Implementing Forefront Identity Manager 2010

Student Manual

Module 5: Managing Synchronization from the Portal

Table of Contents

Module 5: Managing Synchronization from the Portal

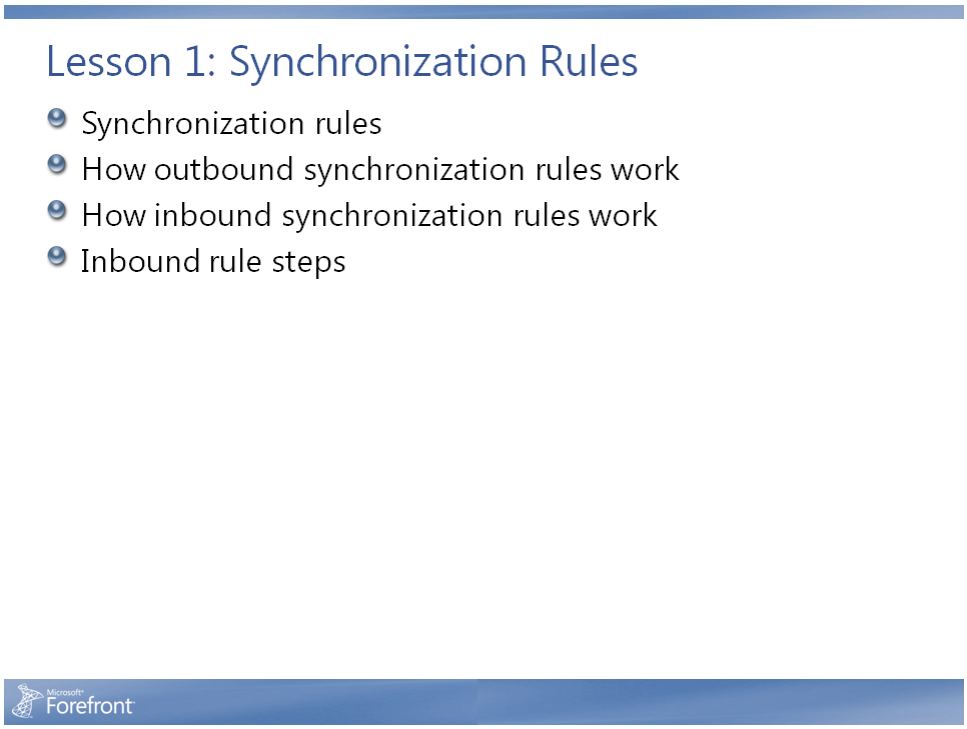
Module Overview

Until now, you have performed synchronization by using the Synchronization Service directly, using classic rules configured in the Synchronization Service Manager. This module is all about how synchronization rules can be configured in the portal, and then acted upon by the Synchronization Service.

We need to understand how to configure synchronization rules, how the Synchronization Service makes use of these, and how synchronization rules map to the classic rules you already know about.

Most of the module concentrates on the general issues to do with synchronization rules, but Active Directory® (AD) is a great example to use, and most organizations using Microsoft® Forefront™ Identity Manager 2010 (FIM) will also be using AD, so the last lesson of the last lab focuses on managing users in AD.

Lesson 1: Synchronization Rules



All the synchronization you learned about in Modules 2 and 3 we will call **classic rules** (because they were available in FIM's predecessor, ILM) and can be simple rules specified in the Management Agent (MA) configuration, or rules extension rules like the Active Directory® Lightweight Directory Services (AD LDS) provisioning DLL you used in the lab.

We now move on to declarative **synchronization rules**, which provide a great opportunity to increase both the transparency of the system and the productivity of those responsible for configuring the system. This lesson covers the use of declarative synchronization rules and contrasts them with classic rules.

Introducing Synchronization Rules

Introducing Synchronization Rules

- Synchronization rules are imported by the Microsoft® Forefront™ Identity Manager 2010 (FIM) management agent (MA), and projected into the metaverse (MV), where they are executed
- No MV extension is required for provisioning
  - It is still available and can be switched on and off in **Tools | Options**
  - Declarative provisioning can also be switched on and off
- Synchronization of FIM MA is a special case – the FIM Service and FIM Synchronization Service databases can almost be seen as one
  - Projection, joining and provisioning are all automatic
  - The FIM MA only uses classic attribute flow

Microsoft Forefront

Introducing Synchronization Rules (Cont.)

- Synchronization rules include definitions of:
  - Scope: To which objects should the rule apply?
  - Relationship: Is there an existing object, and should a new one be created? What if the rule is removed?
  - Attribute flows: more flexible than direct classic rules, including functions, for example, First Name + Left (LastName,1)

Microsoft Forefront

Synchronization Rules are Imported to the Metaverse

So far you have seen how the Synchronization Service can be configured using classic rules: MA configurations, plus some metaverse (MV) configuration, including provisioning which could only be done using code in the metaverse Extension DLL (like the AD LDS provisioning DLL you used in the lab).

We now move on to declarative **synchronization rules**, which are created in the portal and imported into the MV using the FIM MA in exactly the same way that persons are. You configure the synchronization rules to include decisions about joining, projection, provisioning, and deprovisioning (although these words are not used), and with attribute flows. Once in the MV, the Synchronization Service can interpret them and apply them just like classic rules.

There are two fundamental types of synchronization rule: Inbound, and Outbound. There is a third type called Inbound and Outbound, but this should be seen as simply two independent rules pushed together because it makes the interface more convenient.

No Metaverse Extension Configuration Required for Provisioning

As you have seen, when using classic rules for provisioning you create an MV extension and write code. In the declarative model, provisioning is simply part of outbound synchronization and is configured without code. The synchronization rules you create are imported as objects in the MV where they are interpreted by the Synchronization Service. It is not necessary to configure an MV extension in Synchronization Service manager.

However, classic provisioning is still available, and if a rules extension is configured, it is executed after any declarative outbound synchronization rules, and classic rules win.

Disabling/Enabling Provisioning

In the **Tools** | **Options** dialog box, there are switches for both classic (metaverse rules extension) and declarative (synchronization rules) provisioning. There are times when it is useful to disable provisioning—the notable example being during a discovery/initial load (including a disaster recovery reload) process. In such circumstances, you do not want to be provisioning *missing* objects until all existing objects have been imported (because these *missing* objects may not be missing, merely not yet imported), and so you disable provisioning.

FIM Portal Synchronization is a Special Case

In a sense, from the standpoint of the Synchronization Service, the FIM Service database is just another Connected Data Source—there is a Management Agent (MA) for it. But in another sense, they are quite integrated. In a typical setup, most (maybe all) identity objects in the MV will have a counterpart in the portal (or perhaps we might say that most objects in the Synchronization Service Database will have a counterpart in the Service Database). So you might think of the two databases as almost one (or that one is a shadow of part of the other).

Certainly, the FIM Service MA is a special case:

- There are no projection or join rules in the FIM MA. This happens automatically.
- The FIM MA uses simple classic configuration (not rules extensions rules) because we have to get the synchronization rules into the MV before we can use them (declarative synchronization rules cannot apply to the FIM MA).
- Provisioning also takes place automatically.

When you create the FIM MA, you define object type mappings in its properties, for example, matching person as presented by the FIM Service (the portal) to person in the metaverse, and the same for group. That is all you need to establish joining, projection, and provisioning.

Some object mappings are obligatory and are created automatically when you create the FIM MA: DetectedRuleEntry, ExpectedRuleEntry, and SynchronizationRule. We will cover these soon.

Synchronization Rule Definition

A synchronization rule includes three broad definition areas.

Scope

This is about which MA, which MV object type, and which Connector Space (CS) object types the rule applies to (like person in the MV and user in the AD LDS MA). Additionally, for inbound rules, you can specify a filter that decides which CS objects should be involved (unlike a classic MA filter, this will NOT lead to disconnections, it merely decides whether or not the rule applies).

Relationship

All declarative synchronization rules must have relationship criteria, i.e. at least one mapping of MV and CS attributes. This relationship is all about establishing a connection:

- For an inbound rule, this is like a join.
- For an outbound rule, it helps decide whether a new CS object should be provisioned (if there is already a connection to a CS object that matches the relationship, a new one will not be provisioned).

You also decide whether an object should be created:

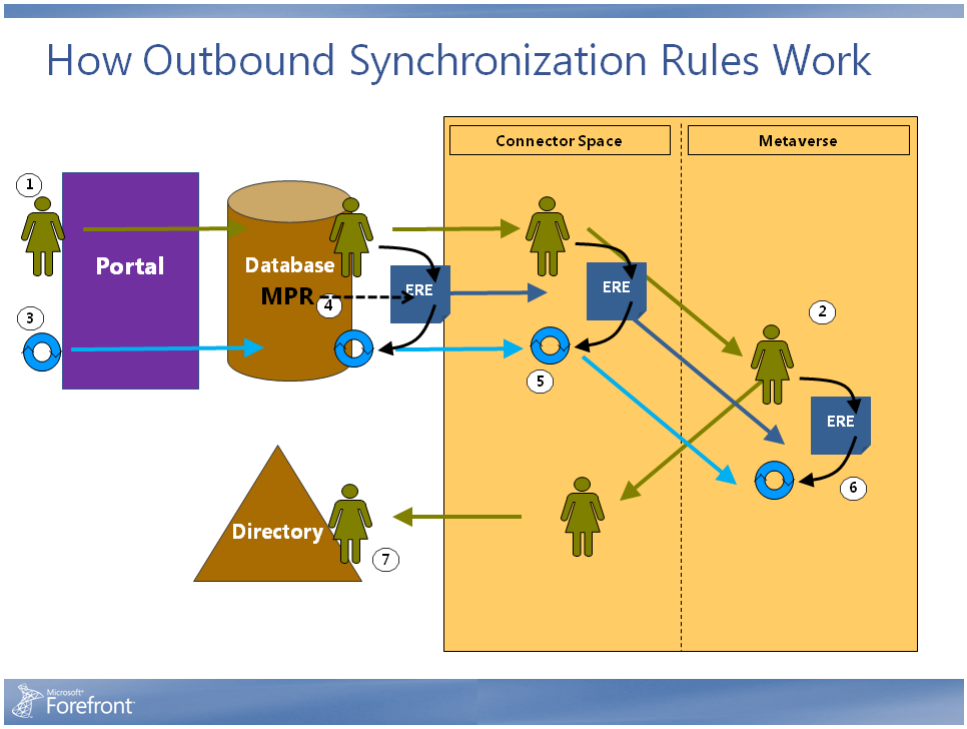
- In the case of inbound – Create in FIM, which means if there is no existing MV object to join to (one that matches the relationship), create an MV object (in classic terms, a projection).
- In the case of outbound – Create in the Connected System, which means if there is no existing CS object (one that matches the relationship), create a CS object (in classic terms, a provision).

Also, for outbound rules you can decide what should happen if the rule is removed (for example, the rule was deleted, although as you will see later there is another more likely way it could be removed). You can decide that the provisioned CS object is disconnected from the MV object. What happens to that disconnector will depend on the classic deprovisioning configuration for the MA concerned.

Attribute Flows

In declarative synchronization rules you can define attribute flows that are more complex than direct classic rules (for example, you can concatenate attributes and use functions), but note that classic rules extension rules ultimately give you the most flexibility (but at the cost of writing code).

