Implementing Forefront Identity Manager 2010

Student Manual

Module 3: More About Synchronization

Table of Contents

Module 3: More about Synchronization

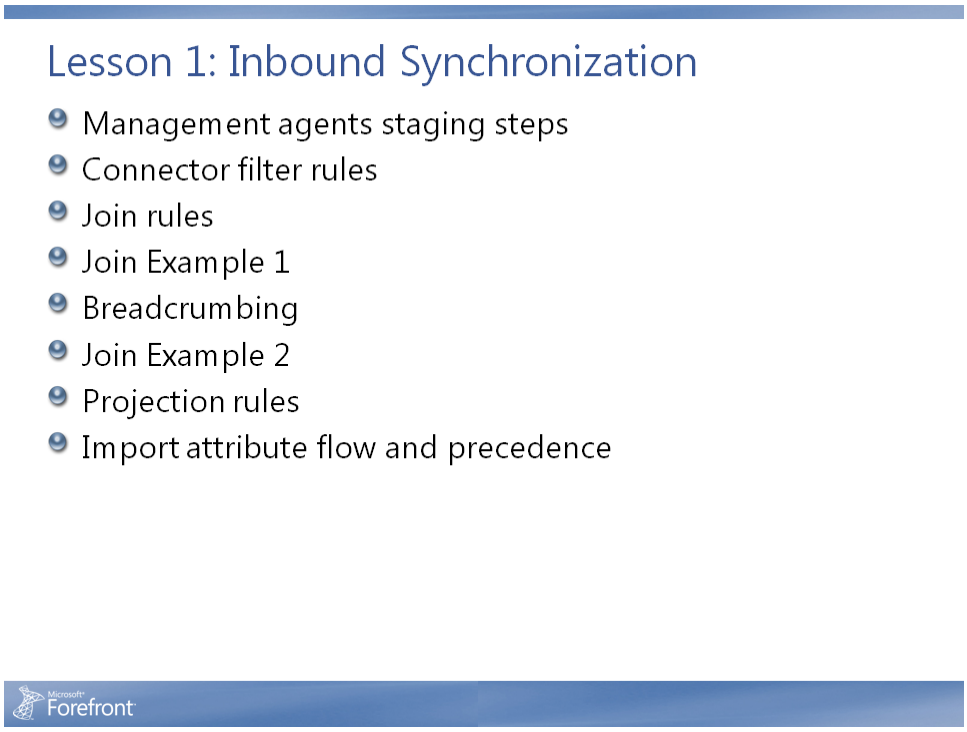
Module Overview

Think about synchronization at two levels: there is synchronization at the level of "what the synchronization service does"—it synchronizes data sources, but there is also the part of the process called full or delta synchronization, as opposed to the import and export part of the process.

Talk of import must distinguish between the staging part of import and the synchronization part of import. These minor issues of nomenclature occasionally trip up the new user.

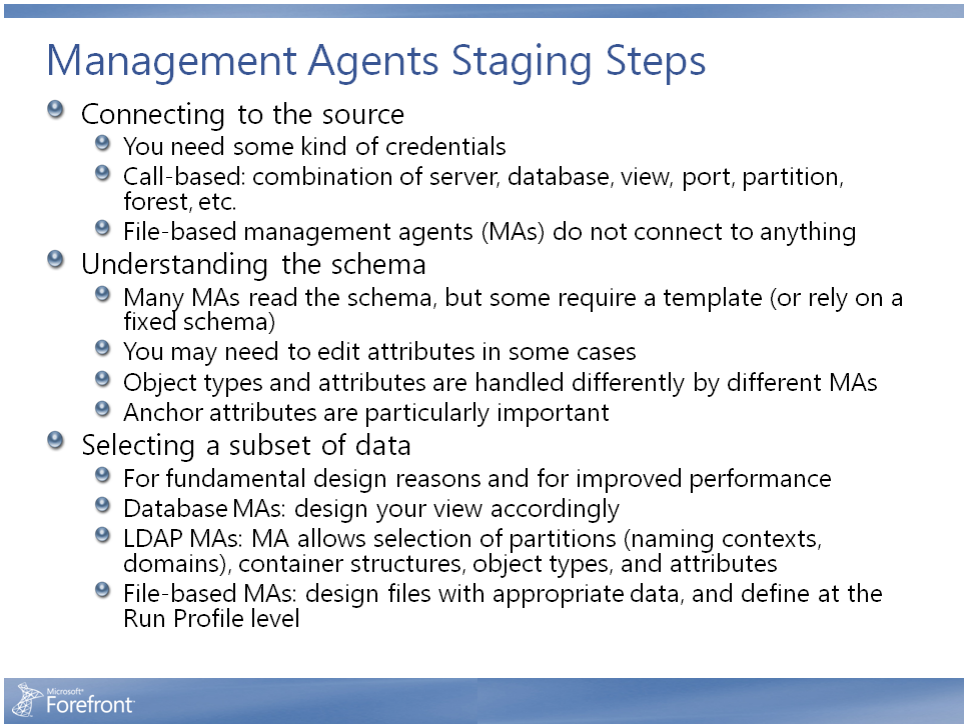
In the last module you started to get to know the interface enough to use it for a simple import and synchronization. In this module you'll focus a little more on the internal workings and fill some of the gaps, digging a little deeper into features that were only mentioned in the last module.

Lesson 1: Inbound Synchronization



You have already imported data and seen it projected into the metaverse (MV) followed by import attribute Flow. The lab in Module 2 involved a data source (HR data) that is authoritative for object creation. Now you must consider sources that are not authoritative for object creation, which can provide attributes that will enrich the identity data already in the MV. This requires joining—the major topic of this lesson.

Management Agents Staging Steps



When you define a management Agent (MA), you define staging steps that vary considerably between different types of MAs, and synchronization steps that (with the exception of the very special Microsoft® Forefront™ Identity Manager 2010 (FIM) Service MA—more later) are always the same.

Connecting to the Source

You must always provide some kind of credentials, plus:

- **Database MAs** generally require a data Source name (or server name plus a database name) and the name of the relevant table or view containing the data to be processed. A view is generally preferable to a table, as it provides a level of abstraction between source data and FIM. A view lets you pre-select both the dataset to be processed by FIM and the attributes that are available to FIM, but it also means that if the underlying table(s) change, you don't have to reconfigure FIM (you may or may not need to modify the views concerned). The table or view you specify for full import is also written to during export. Not all views can be written to in this way—a detail that will have to be taken into account during design.
- **LDAP MAs** such as Active Directory® (AD), eDirectory, and Sun ONE Directory Services, typically require the specification of a server and TCP port to which to connect. AD is a little more complex, requiring a forest and domain, and providing for preferred domain controllers. You can generally specify secure communication where available (for example SSL, Sign & Seal, etc.)
- **File-based MAs** do not connect to anything—there is no equivalent here—though you do provide the name and location of a template file so that the MA can learn the schema.

Understanding the Schema

Many MAs read the schema, but some require a template. As already discussed, you may need to edit attributes in some cases (for example to indicate which ones are references). In file-based MAs you may even want to specify the data type and the length of the data (minimum and maximum). Lotus Notes has a fixed schema (you can modify it manually).

Object Types and Attributes are handled differently by different MAs. LDAP MAs (such as AD) allow you to pick object classes and attributes from a list. With database MAs, you define a view to contain the appropriate records and fields; all of the attributes discovered are then processed. Similarly, the columns or attributes discovered in a template file will determine which attributes are imported and exported by a file-based MA.

Anchor Attributes are worthy of particular (extra) note. The anchor attribute contains the unique value that links an object in the data source to its connector space (CS) equivalent. An MA can make a more or less educated assumption about an Anchor attribute, but you may have to define it. For example a Microsoft® SQL Server™ MA will offer (as a default) the primary key of the source table, if it is defined, but you can override this if necessary (by the way, this default won't work when a view is used). You can assume that other database MAs behave somewhat like this (for example, Oracle).

AVP, Delimited, or Fixed-width MAs

In AVP, delimited, or fixed-width MAs you must define the anchor. It is a reasonable assumption that other text MAs behave like this.

AD MAs

In an AD MA, according to the user interface the Distinguished Name (DN) is treated as the anchor (and indeed, during provisioning, a unique DN must be generated just as an anchor must be generated for most other MAs). However, the way the MA actually keeps track of AD accounts is through the AD GUID, although this takes place under the covers, and you don't actually see this. In this way, a DN can be changed in AD, resulting in a rename at next import. Renames cannot happen in simple anchor cases such as SQL Server or AVP. You can assume that other LDAP-based MAs behave much like this (for example, AD LDS, Sun ONE, Lotus Notes, eDirectory).

LDIF and DSML MAs

These must contain a DN attribute, and you must either define this DN as the anchor attribute or select another attribute as the anchor. The full explanation of this isn't appropriate here, but briefly: if you have the DN as the anchor as well, it isn't possible for FIM to detect a rename (that is, if the object has moved, FIM can't keep track of it). Renames can be recognized through special MOD DN and MOD RDN change types.

Selecting a Subset of Data

For both fundamental design reasons and for improved performance, you may want FIM to process only a subset of the data stored in the data source.

Database MAs

For database MAs, you import all the records and all the fields (columns) in the specified table or view. Therefore, you should design a view to present the appropriate data subset for the MA.

