchors and DNs

Each object has two different identifiers: an anchor and a dn.

Anchor

The anchor is used to identify an object in the remote directory and the definition is "immutable during the lifetime of an object". If the anchor attribute is changed, it should be treated by the sync engine as a delete of the object (and an add of a new object).

In the best case during delta import, the anchor is provided by the remote directory on every change (regardless of the DN). The reason is that it allows the sync engine to understand out-of-order changes reading a delta change log. As an example, think of Active Directory. During delta import, both the anchor and DN are provided in the delta stream. If the sync engine sees that the anchor value does not match the expected value for the DN, it assumed that there was a rename (or delete) of the original object and it hasn't seen that change yet. In Active Directory with replication, this can happen. The sync engine creates a "phantom" waiting for the other change to come it so it can resolve what to do with the older object.

Return anchor during export

When an object is provisioned into the connector space from the metaverse, the DN must always be provided. But the anchor might not be known. The anchor is commonly a GUID generated by the remote directory. A connector has the ability to return the anchor value back to the sync engine on export.

DN

The distinguished name (DN) attribute is used for how other objects should reference the object.

The connector space

Even if you use an ECMA2 DN-style that is not using DN, in the connector space database, the anchor is copied to the DN attribute by the sync engine to ensure both attributes are populated. In the sync engine database, the internal requirement is actually not what you might think - DN is required and anchor is not required.

Export passes

When the sync engine is about to export an object, it could be that it has a reference to another object that has not yet been exported. When it sees that a reference has no chance to be successful to be exported, it removes it from the export. It then put the object in the list of objects that should be retried in a second pass of the export. In the second pass, it comes back to the object and sets the reference with an update. During a single export run, the same object might be processed several times.

Assuming a completely new connector space with staged exports, a common processing sequence would be:

- First Pass: Add all objects, but do not export references.
- Second Pass: Update all objects with the references. With all objects being added in the first pass, the second pass should be able to set the references successfully.

Optimistically exporting references

As it might sound in the introduction of this topic, references are always sent as an update. That behavior can be controlled by the capability flag NoReferenceValuesInFirstExport. When set to "false", which is most common, then reference attributes to objects exported in an earlier batch are exported already in the first pass.

The notable connector where this is set to "true" is FIM Service. This connector always sends the add in the first pass and reference attributes in the second pass. The reason is to make troubleshooting and development easier, even if it might sound like the opposite. A connector might end up with multiple passes without the administrator realizing it. All implementations must consider this scenario and with this behavior, the FIM Service policies have to consider and implement support for this scenario.

Ordering of objects in an export

As a general rule, it is assumed that an object with many references is more likely to include objects not yet exported, causing the object to be pushed to the second pass and retried later in the same export. To avoid this as much as possible, the sync engine at the start of an export sorts all objects based on the number of references they have. The chairman of the board (with no manager reference) and groups with no members are exported first. It then continues with all normal users with only the single manager attribute set. Then the groups are processed, with the largest groups coming at the end.

This is the reason why during export it feels like the sync engine is being slower and slower. As an example during initial sync with Azure AD Connect, it is fast in the beginning with only empty groups to export. But when it comes to all the groups with 40k-50k members, it feels like it is coming to a halt (causing a support call to Microsoft).

ECMA2 and references

During export, it might be that even though the sync engine thinks a reference attribute should succeed when writing to the target directory, it actually fails. In this case, the code should return the object-level error `ExportErrorReferenceAttributeFailure`. This tells the sync engine that all non-reference attributes were successful, but the reference attributes were not. It will put the object to be retried in the next pass. If there are multiple reference attributes on the object, then this error should be returned if at least one fail.

The four passes

In most cases, the connector only has to run two passes to be successful. But in the worst case, the connector might try the same object up to four times during the same export.

- First pass exports the add and update of all non-reference attributes.
- The second pass exports all reference attributes as a single update.
- The third pass exports each reference attribute by itself.
- The fourth pass exports each reference value by itself.

This algorithm allows the sync engine to slowly find out which attribute and value that is causing the export problem. Most connected directories cannot provide information on what data is causing the export problem. But the sync engines slowly tries to export all data that is "good" so only problematic data is left for the administrator to investigate.

Most connectors, not even the built-in, implements the reference retry error for all passes so it is in real life unlikely that you see all four passes in a single export.

Export and statistics

Since an object might be processed many times during a single export, the export statistics might be confusing. It could look like that many more objects were processed than expected, but that could simply be because of the multiple passes behavior.

Delete-Add

A delete-add is a special operation in the sync engine. It occurs when there is a delete operation staged for an object and at the same time there is an add for another object with the same DN. In this case, the delete must be processed before the add, or it will obviously fail.

In case the delete operation fails, then the add will be suppressed and not sent to the connected directory.

You want this behavior when you need to change the anchor of an object or for some other reasons want the object to be refreshed. For example, you might have a corporate policy saying that an employee who becomes a contractor must have their AD account deleted and recreated so no permissions assigned to it directly are preserved.