How Outbound Synchronization Rules Work



Flow of Objects

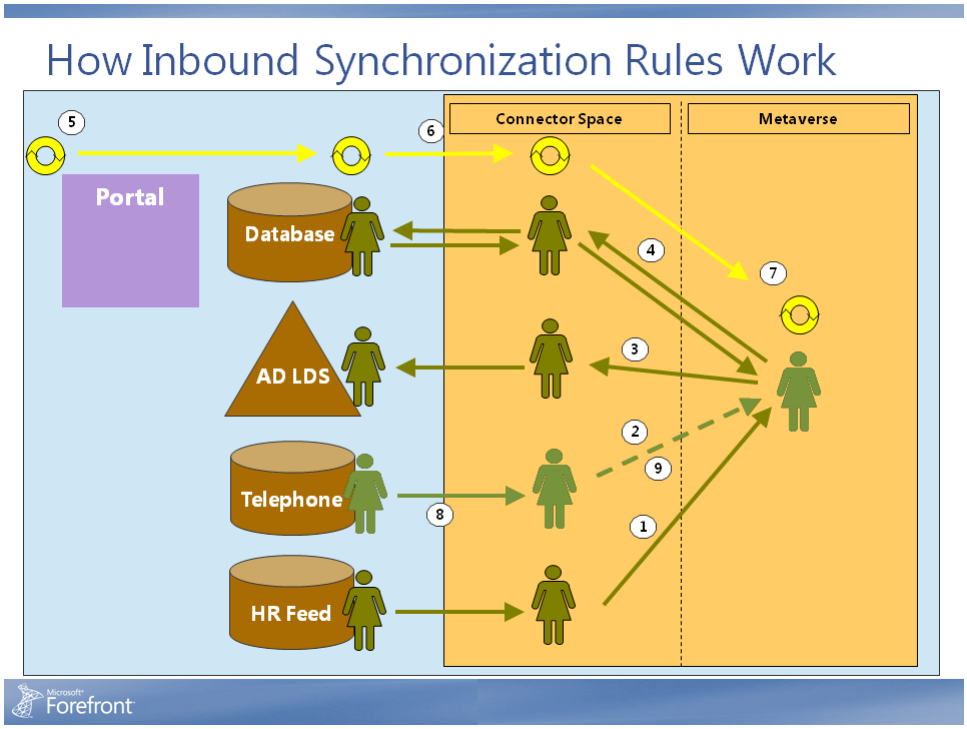
1. Let's suppose that we have a person (or user) who has been created in the portal (we may have many of course, and they don't have to have been created in the portal).
2. They are imported and projected into the MV (this does not need a synchronization rule, the FIM MA does this anyway as long as there is an object type mapping for person).
3. You define an outbound synchronization rule, for example, to provision an AD account.

   You also define a workflow to add the synchronization rule, and an MPR to say when (and to whom) that workflow should be applied.
4. Let's say that it triggers the workflow. It creates an Expected Rule Entry (ERE) object and a pointer to it in the Expected Rule List (ERL) of the person. The ERE itself points to the synchronization rule object.

   You might reasonably ask why there needs to be an ERE. Can't the person simply point to the rule? Well, the ERE carries information, for example, whether or not the rule has been applied.
5. All this is imported during a FIM Service MA import run.
6. The new objects are synchronized like any others. The synchronization rule and the ERE are projected, and the change to the ERL of the person flows into the MV.
7. During a second pass, the new synchronization rule is acted upon by the Synchronization Service, which (in our example) provisions the new account in AD (it has to be exported like any other).

All of the above applies to any identity resource type (like a group).

How Inbound Synchronization Rules Work



Inbound rules are simpler.

Reminder

Here is a reminder of the story so far. Using classic rules, we have:

1. Imported and projected persons from the HR Feed.
2. Imported and joined Telephone accounts.
3. Provisioned and exported AD LDS user accounts.
4. Created a FIM Service MA, which resulted in the auto-provisioning of users in the FIM portal/Application Database, and in the auto-projection of portal users in the MV.

Inbound Synchronization Rules

An inbound synchronization rule flows like this:

1. The inbound rule is created in the portal, and when submitted it is stored in the Application Database. As an example, we will say that it is a replacement for our classic rule for the Telephone Data system.
2. The rule is imported to the CS.
3. It is projected into the MV.

At this point we can delete the existing classic configuration. The synchronization rule in the MV will be picked up by the synchronization engine, which interprets it, including which MA it belongs to and which objects it is scoped for. For each CS object in scope, the rule will apply and could make a join in this case (in general, it could project or simply flow attributes). If already connected, attribute flow will take place provided that the object matches the rest of scoping configuration for the rule (if not, the rule is not executed for the object in question).

4. Next time a change arises in the Telephone Data system, the change is imported.
5. It is synchronized using our new rule.

After that, values could flow out to other data sources as needed.

Inbound Rule Steps

Inbound Rule Steps

1. Create the rule
   - Provide a name and description, and select **Inbound**
   - Decide which MA and pairing of resource types are to be involved
   - Define an external system scoping filter for inbound rules
   - Set the relationship criteria (like a join) – ideally, use data that will never change
   - Decide whether to select **Create resource in FIM**
   - Configure attribute flows
     - Attribute to attribute
     - Number or string to attribute
     - Expression to attribute

Microsoft Forefront

Inbound Rule Steps (Cont.)

2. Import the rule using the FIM Service MA
3. Perform synchronizations
   - Delta synchronization to get the rule into the MV
   - In the lab you are replacing classic rules; this is the time to remove them
   - Full synchronization of the relevant MA – to apply the new rule to the old data

- Outbound rule steps are more complicated, and are covered later

Microsoft Forefront

Configuration of Inbound Synchronization Rule

Configuration of synchronization rules is an option on the Administration home page. Start a new one and then configure as follows:

General

Provide a name and description and select **Inbound**.

> **Note:** Dependencies do not apply to inbound synchronization rules.

Scope

The synchronization rule will apply to a single MA and to a single pairing of object types (MVObjectType←→CSObjectType) within that agent. Select the **Metaverse Resource Type**, the **External System** (i.e. the MA), and the **External System Resource Type** (i.e. the CS object type).

Define an External System Scoping Filter to be applied to CS objects for the MA concerned, if required (in other words if you do not want this rule to apply to all objects from the data source).

> **Note:** Unlike a classic filter, this cannot lead to disconnection of an object. It simply decides whether the rule should be applied to the CS object in question.

Relationship

The relationship criteria will define a single join rule. The conditions are ANDed to reach the final rule, so all stated criteria must be true for the relationship to be initially formed. Note that the join in the Synchronization Service Database is not broken if the data on an object changes so that the relationship is no longer valid (for example, if the ShortName value changed in our Telephone Data Lab example). As with classic joining criteria, it is best to use attributes that are static and collectively unique.

You must have a relationship criterion and this can be seen as the formalization of what in any case is good practice.

> **Note:** Quite separate from the concept of dependency, synchronization rules have a precedence order you can define in the portal synchronization rules page. This is very useful if you have a number of potential relationship criteria (effectively join rules) to be applied in a particular order.

If you want to project the object into the MV on failure of the relationship criteria (and any classic join rules), select **Create Resource in FIM**.

Other options are unavailable, because they only apply to outbound synchronization rules.

Inbound Attribute Flow

Configure attribute flows. These can be simply a matter of selecting a Source (CS) attribute and a Destination (MV) attribute, for example:

- displayName → displayName
- Fax → officeFax
- LastName → sn

However, you could flow a String or a Number:

- 'Full Time Employee' → employeeType
- 42 → TheNumber

And there are a range of functions, or you could use a custom expression (where you have functions of functions):

- FirstName + Left(LastName,1) → mailNickName
- CustomExpression(UpperCase(Left(FirstName,1)+" "+LastName)) → displayName

Submit

Submit your rule and it is written to the database.

Import the Rule

You now run a Delta Import run profile for the FIM Service MA to import this new object (your new synchronization rule).

Synchronization

There are two steps:

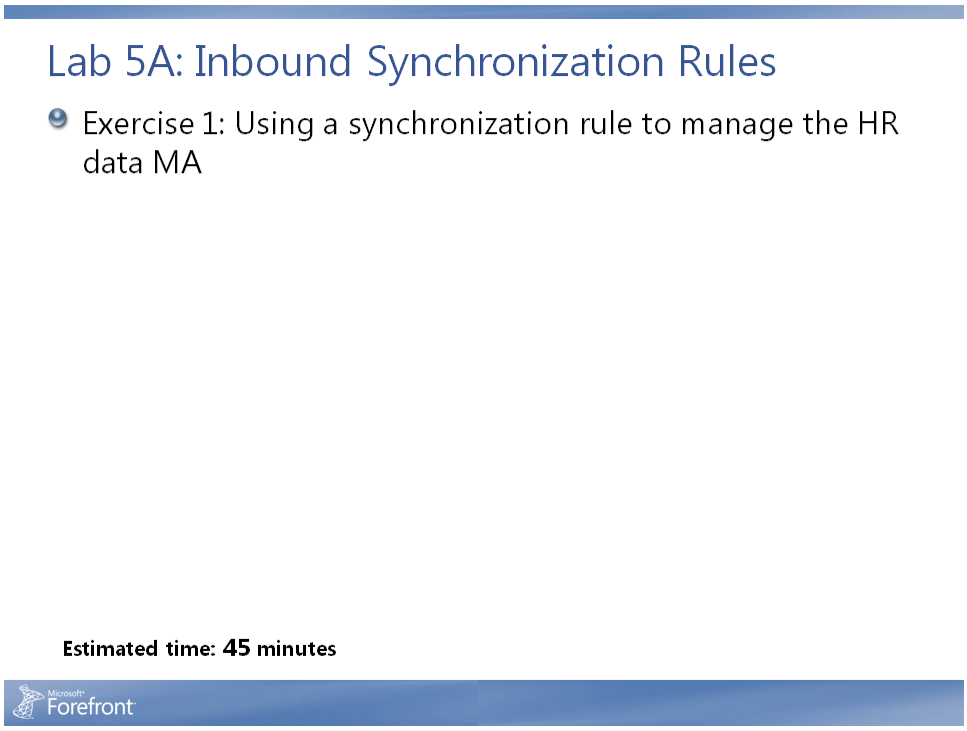
1. Run a Delta Sync run profile for the FIM Service MA to project your new rule into the MV (you will see some export flow, which is the MV GUID flowing out to the FIM Service).
2. Run a Full Sync run profile of the relevant MA (the MA referred to in the synchronization rule) in order to apply the new rule to the existing data.

If you are replacing existing classic configuration, as in the lab that follows, then remove that configuration before Step 2 above. No great harm would be done if you left both in place (if they do not agree, the classic rules win), but it is a waste of effort.