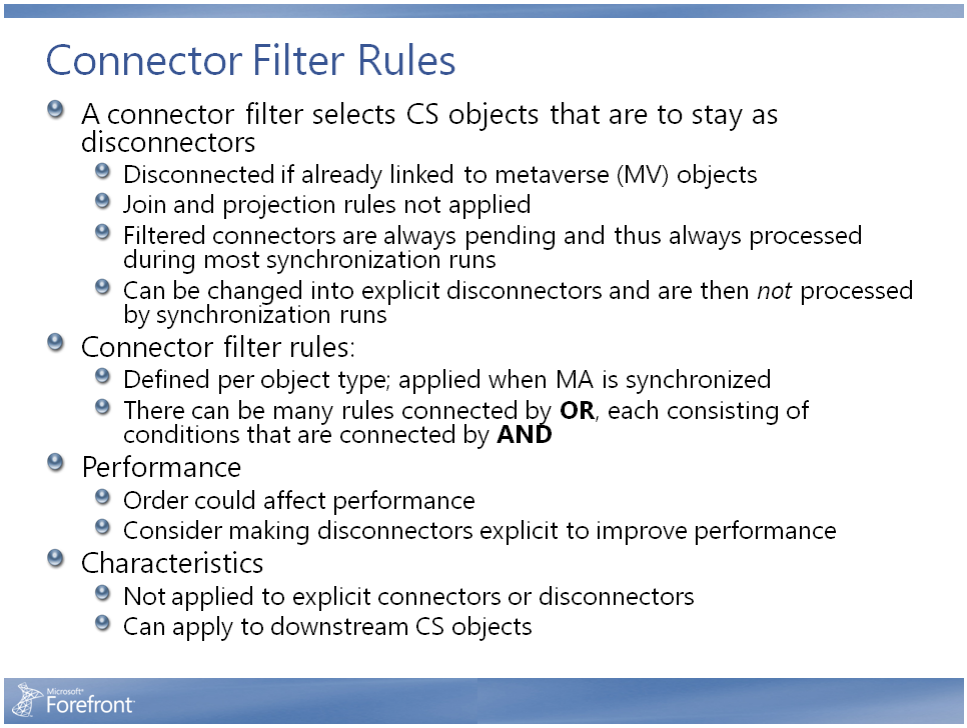
LDAP MAs

For LDAP MAs you may need to define which partitions (for example, naming contexts or domains) as well as the hierarchical container structures within those partitions, and the MAs fully support this. You then select the object classes to process and their attributes.

File-Based MAs

For file-based MAs, since a different file is typically used for export and different import runs, the file to be used is specified in the run profile selection (rather than in the MA itself). Such import files are processed in their entirety, so configuration of a data subset for file-based MAs is performed by the process that generates the text file for the MA.

Connector Filter Rules



A connector filter determines whether an object should remain a disconnector object in the connector ace. Thus, the connector filter prevents these objects from being further processed by the synchronization engine, and even disconnects objects that are already connected (with the exception of explicit connectors—those that have been manually joined). Connector filters are not required—they are used to prevent unwanted objects from being synchronized with the MV.

Subsets of Data

When you think of filters you tend to think of subsets of data, and a clear distinction needs to be made between the data subset that is imported (staged), as already discussed, and the subset of staged data to be held in the CS as disconnectors.

An example of use would be where you have objects in the CS that, while present in the data source, are not active in any way and therefore do not need to be represented in the MV. This could be filtered out at source, and therefore not imported, but this may not be convenient or even achievable.

Connector Filter Rules

You define connector filter rules when you configure an MA. The connector filter is defined per object and is evaluated during synchronization before attempting to join or project a disconnector, and is also applied to connectors (disconnecting them if a filter rule evaluates to true). Explicit connectors and disconnectors are not affected by filter rules.

Filtered Disconnector

If a disconnector satisfies the connector filter rule, it becomes a filtered disconnector. If a connector satisfies the connector filter rule, it is disconnected and is reported as a filtered connector, but it is a

disconnector (and will likely be a filtered disconnector for future purposes). In either case, it is not processed any further by the remaining MA rules. A filtered disconnector can be changed to a connector if it is manually joined or projected (using the joiner), or if the connector filter rule or the data changes in such a way that the connector filter rule is no longer satisfied.

Parts of a Connector Filter Rule

A connector filter rule consists of a number of conditions, all of which must be satisfied for the rule to be satisfied (ANDed conditions). Each rule is processed in the order you specify until one is true, or they are all tested (ORed rules), so there is scope to optimize performance by choosing the order intelligently.

Performance

Non-explicit disconnectors are always pending, that is they are processed during every synchronization run whether delta or full (except for the special case of a combined delta import and delta synchronization, which only processes the deltas just imported), so they are tested against the filter rules very often. You may be able to improve performance by ensuring that a rule that catches lots of them is processed first, avoiding the processing cost of testing against the other rules.

It may well be that you review disconnectors, and while doing so you might note that particular ones will never need to become connectors, and so to avoid this repeated processing altogether, you might convert them to explicit disconnectors.

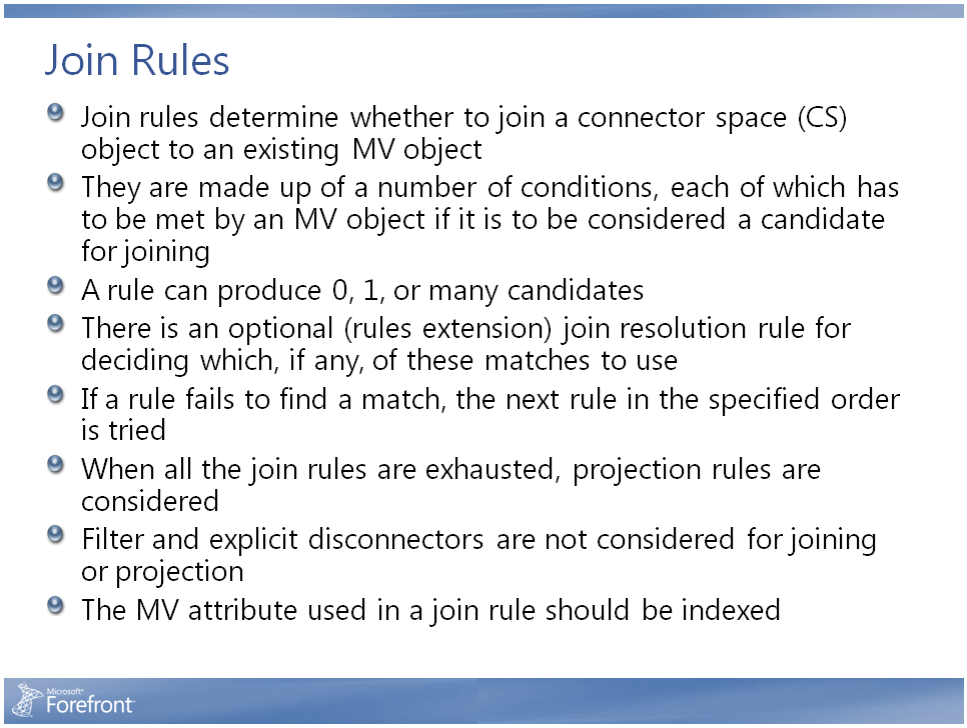
Characteristics of Connector Filter Rules

The characteristics of the connector filter rules are as follows:

- When an object is first imported, it is evaluated against the connector filter before join/projection is attempted. If the object satisfies the filter, then the sync engine will mark that object as a filtered disconnector and will not attempt to join/project the object. Conversely, if the object does not satisfy the filter, then the sync engine will pursue join/projection actions for the object.
- During delta synchronization, the connector filter will be reevaluated for all connectors for which changes have been detected and for all disconnectors. If it is a normal connector, and it now satisfies the filter criteria, the join will be broken and it will be turned into a normal disconnector. If it is a normal disconnector and it no longer satisfies the criteria, it will be sent through the join/project rules for processing. However, if a delta Import and delta synchronization run is done (that is, a single step), only those changes imported in the delta import will be synchronized.
- A full synchronization run will apply the connector filter to all normal connectors and disconnectors, handling them as described before.
- The connector filter is not applied to explicit connectors/disconnectors because their explicit status exempts them.
- When any one of the following export scenarios occurs, the connector filter will be applied to the outgoing CS object before it is pushed out to the data source:
    - A new object is provisioned.
    - An object rename occurs.
    - Export attribute flow is performed.

If the CS object satisfies the connector filter, an error will be generated and the transaction will be rolled back. This maintains data consistency by avoiding the export of changes to a data source that would lead to subsequent disconnection when objects are re-imported.

Join Rules



Join rules determine whether there is an existing MV object to which to join a CS object. If the join criterion is met, the CS object is linked to that MV object. A join rule can be supported by a join resolution rule (which is always a rules extension rule, and beyond the scope of this course).

A join rule is made up of one or more conditions which compare CS object attribute values and MV attribute values, looking for matches. As each CS object is considered, if all conditions are met for a given MV object then that object becomes a candidate for joining. If this is not the case, the next rule in the specified order will be tested, and so on.

The following table shows the action taken when a rule returns a result.

Rule Returns this Result	Action Taken
None of the MV objects are acceptable	The next join rule in the list is evaluated. When all are exhausted any Projection rules will be applied
Exactly one of the MV objects is acceptable.	The join resolution rule is applied, if any. If none, the CS object is connected with the MV object
More than one of the MV objects is acceptable.	The join resolution rule is applied, if any. If none, the join operation fails. Use preview to diagnose the failure, and then perhaps use the joiner to join the connector space object to the chosen MV manually.

How Join Rules Work

When a CS object is joined to an MV object, it establishes a link that potentially enables subsequent attribute flow to occur between the two objects. Until this join occurs for a CS object, it can have no effect on the MV because no relationship to the MV exists.

In the course of normal import runs, newly imported CS objects that do not have links to the MV will be automatically joined to existing MV objects if the join process is successful. In cases where no join can be found, a new MV object may be projected (if enabled by the metaverse projection rules) and the CS object is automatically connected to it.