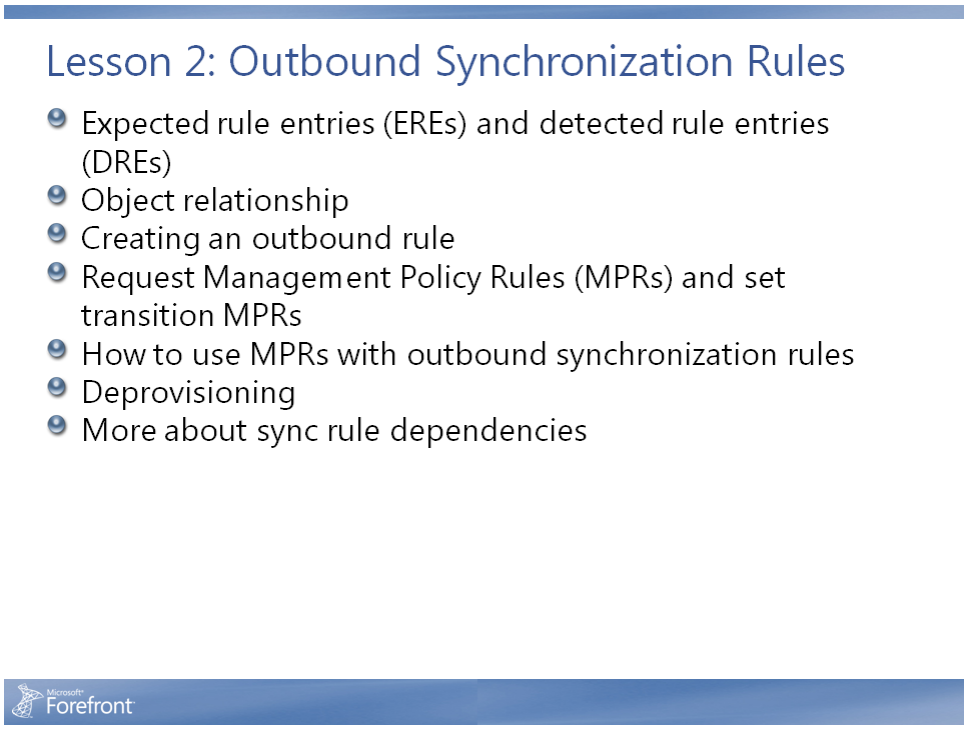
From now on, regular production Delta Sync run profile is all that is needed.

> **Note:** Outbound rules are more complicated and are covered in the next lesson.

Lab 5A: Inbound Synchronization Rules

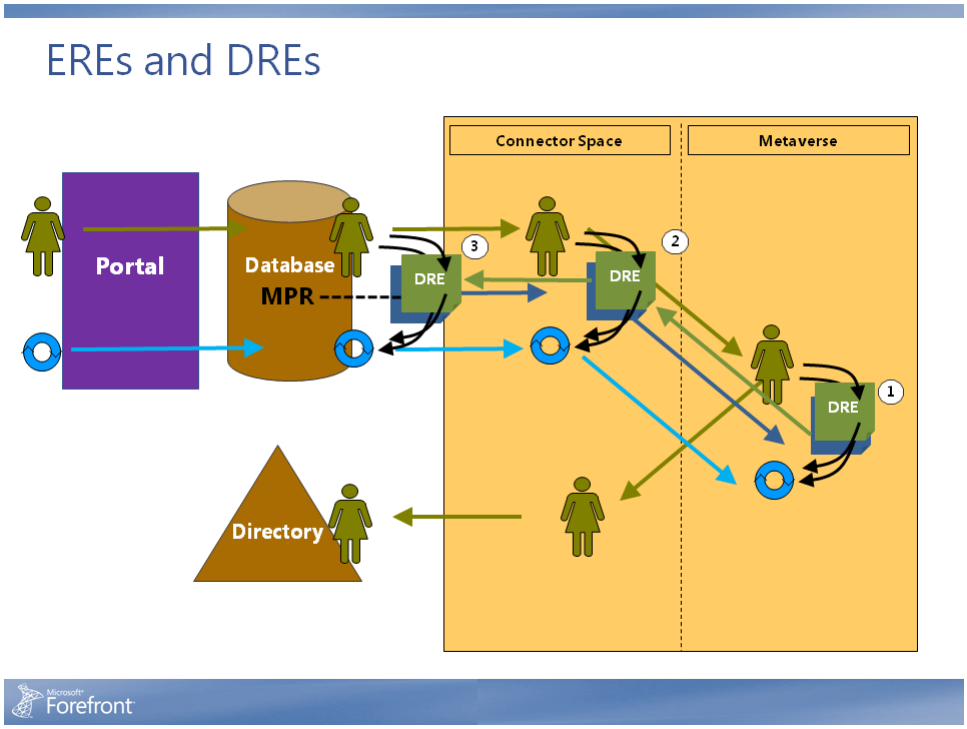


Lesson 2: Outbound Synchronization Rules



In this lesson, we examine outbound synchronization rules for provisioning and deprovisioning, which means brief coverage of workflows and further coverage of Sets and MPRs.

EREs and DREs



Flow of Objects

You have seen the part played by an ERE. In summary:

- A user (person) has been created in the portal.
- They are imported and projected into the MV by normal FIM Service MA behavior.
- You define an outbound synchronization rule, for example, to provision an AD account.
- You define a workflow to add the synchronization rule, and an MPR to say when (and to whom) that workflow should be applied.
- The workflow creates an ERE.
- All this is imported during a FIM Service MA import run.
- The new objects are synchronized like any others. The synchronization rule and the ERE are projected, and during a second pass the new synchronization rule is acted upon by the Synchronization Service, which (in our example) provisions the new account in AD (which is then exported).

When configuring a declarative synchronization rule, you can optionally define one or more attribute flows as existence tests.

If the destination (CS) attributes involved are successfully flowed to the data source concerned, and if this is also confirmed by an import (or if the values in the data source objects already correspond to the values which would flow if the rule were applied), the object is deemed to exist and a Detected Rule Entry (DRE) is created.

You might choose just one key attribute flow for this, like AccountName to sAMAccountName, as an existence test (unless you really wanted to be sure that other particular attributes had flowed).

The mechanism is as follows (assuming that an attribute has been set as an existence test):

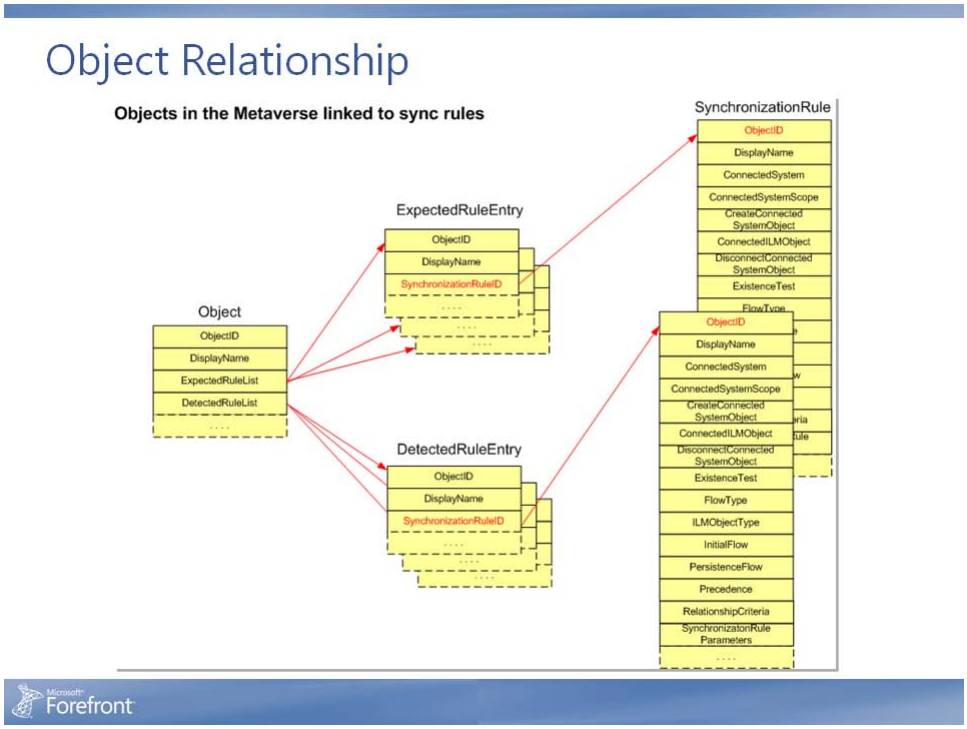
1. The new object has just been exported, and you should always follow an export with an import (to confirm that it got there) and a synchronization (to ensure that any changes brought in are properly handled—passed on, or perhaps overridden). Exports are always delta, and for a directory like AD, the import will be delta too (the synchronization should certainly be a delta).

   In this case we will say that the existence test is satisfied, and the Synchronization Service creates a DRE pointing to the relevant rule, and adds a reference to the DRE in the Detected Rules List (DRL) attribute for the person object.

2. The DRL attribute change and the new DRE object both flow out to the CS.
3. During the next Export run, the DRE and the modified person are exported.

You can now check in the portal for the existence of the DRE (for example, on the provisioning page for a user, or an MPR could trigger a workflow), which tells us whether the synchronization that gave rise to the ERE has been successful.

Object Relationship



Key Points

- Identity objects (such as person and group) have two multi-value reference attributes that are used to manage declarative synchronization for them. The ERL is a list of pointers to EREs. The DRL is a list of pointers to DREs.

> **Note:** Every identity object (that might have references to EREs) must have a classic import flow of the ERL attributes in the FIM MA. If DREs are in use, they must also have classic export flow of the DRL attributes.

- EREs are objects that contain at least a GUID, an action (such as Add), and a reference to a synchronization rule object. Additionally, EREs may contain parameters that you define and that are used in the synchronization for this object (for example, suppose that you generate a random password to be used for provisioning—this would be stored here, temporarily).
- The synchronization rule object contains all the relationship, scope, attribute flow, and so on, that is needed to perform the synchronization.

Creating an Outbound Rule

Creating an Outbound Rule

- Provide a name and description and select **Outbound**, or **Inbound and Outbound**
  - Inbound and Outbound rule is mostly an outbound rule, with an inbound rule hitching a ride
  - Inbound and Outbound will ensure a join
- Apply a dependency – a rule that must be applied (ERE applied) before this rule is applied (in which case the new rule only needs outbound attribute flow)
- Decide which MA and pairing of object types are to be involved (**External System Scoping Filter** is not used for outbound rules)

Microsoft Forefront

Creating an Outbound Rule (Cont.)

- Define the relationship criteria, and, if required, select **Create Resource in External System**
- Decide what should happen if the rule is removed: Should classic disconnection logic apply?
- Enter attribute flows with appropriate options
  - Some attributes must only flow initially (e.g., password)
  - Lots of attributes only need to flow persistently (e.g., manager, department)
  - Very few attributes need to flow both initially and persistently (e.g., distinguished name (DN))
  - Some attributes flow differently initially than they do persistently (e.g., userAccountControl)
  - Use as existence test if you want a DRE to be generated

Microsoft Forefront

Configuring synchronization rules is an option on the Administration home page. Start a new one and then configure as follows:

General

Provide a name and description and select **Outbound**, or **Inbound and Outbound**.

Inbound and Outbound Rules

An inbound and outbound rule is really almost the same as an outbound rule, but with an inbound rule hitching a ride. You should think of it as two completely separate rules handily combined in the interface. Although you will have to associate it with specific resources (for example, users) using MPRs and workflows to generate EREs, the inbound part of the rule leads a quite separate existence, applying to whatever CS objects are in scope.

We have stated elsewhere that you should always have a join rule, or the equivalent, for a synchronization rule, which is an inbound rule with a relationship (this is so that connections can be re-made if broken).

It follows that if you only intend to have outbound flow (in a particular case), you would do well to choose the inbound and outbound type, even though there is no inbound flow (in fact, no additional configuration at all), just to ensure a join, should one be needed.

> **Note:** It is worth emphasizing that the inbound and outbound type is nothing more than a combination of an inbound and an outbound rule handily combined in the interface. Also, if a particular case requires only outbound flow, you could choose inbound and outbound in order to provide for joining.

Dependency

If necessary, define a dependency. This is a rule upon which this rule depends. The dependency will be an outbound rule (or an inbound and outbound rule, but in that case it is only the outbound part that matters), and our new rule will only be applied once this dependency rule has been applied (that is, an ERE corresponding to the other rule must exist and have been applied). Our new rule will have the same scope and relationship (same MA, same object types and relationship), and will only have outbound flow.

So for example, we might have a main synchronization rule to provision a mailbox-enabled account for a user, and other synchronization rules (applying to different sets of users) that flow different values for mailbox quota. These additional rules will have the first (main) rule as a dependency because there is no point in flowing a mailbox quota until you have the account (and also because it saves you having to define scope and relationship all over again). There is more on this at the end of this module.

Scope

The synchronization rule will apply to a single MA, and to a single pairing of object types (MVObjectType←→CSObjectType) within that agent. Select the **Metaverse Resource Type** (for example, person), the **External System** (MA, for example, AD LDS) and the **External System Resource Type** (a CS object, for example, user).

The External System Scoping Filter is unavailable unless you choose the inbound and outbound type, because it is only used for inbound rules.

Relationship

The relationship criteria for an inbound synchronization rule will define a join rule. In this case, for an outbound rule, we must still define relationship criteria (although it does not do anything significant except in the unusual circumstances where multiple objects of different types are associated with a single metaverse object). As for a join rule, it makes sense to choose unique and static attributes for the relationship criteria.

If you want provisioning to take place (if no resource satisfies the relationship criteria already), select the **Create resource in external system** option, so that objects will be provisioned into the connected system (this is all you have to do—compare that with using the DLL in the lab in Module 3). **Create resource in FIM** is grayed out unless you chose the inbound and outbound type (because it only applies to an inbound rule).

Decide on **Enable Deprovisioning**. If the synchronization rule is removed from the resource in FIM (by a workflow, or even by deletion of the rule), should the CS object be disconnected from the MV object? If this is selected, then the normal, classic deprovisioning logic takes over.

Typically, you would expect that if a rule is authoritative for creating the resource, then it would also be authoritative for resource deletion, or put more simply, if you select **Create Resource in External System** it is likely that you will also select **Enable Deprovisioning**. There is more on deprovisioning later.

Parameters

Parameters are used to exchange attribute values with custom workflows which you may define. We do not cover them in this course.

Outbound Attribute Flow

Now configure all the Outbound Attribute Flows, which behave very much like Inbound Attribute Flows, but with a few additions.

Types of attribute flow

There are two types of attribute flow, **Initial Flow Only** and **Persistent**. You will not see the word Persistent in the synchronization rule interface—rules are Persistent, unless you select **Initial Flow Only**.

When an object is first created it must be initially populated with a few attributes before the rest of the synchronization—effectively the normal (persistent) attribute flow—can take place. These attributes are Distinguished Name (DN) (or the anchor if there is no DN) and any *set and forget* attributes.

Some attributes must only flow initially

A password, for example, must only flow initially because you cannot read and synchronize passwords like other attributes (there is an entirely separate mechanism for this). These will have **Initial Flow Only** selected.

Lots of attributes only need to flow persistently

Generally, lots of attributes, like manager, telephone number, name, address, department, job title, etc., only need to flow persistently because you clearly want these to update as changes arise. These will NOT have *Initial Flow Only* selected.

Very few need to flow both initially and persistently

DN is the prime example of an attribute that may need to flow initially and also persistently, and so needs two attribute flows. FIM requires DN (or anchor if there is no DN) to be initially populated, but you may want it to be updated as changes arise. The question should not arise for other anchors because you shouldn't modify anchors.

> **Note:** If you require an attribute to flow both initially and persistently (a common requirement for DN), it is necessary to create two attribute flows: one with Initial Flow Only selected and one with Initial Flow Only deselected (for persistent flow).

Some attributes flow differently initially than they do persistently

userAccountControl can be initially set to 512, but subsequently we need something quite different (if we want to change individual bits).

Use as existence test

If you want a DRE to be returned on successful provisioning, choose some reliable data to **Use as Existence Test** (you could sensibly use the attributes in the relationship criteria).

Request MPRS and Set Transition MPRs

Request MPRs and Set Transition MPRs

- Request MPRs can make use of transitions, but set transition MPRs are made to work with transitions
- MPRs can run workflows – there are three types of workflow:
  - Authentication
  - Authorization
  - Action
- Request MPRs can grant permission for a set of requestors to perform an operation on a target set, and/or they can trigger workflows to run when this happens – used for:
  - Granting permissions
  - Running authentication and authorization workflows
  - Running some action workflows – e.g., notifications

Microsoft Forefront

Request MPRs and Set Transition MPRs (Cont.)

- Set transition MPRs:
  - Cannot grant permissions
  - Run workflows: must be action workflows, based on a defined transition set, depending on the transition type (Transition In or Transition Out)
  - Are especially suited to making decisions about outbound synchronization
- Run mapping: applies a new or modified set transition MPR to all members of the transition set (if it is a Transition In type)

Microsoft Forefront

Set Transitions

We have already mentioned the idea of a transition in relation to **Request MPRs**, in which we saw that permissions could be granted to Create and Delete, and so on, but in which we also saw that permissions could be granted to allow a change to an attribute such that the target moved from one set to another. This concept of transitions is so important that there is another type of MPR, called a **Set Transition MPR**, which exclusively deals with Set Transitions. We'll come to these in a moment.

Workflows

MPRs can run workflows. In fact, in FIM, the only way to run a workflow is to get an MPR to trigger it. So a Request MPR can grant permission for a set of requestors to perform an operation on a target set, but it can also trigger workflows to run when it does. An MPR doesn't have to grant permissions—it can just run workflows.

There are three types of workflow:

- **Authentication** – We might need additional authentication before proceeding with a request (such as asking a user questions before allowing a password reset).
- **Authorization** – We might need authorization before proceeding with a request (such as getting the owner of a group to approve a new member).
- **Action** – We might need an action to happen on successful completion of a request (such as sending a notification that a membership request has been approved or that a contractor has transitioned to full-time employee, or provisioning an account when a new user has been created).

Request and Set Transition MPRs

Request MPRs

As we have said, a Request MPR can grant permission for a set of requestors to perform an operation on a target set, and/or it can also trigger workflows to run when it happens. These are used for:

- Granting permissions.
- Running Authentication and Authorization workflows.
- Running some Action workflows, for example, notifications.

Set Transition MPRs

Set Transition MPRs cannot grant permissions. They simply run a workflow, and it has to be an Action workflow, based on a defined Transition Set, depending on the Transition Type:

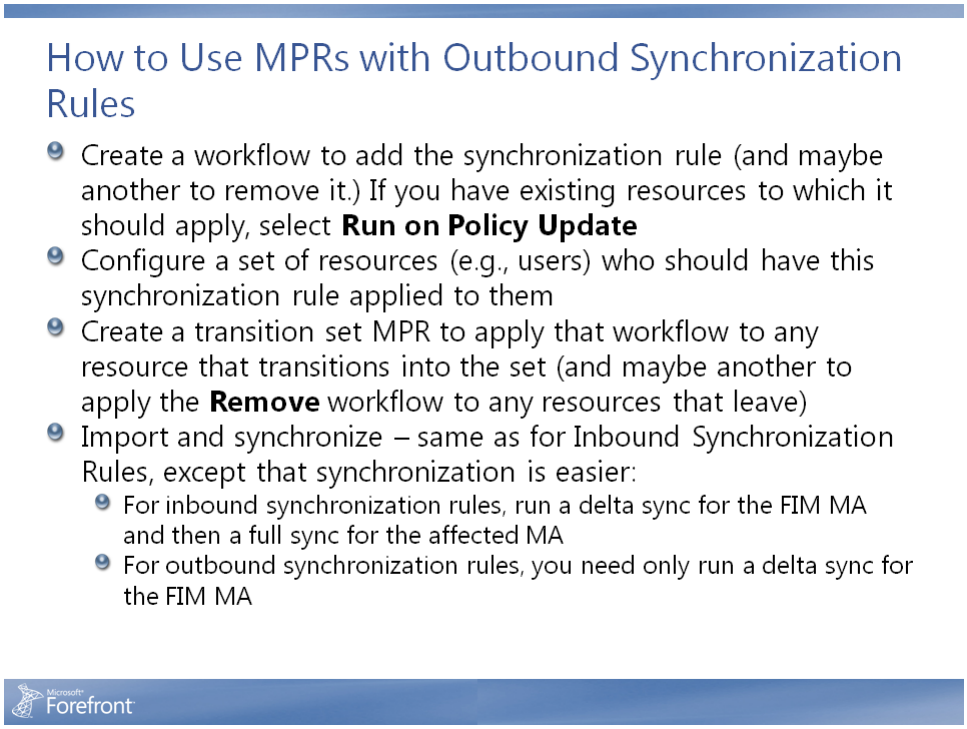
- **Transition In** – Applies the workflow to resources that become members of the transition set.
- **Transition Out** – Applies the workflow to resources that leave the transition set (including deletion of the set).

So they are much simpler and they are especially suited to making decisions about outbound synchronization (for example, whether or not to provision an account, or whether to flow a particular attribute).

Run Mapping

Transition Set MPRs have another trick up their sleeve—they can perform **Run Mapping**. If you mark a workflow to **Run on Policy Update**, then if you modify the MPR (and Creation counts as a modification), or you modify the Set on which it depends, the marked workflows will be applied to all members of the Transition Set (if it is a Transition In type).

How to Use MPRs with Outbound Synchronization Rules



To make use of an outbound (or inbound and outbound) synchronization rule, you must:

- Create a workflow to Add the synchronization rule (and maybe another to Remove it). If you have existing resources to which it should apply, select **Run on Policy Update**.
- Configure a Set of resources (for example, users) to whom this synchronization rule should apply.
- Create a Transition Set MPR to apply that workflow to any resource that transitions into the Set (and maybe another to apply the *remove* workflow to any that leave, or transition out of, the set).
- Import and synchronize.

You import and synchronize as for inbound synchronization rules, except that synchronization is easier. For an inbound synchronization rule, you run a delta sync run profile for the FIM MA and then a full sync run profile for the affected MA. For an outbound synchronization rule, you need only to run a delta sync run profile for the FIM MA.

Example

Suppose we have:

- A synchronization rule to add an AD Account.
- A set for All Active People.
- Workflow1 to add the synchronization rule (with Run on Policy Update selected).
- Workflow2 to remove the synchronization rule.