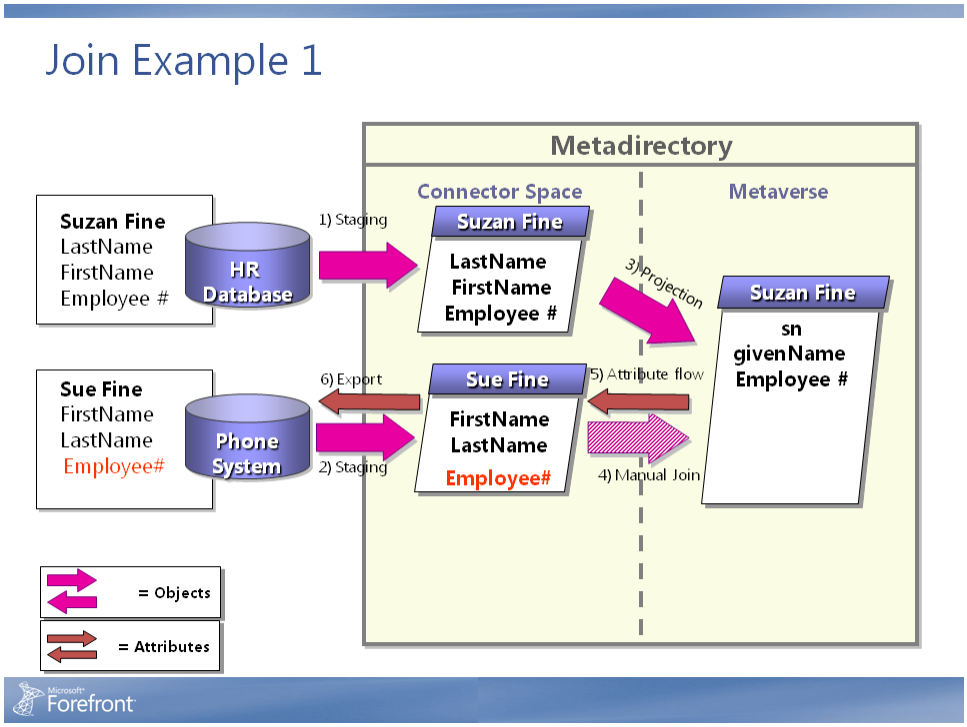
A Join resolution rule can be used to decide whether the MV candidate is suitable for joining, or which one out of a number of candidates is suitable.

During synchronization, filtered disconnectors and explicit disconnectors (those created with the Joiner tool or deprovisioning rules) are not evaluated for joining and will remain as disconnectors.

Joining and Indexes

During joining, each object in the CS is considered in turn, and the join rules are evaluated against all MV objects to try to find a join. Because the MV data is represented by values in a normal SQL Server table (unlike the CS, where the holograms are binary objects), it is possible to index the MV fields. In fact, it is a best practice to ensure that all attributes involved in join rules are indexed in the MV. Attributes can be marked for indexing using the metaverse designer tool.

Join Example 1



Suppose you have two data sources, an HR database, and a telephone system, and that, in the past, the telephone system data has not been handled rigorously.

In the MV you have (among other attributes) displayName (Suzan Fine), givenName (Suzan), and sn (Fine). In the telephone CS object you have displayName (unluckily Sue Fine), FirstName (Sue), and LastName (Fine).

You may be able to join the existing data on displayName, but this is likely to be imperfect.

You may be able to join on surname, but that may be even less reliable (you are likely to find more than one match).

Your join logic might now look like this:

Join Rule #1

```
CSObject:displayName = MVObject:displayName
```

This may work, or it may not.

Join Rule #2

```
CSObject:LastName = MVObject:sn
```

Note that the rules are in order of increasing permissiveness.

This may work, and between the two, for each case they produce lots of results, or just one (or none). You could use a join resolution rule to try to decide which of the matches to use (but this is beyond the

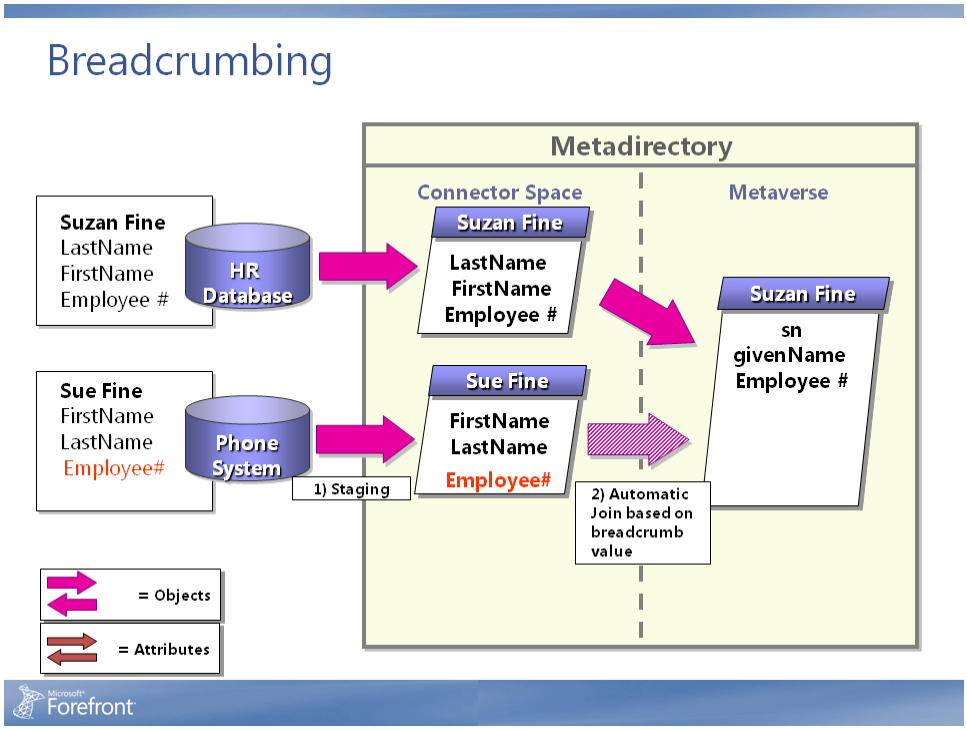
scope of this course, being a rules extension). Even if it resolves to just one, you can't be sure it is correct.

> **Note:** A false positive may be more troublesome than a negative; at least with a negative you know you have a problem, but a false positive might go undetected for a long time.

Unless it can be resolved to just one reliable one, you will have to resort to a manual join at this point.

You could add rules extension join rules, which do more esoteric matches (for example, attempting to match values constructed from bits and pieces of FirstName, LastName, and displayName with various MV attributes), but, again, this is beyond the scope of this course.

Breadcrumbing



Assuming that you have joined manually, and that you anticipate a lot of this (which is very time-consuming), you'd really rather make sure that you never have to do it again.

So once the join has been made, you will flow a unique attribute out to the data sources—this is called breadcrumbing. Of course this assumes that you can modify the external system (there are alternative methods involving third-party systems if you can't).

So now your rules line up like this:

Join Rule #1

```
CSObject:EmployeeID = MVObject:EmployeeID
```

This will work if this object has previously been manually joined and breadcrumbed (or the breadcrumb value has been added by some external process).

Join Rule #2

```
CSObject:displayName = MVObject:displayName
```

This will be the first join that could work when you tackle an existing phone system object.

Join Rule #3

```
CSObject:LastName = MVObject:sn
```

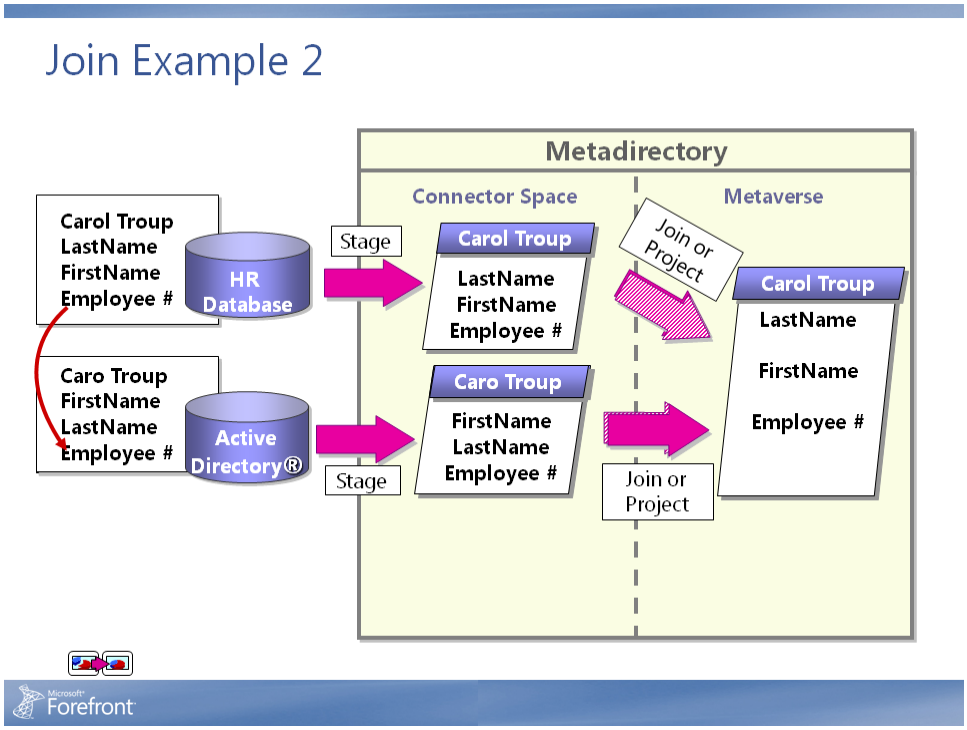
This is the most permissive and least reliable.

In production, your aim is going to be that in future the problem is not repeated, so hopefully you can eventually dispense with rules 2 and 3.

Delete and Reload?

Manual joins result in explicit connectors, and these do behave a little differently from normal connectors in that connector filters do not apply to them. This may not seem like a problem, but it is an inconsistency that could trip up a different consultant working on the system in future. There is little harm done in deleting the CS and reloading the data, allowing it to use the breadcrumb values to join normally.

Join Example 2



One of the first tasks when implementing FIM is to establish existing processes, see how FIM can contribute to these, and if necessary, define improved processes.

Suppose you have two data sources, an HR database, and AD. You might interview the HR director and obtain the following input:

"When a new employee arrives, we enter their details into the HR system and provide the IT people with those details so that they can create an AD account. So we'd like FIM to import our HR data and create those accounts automatically."