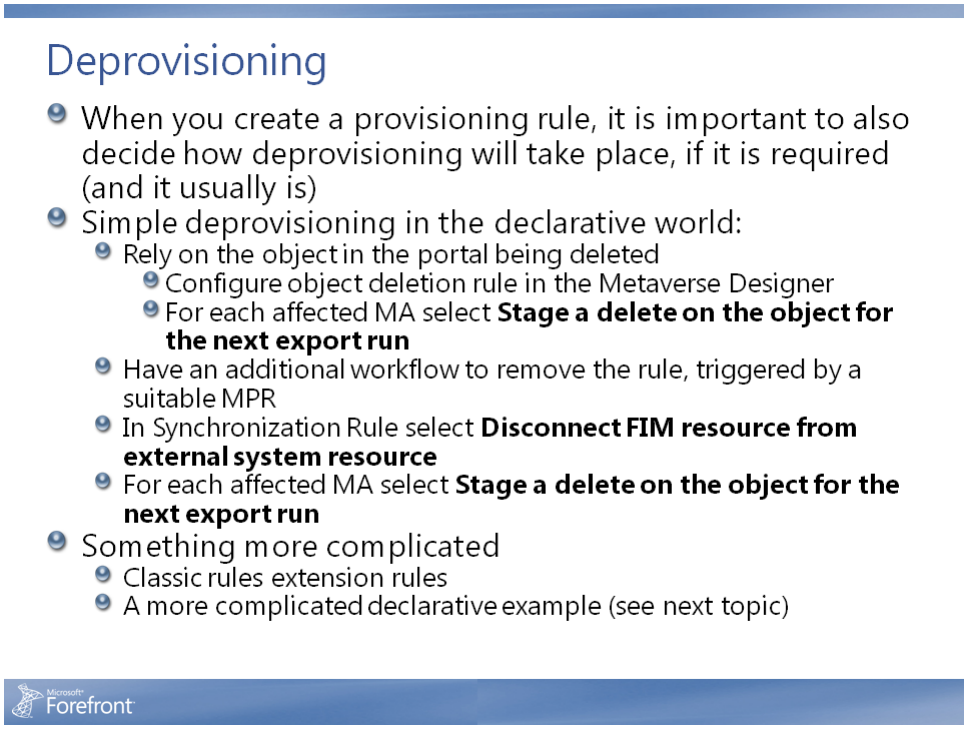
We then create two Transition Set MPRs (MPR1 and MPR2) that run Workflow1 and Workflow2 (respectively) when a user Transitions In and Out (respectively) of the Set.

The subsequent behavior can be summarized thus:

Event	Action
MPR1 is first created	Workflow1 is triggered for every member of the All Active People Set, adding an ERE for each of them with an entry in each of their ERLs pointing to their respective ERE (the EREs all point to the synchronization rule); the FIM Service MA imports all the EREs and modified users, and the synchronization rule is applied to all of them.
New user created who falls into All Active People	MPR1 triggers Workflow1 which adds an ERE pointing to the synchronization rule, and a reference to the ERE in the user's ERL; the FIM Service MA imports the ERE and modified user, and the synchronization rule is applied.
A user is modified such that they drop out of All Active People	MPR2 triggers Workflow2 which removes the ERL entry and deletes the ERE. The associated account could be disconnected, depending on the configuration of the synchronization rule; if the Set is deleted, this happens to all users in the Set.
A user in All Active People is deleted	Whatever else happens, referential integrity removes the ERE, the associated MV object could be deleted, depending on the configuration of the MV Deletion Rule, and the associated accounts will be disconnected, possibly resulting in deletion depending on the configuration of the Deprovisioning Rules for the MAs concerned.

Run on Policy Update does not work for Transition Out Set Transition MPRs. Module 8 covers this in some detail.

Deprovisioning



While provisioning is somehow more dramatic than deprovisioning, it is just as important, and often more complex, to determine the processes which govern the end-of-life activities for an object.

Disable→Displace→Delete is an example process for AD accounts, but in reality this will be dependent on local policies and practices. In considering deprovisioning, it is important to understand the triggers as well as the actions involved.

Simple Deprovisioning in the Declarative World

Assuming that you simply want to deprovision (rather than disable or displace), the following two approaches can be taken in the declarative world:

1. Actual object deletion from the portal – Usually you would expect this to result in corresponding MV objects and previously provisioned objects being deleted too, but that depends on configuration. This method is a satisfactory solution for a case where everyone gets an account until they are deleted. For actual deletion of previously provisioned objects (accounts) to take place, you must make sure that:
    a. In the Metaverse Designer, the Object Deletion Rule is configured in such a way that the corresponding MV object will be deleted when the afore-mentioned disconnection takes place.
    b. For each affected MA, **Stage a delete on the object for the next export run** is selected.

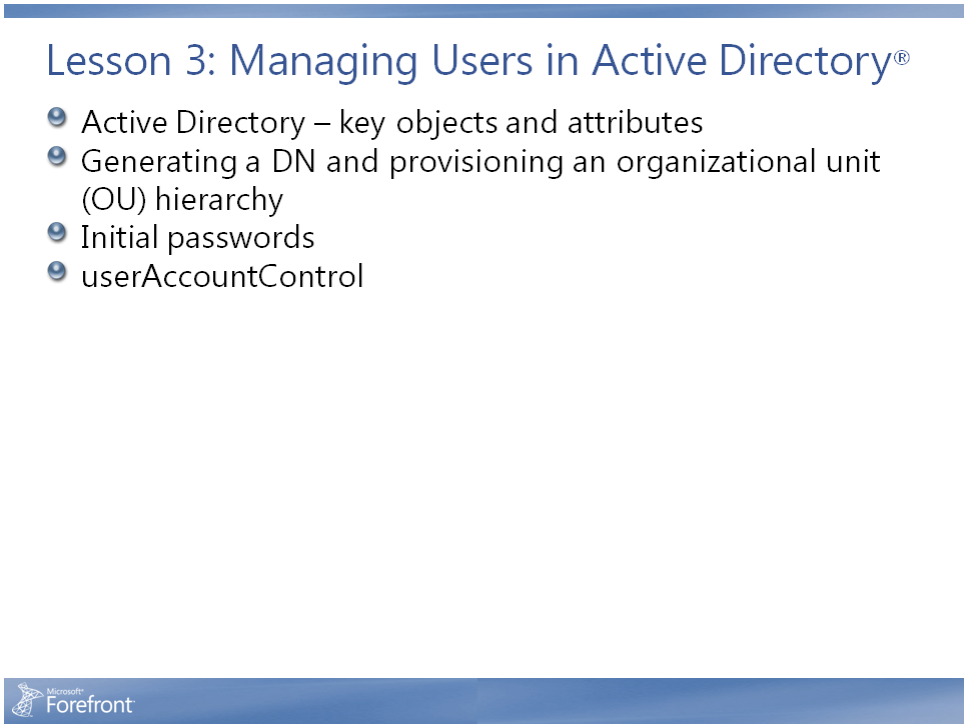
2. A workflow removes the synchronization rule from the object concerned – It would be normal to create a counterpart workflow that does a remove, and another MPR to detect the change you are interested in. For actual deletion of previously provisioned objects (accounts) to take place, you must make sure that:
    a. In the synchronization rule, **Disconnect FIM resource from external system resource when this Synchronization Rule is removed** is selected, so that the CS object for the FIM MA disconnects from the corresponding MV object.
    b. For each affected MA, **Stage a delete on the object for the next export run** is selected.

Something More Complicated

If you are prepared to write code, you can be far more complicated in how you handle the end of an object's life cycle.

In the declarative world, you can also get a little bit smarter. See the final topic in this module.

Lesson 3: Managing Users in Active Directory



In this lesson we discuss the specifics of managing Active Directory® Domain Services (AD DS) with FIM. We look at the key objects and attributes involved, as well as issues concerning DN generation (both for initial provisioning and persistent renames), setting passwords, and controlling the activation status of the user through userAccountControl.

Active Directory – Key Objects and Attributes

Active Directory – Key Objects and Attributes

- Object types include User and Group, and organizationalUnit (or other container) – FIM can create these as needed
- Attributes with uniqueness requirements: userPrincipalName, sAMAccountName, DN (and preferably displayName and group CNs)
- Attributes important to Microsoft® Exchange Server: homeMDB, mDBUseDefaults, mailNickName

Microsoft Forefront

Active Directory – Key Objects and Attributes (Cont.)

- Special cases:
  - DN must have an **Initial Flow Only** flow, even if it also has a persistent flow. You may want to generate a DN based on attributes
  - unicodePWD (and perhaps PwdLastSet) can only have an **Initial Flow Only** flow (it cannot be persistent). You must decide on a suitably secure policy in this case
  - userAccountControl is often set to 512 with an **Initial Flow Only** flow. A persistent flow needs some thought
- Managing Active Directory® Lightweight Directory Services (AD LDS) has much in common with managing AD

Microsoft Forefront

Users and groups are the most commonly managed object types in AD, but others, such as contact, might be involved, and there will always be a container of some kind, such as Organizational Unit (OU).

Where an object is required as a container, it can be automatically provisioned, provided its parent (or grandparent, etc.,) exists and is being managed.

Attributes with Uniqueness Requirements

Various AD attributes will possess uniqueness requirements, either across multiple forests (although these are rare), a single forest, or only within a single domain. Among these are:

- userPrincipalName, which must be unique across an AD forest.
- sAMAccountName must be unique within a single AD domain (preferably also unique across an AD forest or multiple AD forests).
- DN must be unique across all connected AD environments, if you are synchronizing with more than one.

There are other AD attributes that are often configured as unique as a best practice even though it is not a technical requirement. For example, if Microsoft® Exchange Server 2010 is in use within an AD environment, attributes such as displayName (for users and contacts) and cn (for groups) should have enforced uniqueness so that Address Book entries are unambiguous.

In the FIM portal, Account Name is selected (by default) for uniqueness within the portal (although it doesn't provide any help, or even a helpful message). Provided that the portal is authoritative for all objects, Account Name can be reliably used to meet the uniqueness requirements in attribute flows for DN, sAMAccountName, and userPrincipalName. Any more sophisticated requirements will need custom workflows or some other solution (like a classic rules extension).

Exchange Attributes

Attributes which are required for provisioning Exchange Server 2010 mailboxes include:

- homeMDB – The DN of the mailbox store which will contain the new mailbox.
- mDBUseDefaults – Specifies that the default settings (drawn from the server configuration) to be used for settings such as mailbox size limits.
- mailNickName – The short name for the mailbox, which needs to be unique within a mailbox store.

Special Cases

DN Needs Initial Flow Only

Each object provisioned to AD requires a DN. DN is notionally the anchor in this case (although, under the covers, objectGUID is being used so that DN changes can be tracked), so like all anchors this **must** be set using an **Initial Flow Only** setting. If you wish to perform renames (which involve changing the DN) you will need a persistent flow as well. This requires two (probably identical) flows, one set for Initial Flow Only flow and the other not.

Your DN will be generated using one or more attributes (see next topic).

unicodePwd and pwdLastSet

As mentioned elsewhere, some attributes require Initial Flow Only, others require a persistent flow, and some require both.

The attribute unicodePwd has to be an Initial Flow Only attribute flow and is also write-only. It may be written once, at provisioning time, and subsequently it can only be reset using the password reset facilities. Therefore, a persistent flow is not permitted.

Setting pwdLastSet to 0 as an Initial Flow Only will force a password change at first logon.

userAccountControl

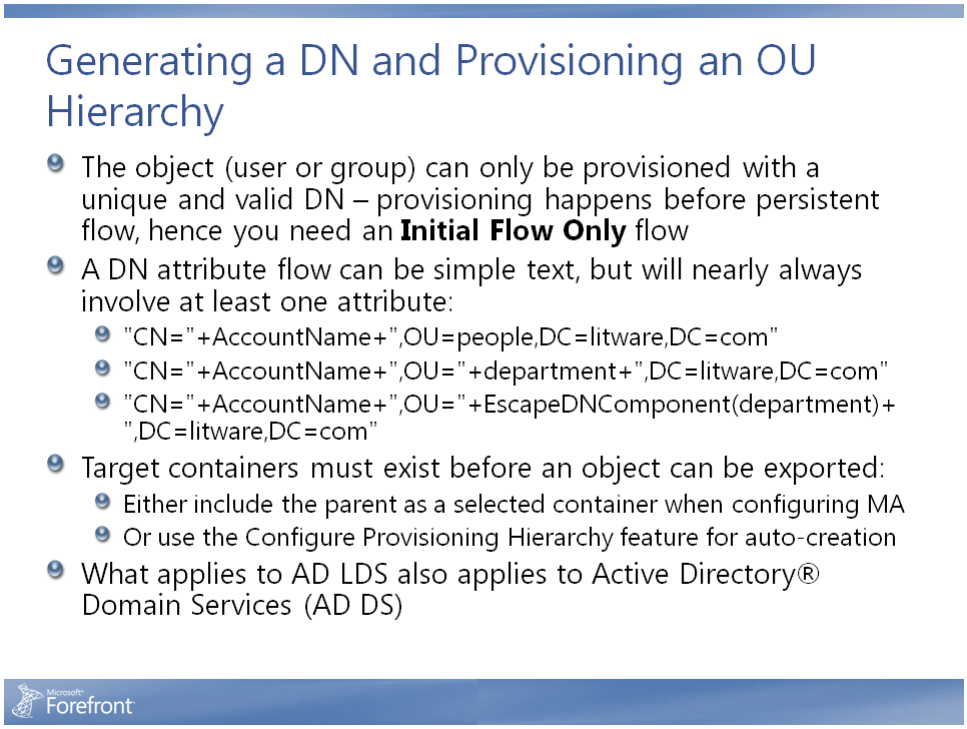
The userAccountControl attribute is often set to an initial value using an Initial Flow Only attribute flow: either 512 for a normal account, or 514 for a normal but disabled account.

Subsequently, the userAccountControl attribute can be used to control various properties of the account, by far the most common of which is the enabled/disabled status of the account. This requires manipulation of the "2" bit, and turns out to have a few other twists. This is discussed in detail shortly.

AD LDS

AD LDS, the lightweight directory service formerly known as ADAM, is managed in a similar way except the schema tends to be different, the Exchange Server 2010 attributes are not relevant, and the enabled/disabled state is controlled using the Boolean msDS-UserAccountDisabled attribute instead of the bitwise userAccountControl. Otherwise, the issues such as management of the DN, or unicodePwd, are much the same.

Generating a DN and Provisioning an OU Hierarchy



DN Requirements

DNs must be unique (to the MA, at least) and set during provisioning with an Initial Flow Only attribute flow. Unless your rule will only ever manage a single object, this flow will be constructed using at least one attribute to guarantee uniqueness. For example:

```
"CN="+AccountName+",OU=people,DC=litware,DC=com"
```

```
"CN="+AccountName+",OU="+department+",DC=litware,DC=com"
```

Where you use attribute values to generate DN components, you need to think about special characters. A comma makes for an awkward DN (since it is the separator used between components of the DN), but it is allowed provided you make use of the EscapeDNComponent function, for example (assuming that the department attribute might contain a comma):

```
"CN="+AccountName+",OU="+EscapeDNComponent(department)+",DC=litware,DC=com
```

Actually, you can exclude special characters simply by forbidding these characters with a validation (beyond our scope here). The AccountName attribute has such a validation.

The EscapeDNComponent function escapes a comma by preceding it with a backslash. A DN component like "HR, US" appears as "HR\, US" so that the comma is identifiable as a piece of data and not a separator.

The RDN (Relative Distinguished Name, the first component), particularly for users, will commonly be globally unique, as a policy, because of the potential for renames. Although the RDN only needs to be unique in the current container of the object, if the object can be moved to another container, the

potential for naming conflicts exists, and an RDN-uniqueness policy is a good idea. In the preceding examples, the use of AccountName at least partially achieves this.

Containers

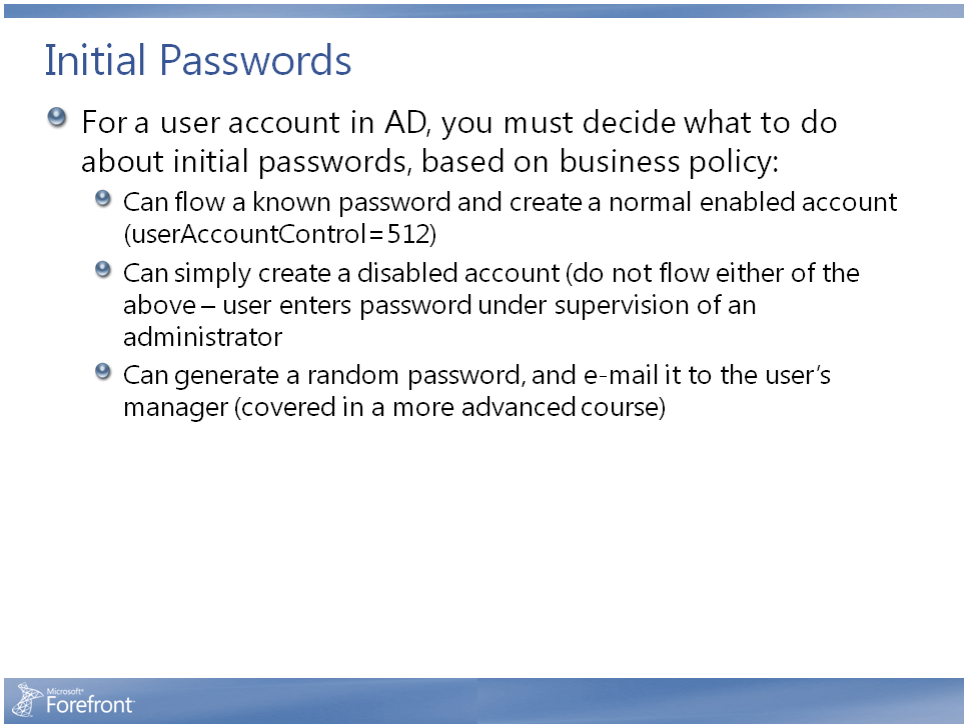
The containers into which you provision objects must be selected in advance in the MA, or (if they do not already exist) they can be provisioned automatically using the Provisioning Hierarchy settings in the MA.

Configure Provisioning Hierarchy

In AD MAs (AD DS or AD LDS), you can define what kind of containers should be created if a DN needs it to be created. These settings identify the type of object to be provisioned for each RDN component prefix (for example, ou → organizationalUnit), and provided that the new container can be provisioned below an existing container (one that is included in the MA's Containers settings) any missing containers below it will be created.

> **Note:** The hierarchy of your organizational units does not have to bear any relationship to the structure of your organization. We used department as an example, but OU structures should be based on whatever is administratively most convenient.

Initial Passwords



Initial Password Policy

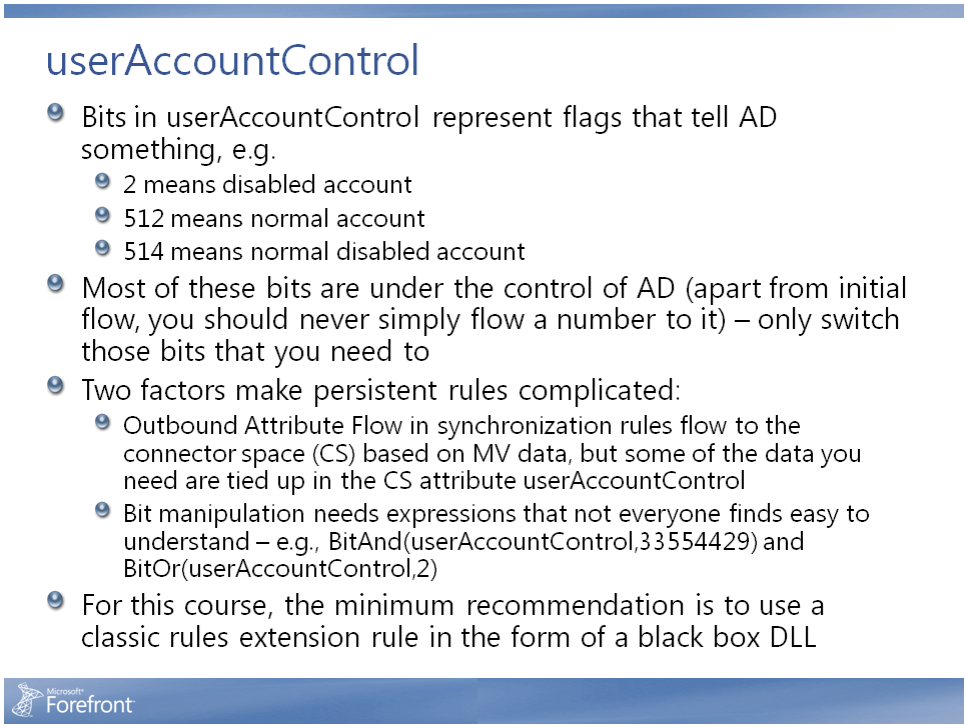
The policies for the setting of initial password range from the very secure but high-effort (for example, blind-printed envelope sent to the address of the user used on their pay slips), to medium-effort (personal visit to helpdesk with photo ID) to the less-secure but relatively simple (for example, set automatically and e-mailed to the user's manager, or set by a local helpdesk by telephone request), to the very simple (set to a standard value). The policy appropriate for a particular environment depends on the local requirements.

Simple Examples

Some simple examples we can use on the course are:

- Set userAccountControl to 512, set unicodePwd to a known value (as initially configured for the course), and set pwdLastSet to 0.
- Set userAccountControl to 514, and require a user to sit with (physically or on the telephone) admin or the helpdesk, and enter their password for the first time.

userAccountControl



userAccountControl

Related to the question of initial password policy is the question of whether an account is provisioned in an enabled state or not. In the aforementioned case of an initial password set by the helpdesk, the account can be enabled by the helpdesk user, in which case it need not be provisioned in an enabled state. In other cases the user must be initially enabled. By default, a user account is provisioned in a disabled state and any other requirement is implemented in an Initial Flow Only to the userAccountControl attribute.

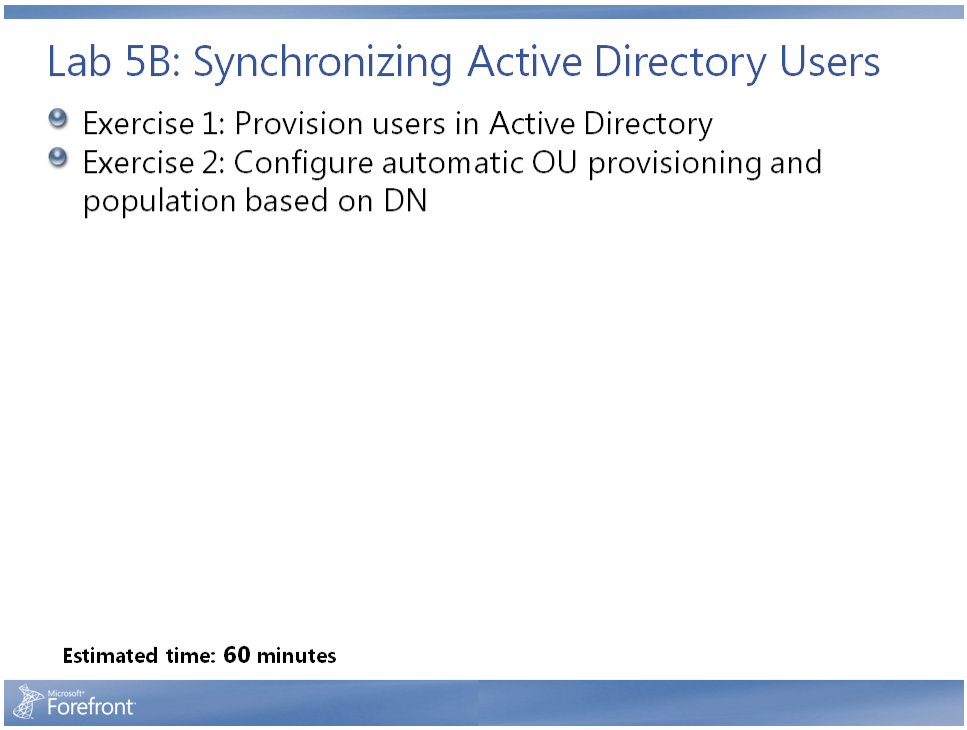
> **Note:** The userAccountControl attribute is not an ordinary attribute like other attributes (such as department or description, which represent just *one* meaning). Rather, userAccountControl represents several different account security settings. The bits of the userAccountControl attribute control different security settings, so altering one security setting, such as enable/disable account, is more complex when the account already exists because you cannot be certain of the other security settings' values (represented as bits in the userAccountControl).