However, it would not be unusual to find that this procedure is less than rigorous. You might get this input from the IT department:

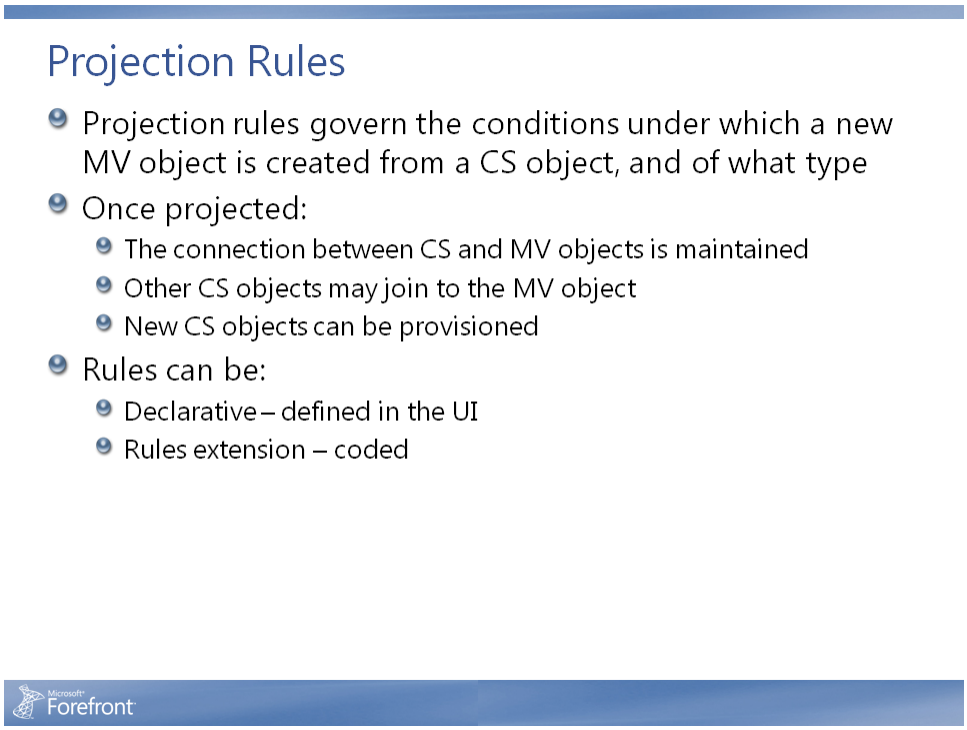
"Ah yes, we know that's what is supposed to happen, but very often the HR department is overwhelmed with work, and the new staff desperately need access to IT systems. So we create an account, and sort it all out later."

In this case, you could do the following (this is just one suggestion, for this example scenario):

- Instigate a system whereby, without delay, a new employee is issued with an employee number, even if there is no time to enter them into the system.
- Now an account can be created in the HR or IT system, using this employee number.

- FIM imports new accounts (from either system), and you use the following join/projection logic in both systems:
    - Try to join using employee number.
    - Project a new MV object (the projection only takes place if the join fails—that is, if there is no existing object).

Projection Rules



Projection rules govern the conditions under which a new MV object is created from a CS object. Projection rules are responsible for determining if projection into the MV should occur and the appropriate object type to employ.

Projection rules differ from join rules in that during a join process the MV is searched for existing objects; during the projection process, projection rules determine whether or not a new object is created in the MV so that the CS object can link to it. MAs apply projection rules to objects where a join has failed or join rules were not configured.

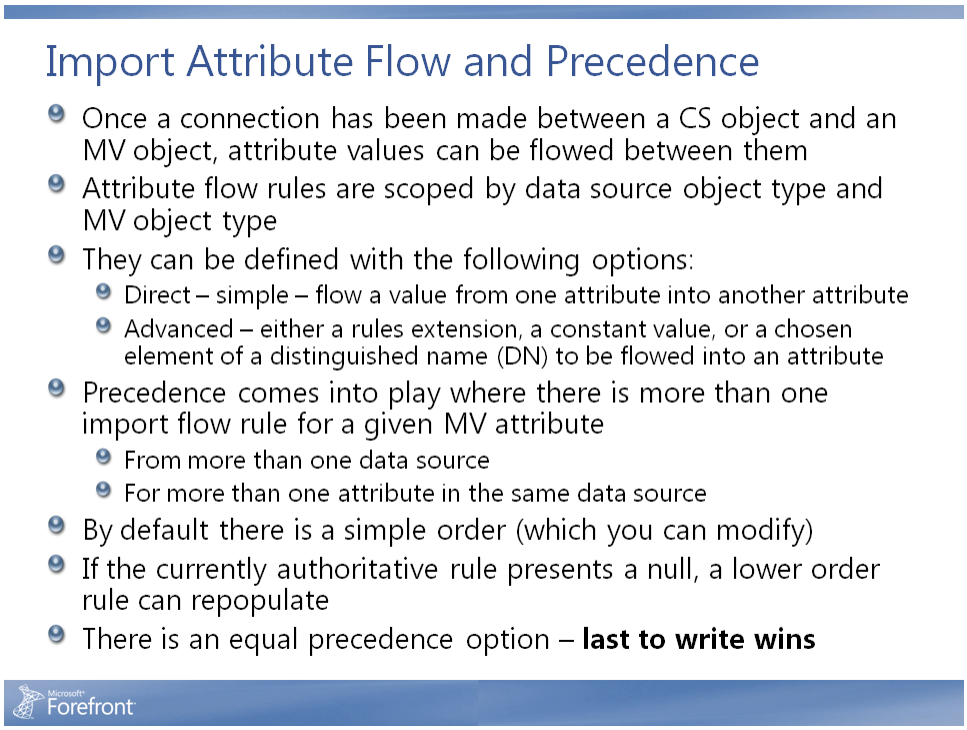
> **Note:** At least one of your MAs must have a projection rule, or you'll never get anything in the metaverse. Projection rules only create a new object—attributes are not populated with values until you configure attribute flow rules.

Declarative Rule or Rules Extension

You can define only one projection rule for each object type in each MA. The projection rule can be a:

- **Declarative rule** – Defines which MV object type should be mapped for a chosen data source object type.
- **Rules extension** – Enables users to write a rules extension that can examine the source object and determine whether it is to be projected to the MV, and, if so, as what object type—but this is beyond the scope.

Import Attribute Flow and Precedence



Import Attribute Flow

A connection can be made between a CS object and an MV object via:

- Projection
- Join rules
- Manual joining

Once this has happened, attribute values can be flowed between them.

Attribute flow rules are scoped by data source object type and MV object type, and can be defined with the following options:

- **Direct** – simple; flow a value from one attribute into another attribute
- **Advanced** – either a rules extension, a constant value, or a chosen element of a DN to be flowed into an attribute

You can have multiple import flow rules to a given destination attribute from different source attributes. These will be ordered by attribute precedence.

Precedence

It is a common experience for most organizations that, for some attributes, there is no single authoritative source for the information so sometimes the value will be provided by (say) the HR system

while at other times by (say) a directory service. Clearly only one of them can win. Which one wins is (by default) determined by the precedence order configured in the metaverse designer.

The sources of the data for flow rules might be different attributes in different data sources (such as someone's surname held in different systems, where you deem one to be more likely to be right), or they could be different attributes in the same system (such as home telephone or mobile phone numbers from the HR system where you want the mobile number if present, and the home number if there isn't a mobile number).

Precedence is scoped by object type. So if, for example, you have a title attribute from a particular data source flowing into the MV title attribute, it could be that it is precedent when flowing into a person object, but not for a contact object.

Now, suppose there are three data sources configured with Import Attribute Flows to the MV title attribute of a particular object type—person:

Data Source Name	Data source Attribute	MV Attribute	Precedence
HR	JobTitle	title	1
AD	title	title	2
Lotus Notes	JobTitle	title	3

If you change a value for JobTitle for a user in the HR system and perform an import and synchronization, FIM will flow the value in because it is number 1 in the precedence order.

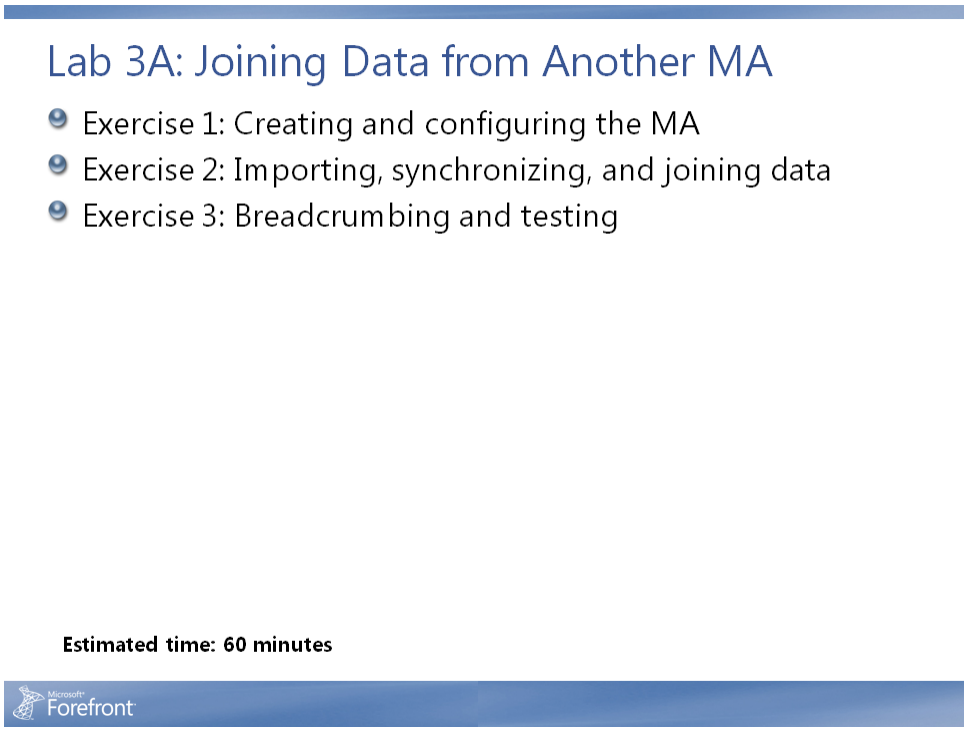
If a change is made instead in AD and imported and synchronized, FIM will determine that it is second in the precedence order and will only flow a value into the MV if the MV attribute value is null (because the HR system has not provided one), or it has been populated by a lower precedence rule (for example, Lotus Notes).

Any value in the Lotus Notes JobTitle attribute can only flow if both the HR system and AD have null values.

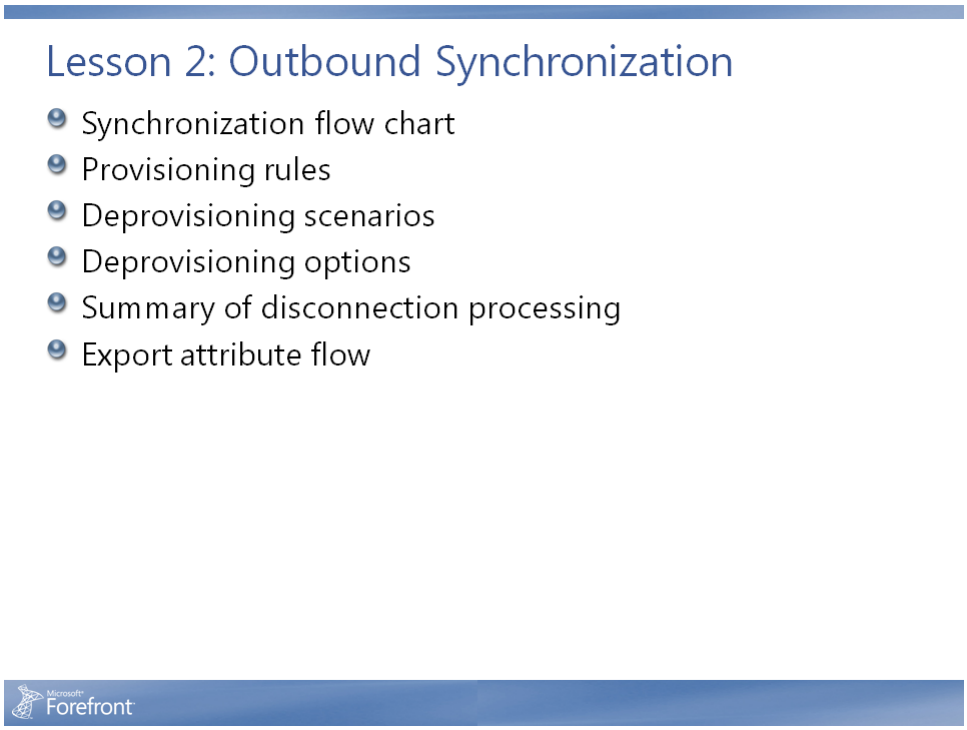
If the MV value subsequently becomes null, the next rule in the precedence order is invited to provide a value—this is called **Repopulation**.

> **Note:** It is possible to configure an attribute for equal precedence in which case last to write wins. Keep in mind that the results can be unpredictable, because the order of events matters, so during a reload of data (perhaps on disaster recovery), the result could be different.

Lab 3A: Joining Data from Another MA

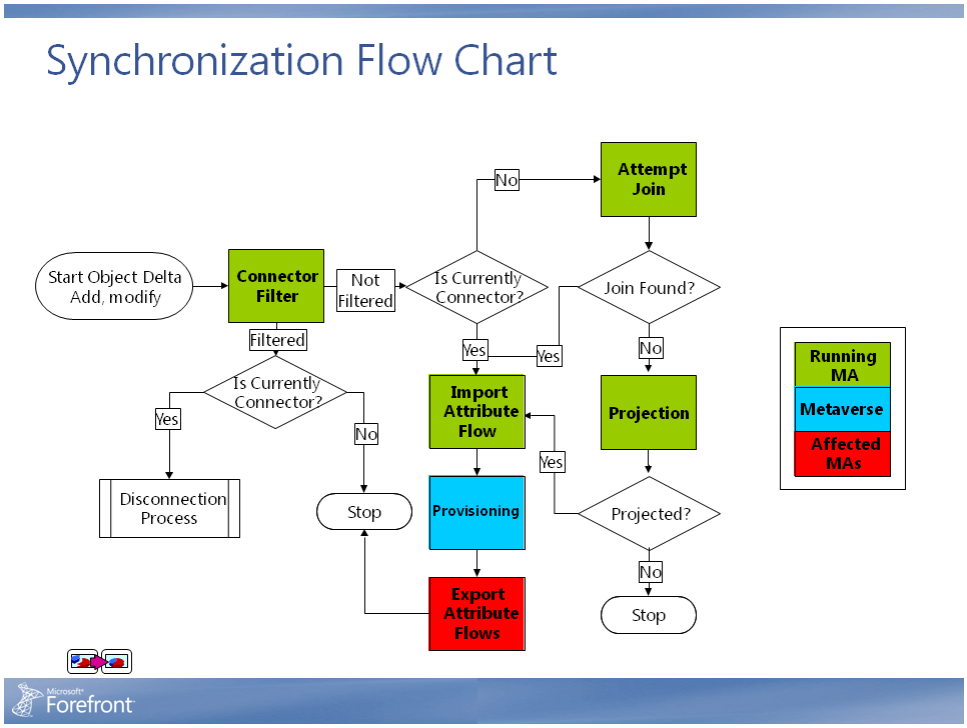


Lesson 2: Outbound Synchronization



In this lesson you will examine outbound synchronization.

Synchronization Flow Chart



The synchronization engine processes each connector space object separately, and applies a series of rules to establish what to do with it. The diagram on this page shows the way rules are applied to objects that have a pending add or modify (deletes are a different matter). Note that each of the blocks in the diagram actually consists of rules (or groups of rules) that can be either declarative (created in the user interface) or extended rules (within a rules extension), except that provisioning rules are always rules extensions.

Synchronization Flow

Whenever a CS object is created or modified, the MA's connector filter is evaluated. If this results in a change of connection state, either the disconnection rules are evaluated (see later) or the join and projection rules are evaluated. If a successful join or projection is achieved, any import attribute flows take place.

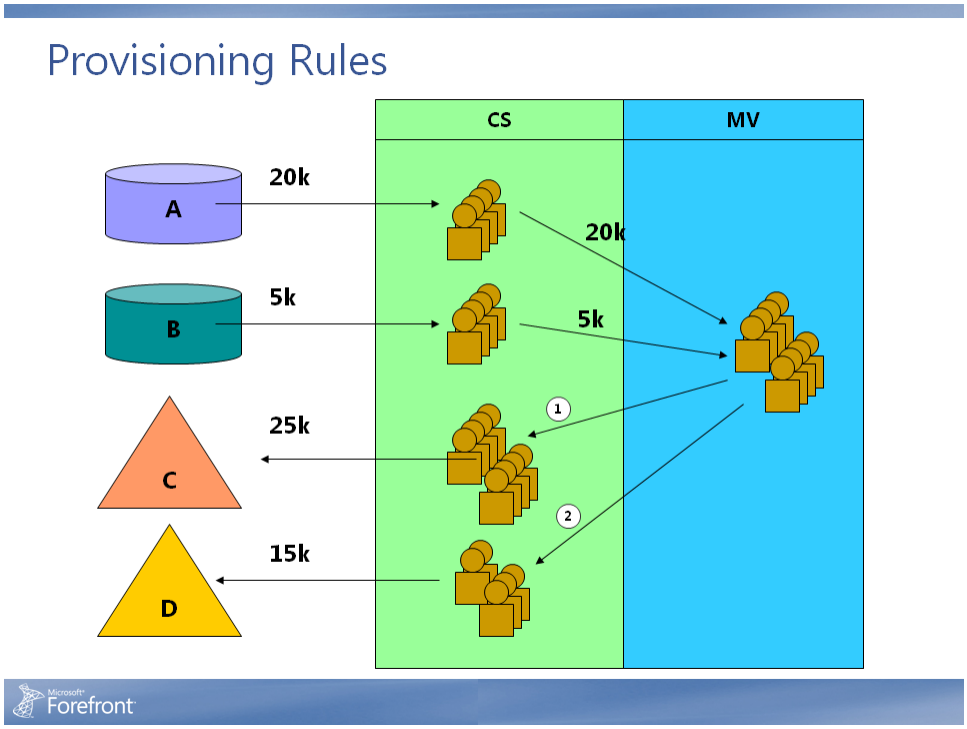
Because this modifies (or at least touches) an MV object, the MV provisioning code is executed—more later.

Export Attribute Flows

Any MAs with export flows from MV attributes that have been modified now execute those export flows, and their CS objects are updated with appropriate values. Note that these MAs are not explicitly run (that is, via a run profile being executed); this is all the result of your original MA importing changes to a single object. These export flows will only reach their ultimate destinations in the connected directories when the relevant MAs are executed with an export run profile.

Note: Renaming or moving an object (from one directory container to another, for example) can only be done in your provisioning or deprovisioning code by setting the anchor attribute or DN. Modifying an object's other attributes can be done as part of export attribute flow.

Provisioning Rules



There are two fundamental methods of controlling synchronization: using classic rules (which you are doing now) and using declarative synchronization rules defined in the portal, which will be covered later.

Provisioning using classic rules can only be done in code. The scope of this course cannot extend to learning code, and so when it comes to the lab, you will simply be given a .DLL file to be treated as a black box.

In spite of not knowing what is in the black box, it is vital that you understand how to use it. Most of what is covered in this topic applies equally to the declarative synchronization rules defined in the portal, which will be covered later.