The ongoing control of the enabled state relies on a persistent flow to the userAccountControl attribute, but there are two issues that make this difficult:

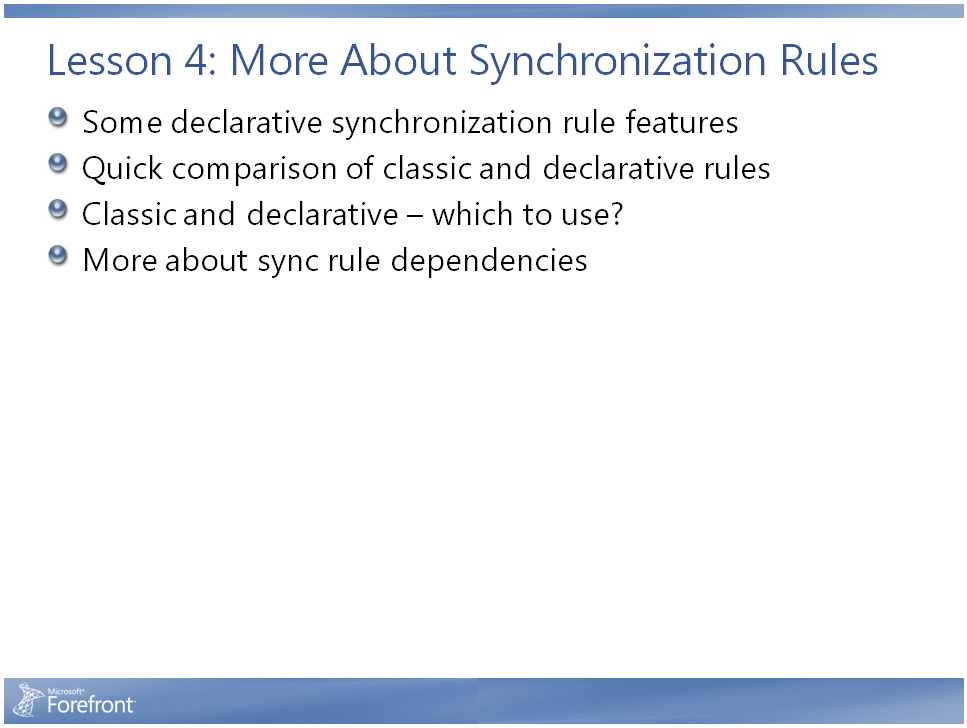
1. AD is authoritative for most of userAccountControl, so when we edit it we must make sure we start with the latest value from AD; the latest we have is sitting in the CS, but our flow rules flow from the MV to the CS.
2. The actual expressions required for bit manipulation are far from simple for non-programmers. You can end up with expressions like BitAnd(userAccountControl,33554429) and BitOr(userAccountControl,2).

In a more advanced course we can explore this issue in depth, but in this course we are going to take a really simple way out. We are going to make use of a classic rules extension rule in the form of a DLL (and we ask that you accept it as a *black box* that does the job). Arguably, this really is the simplest and best way to do this task.

Lab 5B: Synchronizing Active Directory Users

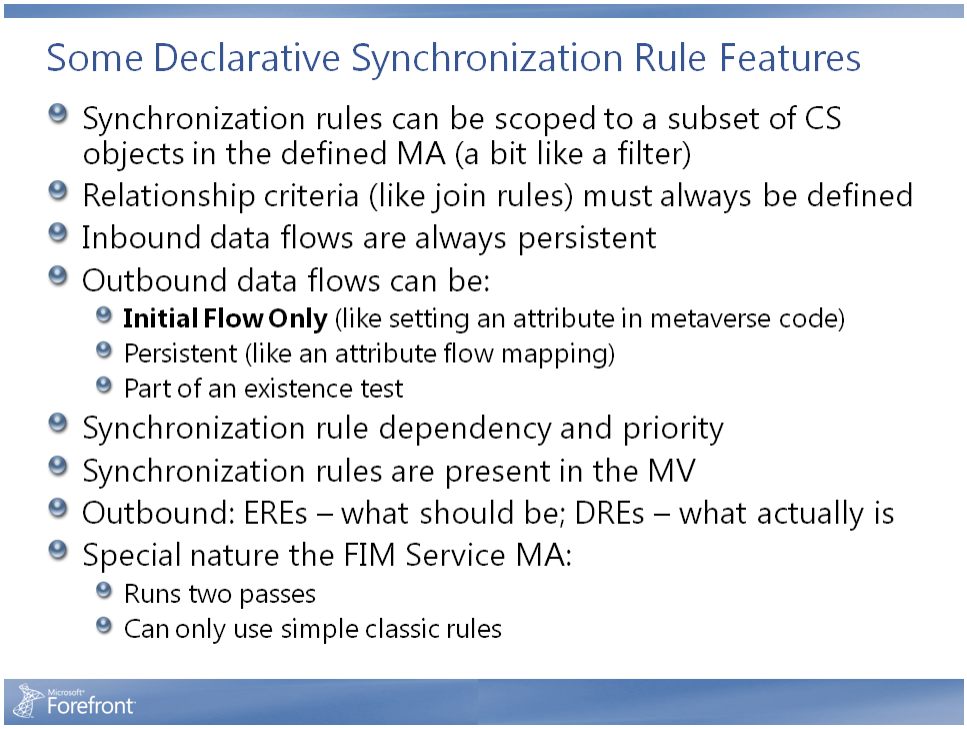


Lesson 4: More About Synchronization Rules



In this lesson we look in more detail at some features you have seen and we introduce some further features and ideas.

Some Declarative Synchronization Rule Features



Synchronization Rule Scope Filter

As already mentioned, a synchronization rule has a scope, which defines which MA and which objects are involved. For an inbound synchronization rule (or the inbound part of an inbound and outbound rule), you can also define an External System Scoping Filter. This filter can further restrict to which CS objects the synchronization rule should apply, for example, **EmployeeStatus equal Active**.

This has some similarities to the classic MA filter, but note that it is an inclusion filter, and not an exclusion filter. The major difference is that it does not control connection like a classic MA connector filter does. The fact that the filter is false (or becomes false) does not lead to disconnection.

Mandatory Relationship Criteria

It is a best practice to define a join rule even if the normal use cases for an MA do not explicitly require it, so that data recovery works. In FIM synchronization rules, this best practice has become mandatory. Synchronization rules must define relationship criteria and this is very important for inbound rules because inbound rules define joins which would be used during disaster recovery.

For outbound rules the relationship criteria are not always important (and the cases where they are important are beyond our scope here), but as a matter of good practice they should be entered with the same care as for inbound rules.

For a relationship (join) to be effective, you may have to breadcrumb the data source with a permanent unique joining value (for example, user name or employeeID) so that a reliable join is always possible. Ephemeral breadcrumbs are not a good idea, for example, a Metaverse GUID that could change during a disaster recovery rebuild is not suitable.

> **Note:** In the case of a single, simple outbound rule that creates users in AD (for example), the relationship criteria does not appear to matter at all. However, in more complex cases, where multiple rules could create multiple connectors for a single MV object (for example, a user and an admin account for a particular person) the relationship can have an effect.

In scenarios where complex joining is required (for example, where a transformation is required as well), classic join rules are still available. However, if you intend for declarative synchronization rules to take over, you will need to ensure that joining can be done on simple values (for example, by breadcrumbing). You should not see relationship criteria as equating exactly to join rules. Join rules can perform complex joins. Relationship criteria will only perform simple joins (although there is a case for saying that these are all you normally need).

Persistent and Initial Attribute Flows

Inbound data flows are always persistent. Outbound data flow can be:

- Persistent, in which case it is updated whenever the source attribute in the Metaverse changes. This corresponds exactly to Classic Attribute Flow Rules.
- Initial Flow Only, meaning that the attribute is only set when the object is originally created.

Initial Flow Only applies to outbound synchronization only and is analogous to the setting of attribute values in provisioning code. This kind of flow must be used for the anchor (or Distinguished Name), must be used for initial password set, and should be used for any other *set and forget* attributes.

> **Note:** It is sometimes necessary to have both types of flow (two separate definitions). This is true for the DN (which is the anchor attribute in an LDAP MA like the Active Directory type), so that an initial DN is set, and so that subsequent renames are properly handled.

Existence Test

An Outbound Attribute Flow can have a further setting—you can make it part of an existence test. Any number of outbound flows can be defined as being part of an existence test.

If:

- The synchronization rule is in scope for an object (based on object type).
- And the values in the data source match the values specified by the Outbound Attribute Flow for each flow which is marked as *use as existence test*.
- And it has been confirmed by an import.

Then the object is deemed to exist correctly, and a DRE is created and exported to the portal where it can be detected (manually or programmatically).

Synchronization Rule Dependency and Priority

A synchronization rule can be configured to depend on another synchronization rule having been applied. The final topic in this lesson provides some examples.

Also, there is a synchronization rule precedence order that you can modify, which defines the order of processing. You would use this, for example, to specify the order in which relationship rules (joins) are applied (for inbound rules).

Rules are Present in the Metaverse

Synchronization rules are imported by the FIM Service MA, and projected into the MV as synchronization rule objects. What happens next depends on the type of rule. For an outbound rule, to apply to, say, a person object, there must be a reference between the person and the object. This relationship is implemented using another object, the ERE, and is configured using an MPR. For inbound rules, no such reference is necessary. Put another way:

- For an outbound synchronization rule to apply to an object, an MPR must have created an ERE that links the object to the rule.
- Inbound synchronization rules apply to all objects (existing and future) as long as they meet the scoping requirements (MA, object type, filter, etc.).

Expected Rules Entries and Detected Rules Entries

These two additional objects support outbound synchronization and are also imported to the MV.

As just discussed, an ERE defines that a specific synchronization rule should apply to a specific MV object. For a given object, there is a list of EREs known as the ERL (actually a multi-value reference attribute), which defines the whole set of rules which should apply to this object.

EREs define what *should be* the case for an object, but it is also often necessary to track what *is actually* the case, and therefore there is a counterpart to the ERE, known as the DRE. DREs are created when a defined existence test is true, and the corresponding list of DREs is the DRL (another multi-value reference attribute).