How to Enable Provisioning Code

In **Options** on the **Tools** menu, you can click **Enable metaverse rules extension**, and then browse for a .DLL file (which must be in the extensions folder).

This DLL has two purposes—it can simply provide an MV deletion rule, in which case you do nothing else, but it can provide provisioning rules, in which case you must additionally click **Enable Metaverse Rules Extension**.

The Purpose of Provisioning Code

Provisioning code (and for that matter the declarative Synchronization Rules you see later) decides when and whether to create new CS objects ready to export to data sources, but it can be used for other purposes (for example, resetting DNs and deprovisioning). Because it has many uses, FIM arranges

things so that the provisioning rules run often—in fact, they run every time an MV object is touched (involved in) the synchronization process (whether or not anything actually changes).

The Need to Run Full Synchronizations

Generally, when provisioning rules are changed (which may be rare in a production system), the change must be applied to all existing objects. As already described, this is why you have the full synchronization run profile. The trick is to do it with as much economy as possible (because full synchronizations are inherently time-consuming).

There is no simple way of describing a rule for this; you may be able to rely on running full synchronizations for just those MAs that have a projection rule, or you may be able to identify one MA that manages a connector for every single object (and so it would be enough to run just this one). You simply have to know your data!

Perhaps the easiest way to explain these ideas is through an example:

Suppose you have four management agents: A, B, C, and D.

Management agent A has projected 20,000 objects. Management agent B has projected 5,000 objects. Now the MV has 25,000 objects.

1. You introduce some MV code designed to provision an object in C for each MV object. To make this work, you will need to run full synchronizations for MAs A and B. (Note that there are still 25,000 MV objects, and that there are now 50,000 CS objects.)
2. Now suppose you change the provisioning code so that it also provisions accounts in D, and let's say that this is only for some of the MV objects. To make this happen, you need the provisioning code to run for each of the MV objects, so you need a set of full synchronizations that will make sure each of them is touched at least once and preferably only once. (Note that there will still be 25,000 MV objects and 65,000 CS objects.)

Which MAs need a full synchronization run? Have a think about this for a moment, and then read on.

You could, of course, perform full synchronization runs for A, B, and C—safe, but slow. You could just perform full synchronization runs for A and B. Note, though, that in this case, just C alone would do it.

However, if, in the meantime, any A and B objects had become disconnected from the MV, leaving their MV objects connected to C, the A and B answer would no longer work.

And if, in the meantime, any C objects became disconnected from the MV, leaving their MV objects connected to A or B, the C alone answer would no longer work.

As you see, there is an easy answer (all of them)—but it is not the best answer!

Deprovisioning Scenarios

Deprovisioning Scenarios

- Deprovisioning is (strictly speaking) account deletion
- It is not just what happens at the end of an identity life-cycle (though that is very interesting)
- Deprovisioning – and reprovisioning – may take place many times in the lifecycle of an identity
- Can be initiated by:
  - Actual deletion
  - Filtering actions
  - Provisioning rules (classic rules extensions, or declarative synchronization rules)

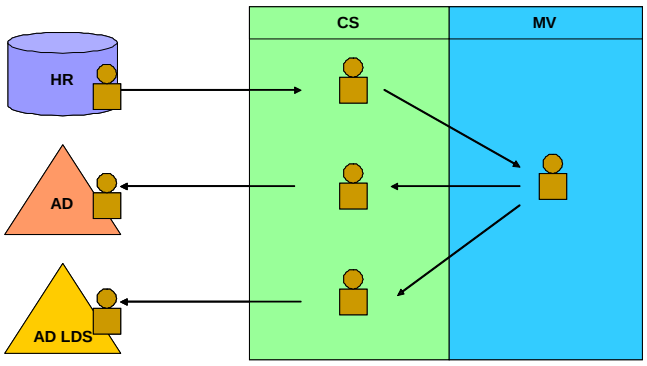
Microsoft Forefront

Provisioning is quite plainly the creation of new identity objects (accounts, records)—usually at the start of an identity's life-cycle. Deprovisioning is the opposite—not always at the end of an identity's life-cycle (when you might not choose to delete objects, but move or disable them). During the lifetime of an identity, events such as roles changes, sabbaticals, suspensions, and so on may result in many deprovisioning and re-provisioning actions.

Deprovisioning actions can be initiated in a number of ways. This topic presents some example scenarios. These scenarios are presented as part of an analysis of classic synchronization rules, but they apply more or less equally to appropriately configured synchronization rules created in the portal (see Module 5).

Simple Deprovisioning Scenarios

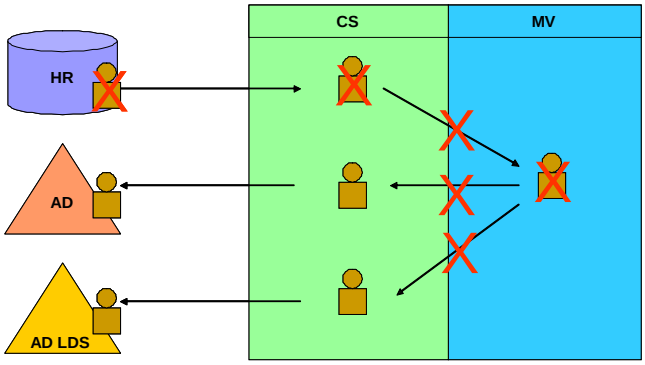
The start point for each of these is the same:



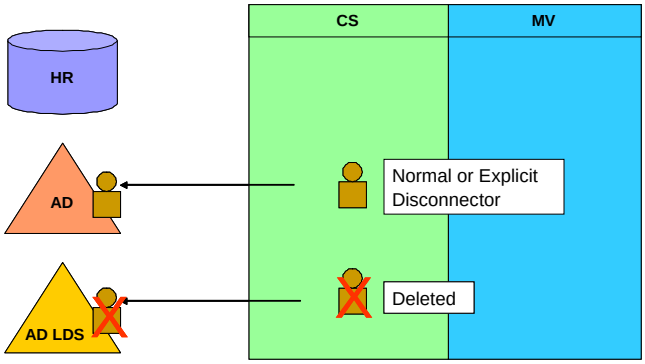
The HR system has projected an MV object. The provisioning code has created two new accounts for each HR object, one in AD and the other in an instance of AD LDS (though these details are not very relevant). The HR system, then, is authoritative for object creation in these examples, and it is also authoritative for object deletion. The end point is a little different in each case. The primary difference between the scenarios is the way deprovisioning is initiated.

Actual Deletion

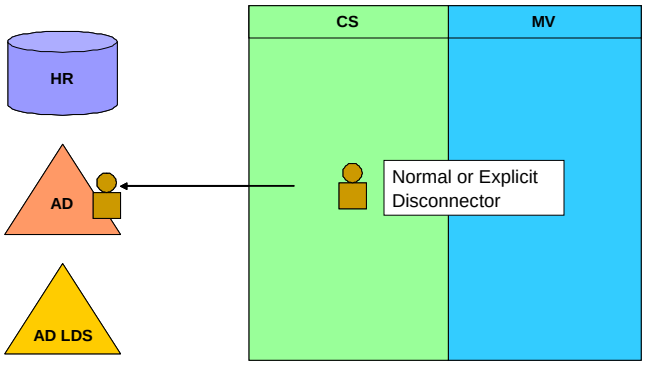
Suppose the HR object gets deleted, resulting in disconnection from the MV object. This causes the MV object deletion rule to be applied (see next topic), and your rule is set to cause deletion upon disconnection by the HR MA:



You now have two orphans that will be dealt with according to the deprovisioning rule set for each MA. The AD LDS MA is configured to stage a deletion of these objects (which will export an actual deletion during the next export run). The AD MA is set to leave an explicit disconnector (the third choice is a normal disconnector, which you might use if you intended that it connect up again at some future point):



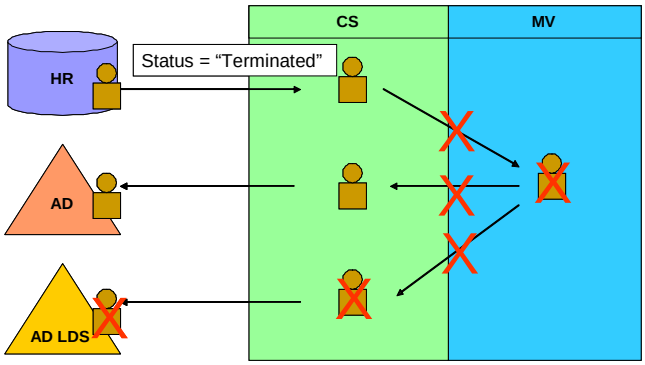
The end result is as follows:



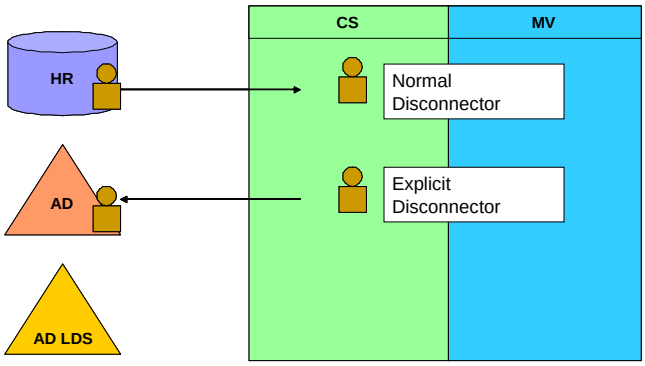
This example works for an actual deletion in the HR system, which you don't want to result in a deletion in AD. There is no record of the object in the MV.

Filtering Actions

In this case, suppose that an attribute in the HR system has changed as it's imported (for example, status = "Terminated"). You could have a connection filter configured to disconnect in these circumstances. Things then proceed very much as before:



However, the final state is a little different. The HR record has not been deleted, and you keep a corresponding normal disconnector:



Should the status flip back, projection and provisioning would start all over again. Actually it would not be successful, as the AD object still exists so there would be an "object already exists" type of exception. This is a whole new discussion, which is really the province of a more advanced course.