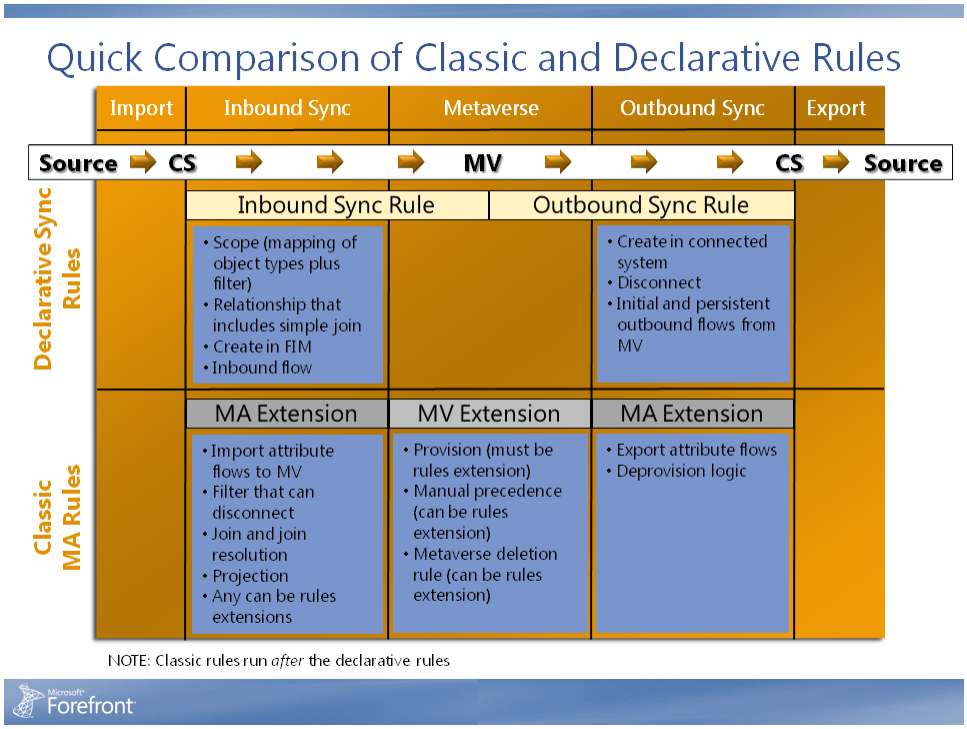
Two Pass Processing in the FIM MA

The FIM Service MA, because of its special role in processing rules as well as data objects, has a unique processing approach. When the Synchronization Service is synchronizing the FIM MA, it performs two passes. The first pass processes objects related to rules (synchronization rules, EREs, and DREs), to make sure that the most up-to-date rules and their links with data objects are used, and then the second pass processes the data objects themselves.

Exceptions in the FIM MA

As already mentioned, declarative synchronization rules cannot apply to the FIM MA.

Quick Comparison of Classic and Declarative Rules



Declarative Synchronization Rules

Inbound

Scope:

- Which system (which MA).
- What pairing of MV object to CS object.
- Filter (a bit like a classic connector filter but reversed).

Relationship:

- You must define a criterion. It corresponds to a join rule (but it is compulsory).
- You can indicate if an object should be created in the MV.

Inbound Attribute Flow:

- Direct
- String or number
- Functions and expressions

Outbound

Scope:

- Which system (which MA).
- What pairing of MV object to CS object.

Relationship:

- You can indicate if an object is to be created in a connected system.
- You can indicate that the object should be disconnected if the rule is removed (i.e. if the ERE is deleted), but the classic disconnection rule decides what to do next.

Outbound Attribute Flow:

- Can be Initial Flow Only (like setting an attribute in a provision rule).
- Can be persistent (like inbound flow).
- Direct
- String or number
- Functions and expressions
- Any number of outbound flows can be defined as being part of an existence test. If both ends of the rule correspond, the object is deemed to exist correctly and a DRE is created and exported to the portal where it can be detected.

Classic Rules Extension Rules

Simple classic rules can be configured in the Synchronization Service Manager interface. Portal synchronization rules offer more flexibility, but classic rules extension rules offer the most capability.

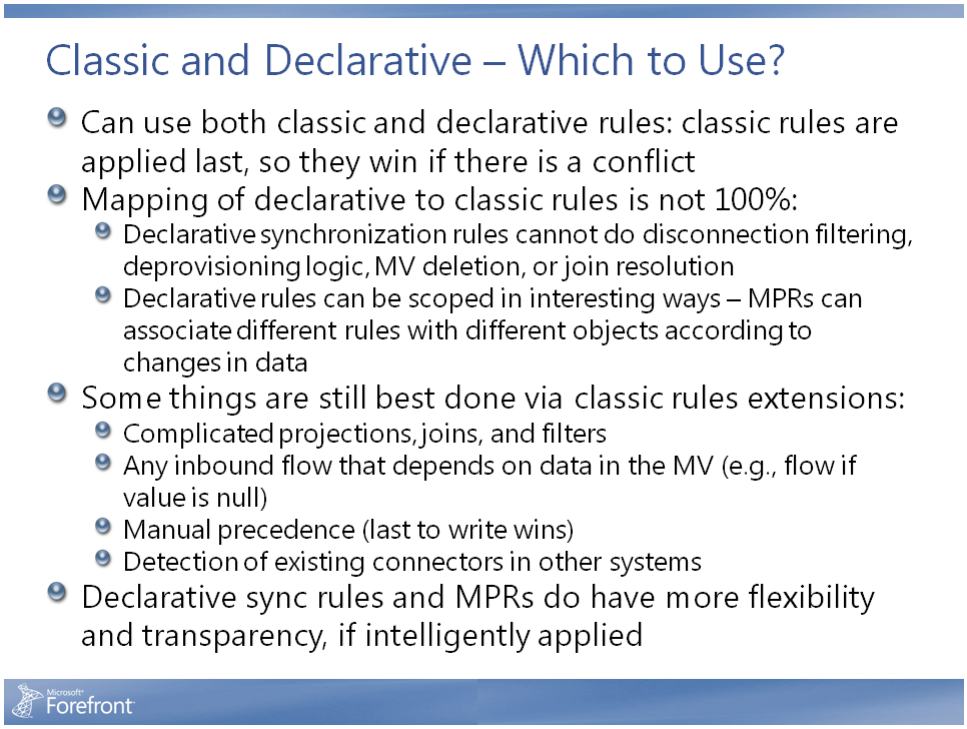
Classic rules are always necessary for:

- Disconnection logic (may or may not involve an extension).
- When to delete an MV object – the MV deletion rule (may or may not involve an extension).

Classic rules extension rules may be necessary for:

- Complex joins and join resolutions.
- Complex precedence – *manual precedence.*
- Complex projection.
- Complex attribute flow.
- Complex provisioning and deprovisioning requirements.

Classic and Declarative – Which to Use?



Classic and Declarative Synchronization Rules are Both Available

The classic FIM MA configurations, as well as MA and MV code extensions, are all still fully supported.

The classic rules are processed after any declarative synchronization rules and, therefore in case of a conflict, the classic rules win.

Synchronization Rules and Classic Rules Do Not Map 100%

In many cases, the synchronization rules of some classic rules are not identical in their behavior to the classic rules. For example, a scoping filter is a bit like a classic connector filter. However, it is reversed, and (like a connector filter) will prevent objects from joining, but (unlike a connector filter) will not disconnect a join if a joined object no longer meets its criteria.

The application of synchronization rules is in some ways more flexible than the (non-coded) classic rules (although coded rules extension rules are very flexible). The classic rules apply to an object type in general, or, in the case of attribute flows, to each MVObjectType←→CSObjectType combination. Synchronization rules are applied using policies, which can be driven by the attributes of an object, so different objects of the same type, which have key differences in their attribute values, can have very different rules applied to them. Intelligent use of these subtle scoping possibilities can reduce the need for coded extensions.

Classic Rules Extension Rules Are Still Useful in Some Cases

It is likely that in most implementations, classic rules will still be used, and not just in migration cases where ILM configurations already exist.

Examples of features that are not possible in declarative rules are deprovisioning logic, MV object deletion rules, and join resolution.

Classic rules extension rules can handle more complex scenarios:

- Projection rules – for example, check uniqueness before projecting.
- Join and join resolution rules – for example, mapping on partial data, looking up nicknames or even *sounds like* tokens, although it should be mentioned that cleaning source data is often a better approach than using complicated join rules.
- Attribute flow rules – Rules extension rules see more than the declarative rules, for example, an outbound flow rule can be made to depend on CS attributes (not possible with declarative rules).
- Precedence – Declarative rules can only use standard precedence or equal precedence.
- Deprovisioning – Complex requirements.

The list is almost endless because once you are in code, all things are possible.

Policy-driven Declarative Synchronization Rules are Flexible and Transparent

The advantage of declarative synchronization rules, however, is a clear gain in the transparency of the system. The contents of the rules can be inspected (the specific rules applied (or waiting to be applied) to an object can be seen, which aids troubleshooting), and the flexible scoping of the rules, described above, allows novel rule assignments.

More About Synchronization Rule Dependencies

More About Synchronization Rule Dependencies

- Synchronization rule dependencies is an intricate subject – best explained by examples:
  - Example 1: Required to flow different attributes for FTEs and contractors – could have:
    - One common rule
    - Two more rules to handle exceptions
  - Example 2: Different sizes of Exchange Server mailboxes for different users – could have:
    - One common rule for all AD users
    - Another rule for mailbox-enabled users
    - Three more rules to handle different sizes

Microsoft Forefront

More About Synchronization Rule Dependencies (Cont.)

- Example 3: Disabled accounts moved to special folder
  - Have three synchronization rules (and associated sets, workflows, and MPRs)
    - A – live account (deprovision if removed)
    - B – flows attributes to disable
    - C – disable and move (change DN)
    - A takes precedence over B, which takes precedence over C
  - Live → disable → displace → delete = A → AB → ABC → none

Microsoft Forefront

This is a surprisingly intricate subject, and is perhaps most easily explained through examples.

Example 1: Flowing different attributes for FTEs and Contractors

We can have one synchronization rule for FTEs and one for contractors, but generally there will be many common flows. If we create three rules, one common and one each to handle the special requirements for FTEs and contractors, we save effort (we do not have to repeat ourselves). If we do that, we will need to make each of the special synchronization rules dependent on the common synchronization rule—the dependency is simply a matter of the existence of EREs. We should also pay attention to the order in which the synchronization rules are run. It may not matter, but there is a simple and attractive logic in the common synchronization rule running first, provisioning an object if necessary, followed by the special synchronization rules.

Example 2: Different sizes of Exchange Server mailboxes for different users

We might want to set different sizes (100MB, 250MB, and 500MB) for different sets of users, but we would not want to set the Exchange Server attributes if there is no Exchange Server mailbox for the user. So we might have:

1. A common AD attributes synchronization rule.
2. A synchronization rule that sets Exchange Server attributes, which is dependent on the above; this is added (using an MPR) to those users that are supposed to have a mailbox.
3. Three more synchronization rules to specify the different mailbox sizes, each dependent on the above; these are added to different sets of users using MPRs.

Again, the logical order is 1, then 2, and then the 3 others in any order.

Example 3: Disabled accounts moved to special folder

When someone leaves, you could arrange for their account to be deleted, or you could disable it and move it to a different OU.

You could have three rules:

- Rule A, which flows a particular DN and attributes to maintain a live account. Enable Deprovisioning is selected.
- Rule B, which flows attributes so as to disable the account; it takes precedence over Rule A.
- Rule C, which flows a different DN (for a container to hold disabled accounts) and attributes so as to disable the account; this takes precedence over Rule B.

We can use Rules A, B, and C with suitable Sets and MPRs to control the Disable→Displace→Delete process:

- Live accounts – When a user is created, add Rule A.
- To disable an account – Add Rule B (to the set of disabled users, perhaps employeeStatus="inactive").
- To displace an account – Add Rule C (to the set of expired users, perhaps based on date or employeeStatus, or whatever).
- To delete an account – Remove Rules A, B, and C.