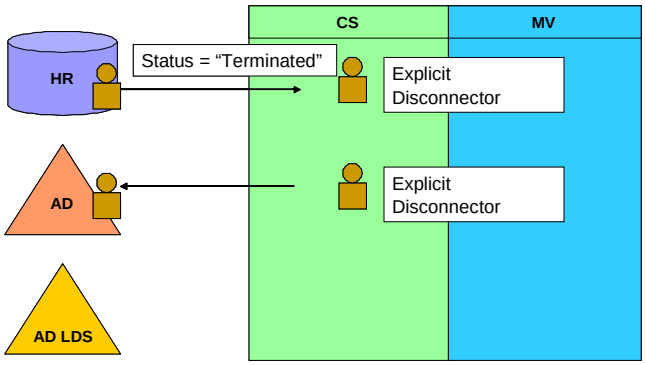
This works where the HR record is never deleted, but you do not require a record of the object in the metaverse, and you require the AD LDS object to be deleted. Actually, the MV object deletion rule could be changed not to delete it, and attribute flow rules arranged to make sure that the AD and AD LDS objects are disabled.

Example with Provisioning Code

Where provisioning code is used for deprovisioning actions, a range of possibilities opens up. In this final case, suppose that an attribute in the HR system has changed and is imported (for example, status = "Terminated"), but that there is no filter. The provisioning code runs (as it does during every synchronization of an object), detects the attribute value, and then does one of the following things.

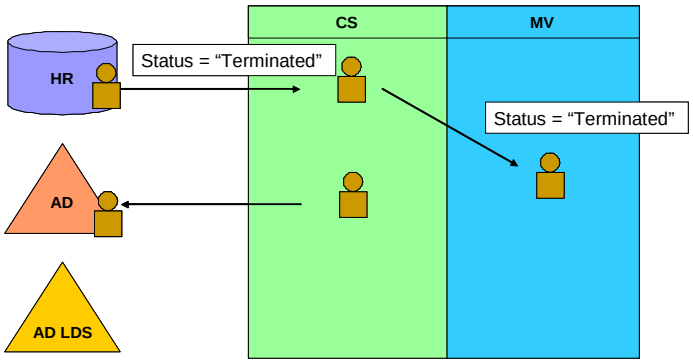
Disconnect all connector space objects

This leads to three orphans and deleting the MV object itself. The AD and AD LDS objects are treated as before, and the HR MA could have been configured to leave the HR CS object as an explicit disconnector (in the previous example, the HR object is a normal disconnector which rather unnecessarily goes on being synchronized, the filter leaving it as a disconnector again):



Disconnect only the downstream AD and AD LDS objects

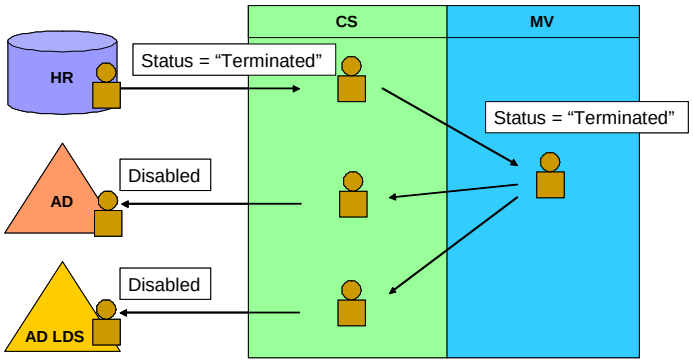
This results in a deleted AD LDS object and an explicit disconnector AD object as before:



The difference here is that you keep a record of the identity in the MV, which could be desirable for report and audit purposes.

No disconnection at all

Leave everything connected up, but make sure that accounts are disabled and perhaps moved to special containers:



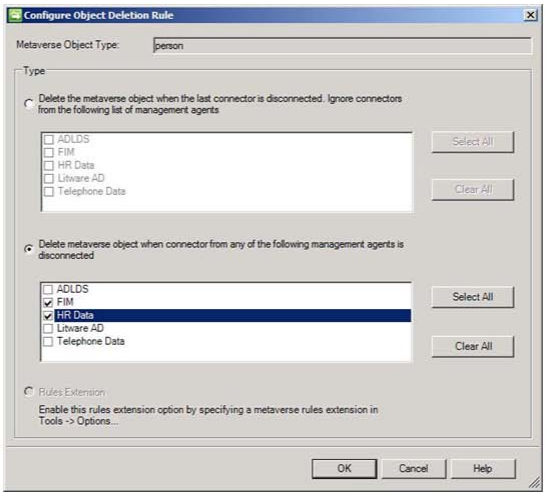
This gives you a lot of control, and of course keeps everything in place. Re-enabling the identity would be easy in this case, and of course you have a complete record of the identity.

These are not the only possible configurations. Hopefully these examples have given you some insight into this subject.

Metaverse Object Deletion

Metaverse Object Deletion

- The Metaverse Designer specifies when an MV object is deleted



Microsoft Forefront

The last topic raised the question of MV object deletion, and that is reconsidered here.

A detailed explanation is presented here, but might be best kept for future reference.

At first glance the configure object deletion rule dialog can either look confusing or deceptively simple—both of which can be a problem! In simple cases it is easy enough, and it offers quite a bit of flexibility before the need for a rules extension (when you can be as flexible as you like), but there are scenarios that add complexity.

First, an MV deletion is triggered by the disconnection of a CS object. Whenever a CS object disconnects the question arises: Should the associated MV object be deleted? This rule then decides the fate of that MV object. How did the disconnection arise? Often it will be that the source object has been deleted, but it could be a filtering action, or it could be a programmatic, or even a manual action—it doesn't matter.

Next, something simple: There is a default behavior in FIM that an MV object will be deleted when its last connector disconnects. Whichever of these options you take, you cannot switch off this default behavior, which ensures that you do not end up with an orphaned MV object.

A nasty complication is that it is possible to have more than one CS object from a single MA connected to the same MV object. This is not an ideal situation, but it is not unheard of and sometimes necessary. The situation does not arise in the training course, so in the examples below you will ignore the possibility.

The top option on the dialog allows you to modify the default behavior by excluding some (presumably) unimportant MAs from consideration. In the Module 1 scenario you might consider that if the last object

associated with either of your two authoritative sources disconnects, this should give rise to MV deletion (and hence deletion in the three non-authoritative sources—assuming that the deprovisioning configuration for the three non-authoritative sources lead to deletion). In that case you would configure:

Delete the metaverse object when the last connector is disconnected. Ignore connections from the following list of management agents:

- ✓ ADLDS
- FIM
- HR Data
- ✓ Adatum AD
- ✓ Telephone Data

If Adatum AD disconnects, nothing happens (unless it is the very last one).

If FIM disconnects but HR Data is still connected, nothing happens.

If HR Data then disconnects, MV object is deleted.

If it were possible to have a case where there never was a FIM or HR data connector, the default behavior applies (disconnection of last connector triggers deletion).

The middle option allows authoritative sources to directly control deletion independently. In the Module 1 scenario you might consider that if any disconnection takes place from either of your two authoritative sources, this should give rise to MV deletion (and hence deletion in the three non-authoritative sources). In that case you would configure:

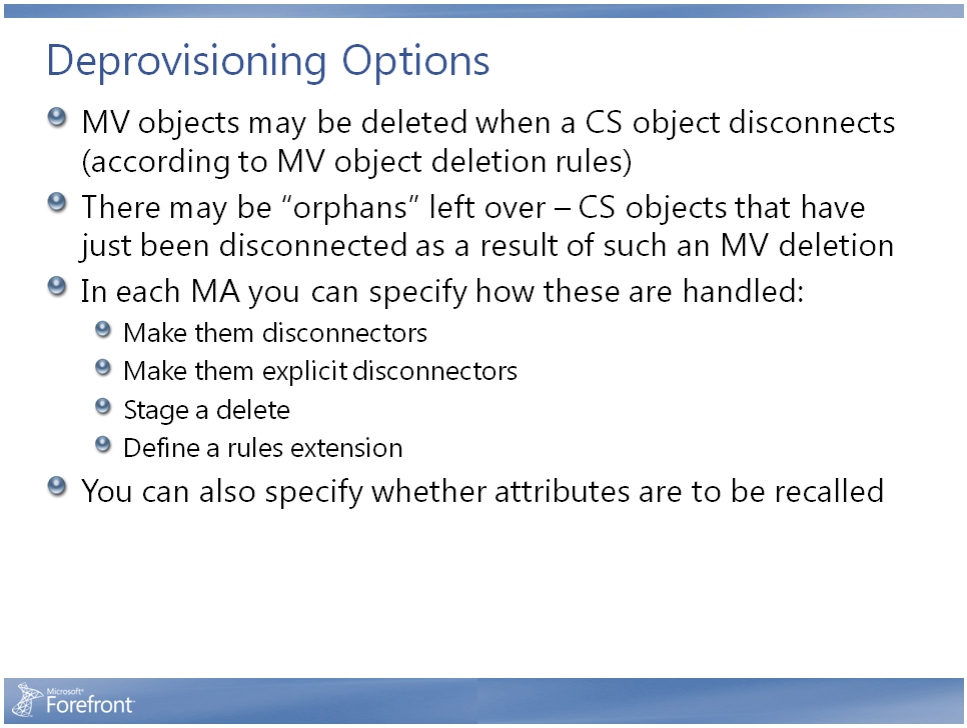
Delete the metaverse object when the connector from any of the following management agents is disconnected:

- ADLDS
- ✓ FIM
- ✓ HR Data
- Adatum AD
- Telephone Data

If it were possible to have a case where there never was a FIM or HR data connector, the default behavior applies (disconnection of last connector triggers deletion).

The last option allows you to write code to deal with complex requirements. In the training scenario you might consider that if a contractor is deleted from the FIM portal, or a full-time employee is deleted from the HR application, this should give rise to MV deletion (and hence deletion or disconnection in all other sources depending on their configuration), but NOT if a full-time employee is deleted from the portal. In that case you could configure a rules extension, in which your code could examine what just disconnected and whether the MV object concerned represented a contractor or a full-time employee. If the very last connector disconnects, the default behavior applies, and your code is not even called.

Deprovisioning Options



When an MV object is deleted, or because of other provisioning rule actions, CS objects are turned into orphans, CS objects suddenly aren't connected to their MV object counterpart. The Deprovisioning rule defined in an MA decides what to do with these downstream disconnections.

Make them Disconnectors

If they become disconnectors, then every time a synchronization run of the MA is performed, they are run against the filter, join, and projection rules, perhaps resulting in it joining to an MV object again.

Make them Explicit Disconnectors

If they become Explicit Disconnectors, then they are not run against the filter, join, and projection rules, when a synchronization run of the MA is performed, and thus will never rejoin to a new MV object, even if a new match becomes available.

Stage a Delete on the Object for the Next Export Run

You can put the CS object into a pending delete state; when the next export run is performed, the corresponding data source object will be deleted.

The deprovisioning rule is not implemented in the following situations:

- The object is manually disconnected (for example, in the Joiner tool).
- The object is disconnected because it now satisfies a connector filter.
- The object is imported as a delete in a delta import or is assumed to be a delete because it has gone missing during a full import.

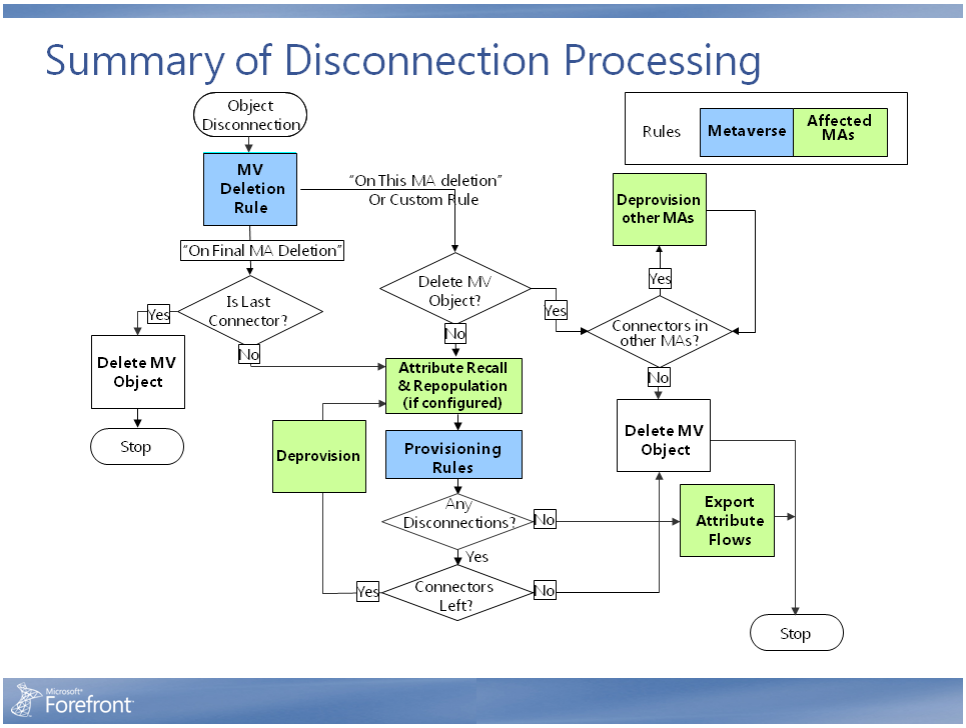
Do Not Recall Attributes

Normally, when an object is disconnected, any attribute values it contributed to the MV object will be withdrawn unless the option **Do not recall attributes contributed by objects from this management agent when disconnected** is selected.

If this option is selected, these values will remain unless there is a flow rule from another MA that can contribute these values. Where there are multiple flow rules, they will be invited to provide a value in order of precedence.

This option can apply to upstream or downstream disconnections.

Summary of Disconnection Processing



Note: Deprovisioning is a complex topic, and you may feel that this diagram and topic can be left for the moment and kept for future reference.

When an MA disconnects a CS object from the MV (for example, because the object is to be deleted, or because it matches the MA's connector filter), the disconnection process is followed.

MV Object Deletion

The MV deletion rule is evaluated and a decision is made whether the MV object should be deleted. If so, all other MAs must evaluate their deprovisioning rules and decide what to do with their CS objects that were joined to this MV object.

If the MV object is not to be deleted, the MV provisioning rules are executed. They could conceivably disconnect selected connectors; if this happens, the appropriate MA's Deprovisioning rules are evaluated.

By default, when an object in an MA's CS is disconnected from the MV, the attributes provided by that object are deleted from the MV (and in each case the next MA down the precedence list is invited to populate it, and so on down the list). This behavior is controlled by a switch on the deprovisioning page of the MA configuration.

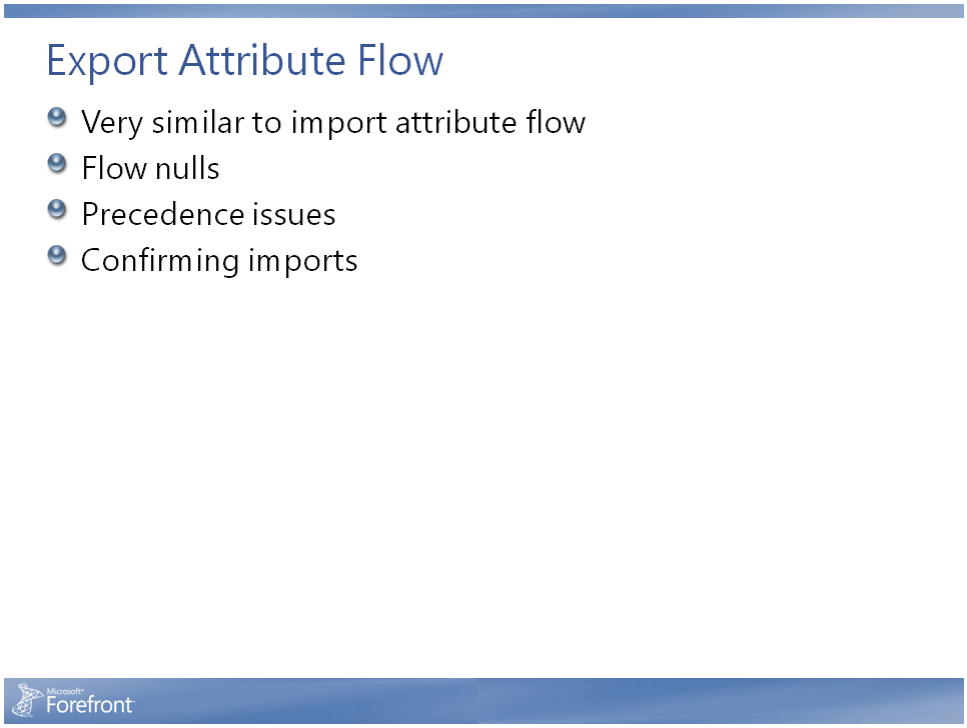
Provisioning Retry

If a further disconnection is made by a provisioning rule, the MV object is marked for provisioning retry, and the object will be passed through the provisioning rules again. This allows a developer to replace an object with an object of a different type (for example, a contact with a user) by removing the first object

and provisioning its replacement (which can have the same anchor) in the same synchronization run. This provides a slight exception to the rule about delta synchronizations; a delta sync will process objects that have modifications, and also any MV objects marked for provisioning retry.

Note that if you delete the CS of an MA, any MV objects that were connected to CS objects that have been deleted are marked for provisioning retry in this way, and so they will be processed during the next synchronization run of any sort.

Export Attribute Flow



Export attribute flows are like import attribute flows, with a few additional points.

Null Values

The only significant difference between export and import flows is that on export, you can control how the MA handles null values. If an MV attribute contains a null value, you can choose whether that null value should be exported to the connected directory. If it is exported, it will overwrite any existing data in the data source, which may be undesirable (anything sometimes being better than nothing). Selecting allow nulls will cause this behavior, while deselecting this option causes the MA to ignore null values, and they will not be exported to the data source.

Precedence Issues

If the value of an attribute in the MV comes from an MA with a lower precedence than this MA (with the export flow), the export will not be processed. No statistics are displayed to indicate this (you can use preview to examine the synchronization process in detail; if precedence is preventing export, you will see the message **Skipped – Not Precedent** in preview).

If you have an import attribute flow from a source, and you wish also to have an export attribute flow (unusual, but not unheard of), then even if that import is the only one for the MV attribute concerned, the export flow will not flow against it unless you select equal precedence for that attribute (in the metaverse designer).

Export Data Flow

Attributes subject to export attribute flows will be modified in the CS of the relevant MA during a synchronization run. The data will only be delivered to the data sources when an export run step is executed for that MA.

Confirming Imports

In general, when FIM exports data, it can't always be sure it got there safely. A call-based MA will receive some sort of acknowledgement from the target system that the data was received, but this is not sufficient to assume that all the attribute changes were accepted and understood. Therefore, for exports by a call-based MA, CS objects switch to an awaiting export confirmation state (internally known as an unconfirmed export).

Exporting file-based MAs do not get any direct information that the data source has received the data, let alone that it has been successfully assimilated by the connected directory—they only know whether they have successfully written a text file, which may be lost before the data source can read it. It is important, therefore, that file-based MAs can completely repeat any exports, so objects exported by a file-based MA enter the export in progress state (internally this is known as an escrowed export).

Once the system receives a successful import of the data that has been exported, the CS objects return to their normal states. It is common to execute an import step (usually a delta) and a delta synchronization as soon as possible after the export step, in order to confirm that the same data was imported as was exported. If an attribute is re-imported with a value that contradicts what was exported, an **exported-value-not-reimported** error occurs.

Lab 3B: Provisioning AD LDS