Red Hat Directory Server 12

Managing the directory schema

Creating and managing the custom schema

Last Updated: 2024-01-15

Red Hat Directory Server 12 Managing the directory schema

Creating and managing the custom schema

Legal Notice

Abstract

You can store additional data in the Directory Server by adding custom schemas, which are created by using the dsconf utility and the web console. You can also extend the schema and validate the syntax of existing attributes value.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For submitting feedback through Jira (account required):
  1. Log in to the Jira website.
  2. Click **Create** in the top navigation bar
  3. Enter a descriptive title in the **Summary** field.
  4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
  5. Click **Create** at the bottom of the dialogue.
- For submitting feedback through Bugzilla (account required):
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. CREATING A CUSTOM SCHEMA USING THE DSCONF UTILITY

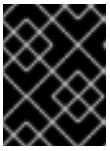
You can add custom attributes and object classes to Directory Server by extending the schema. You can extend the schema:

- Using the **dsconf** utility on the command line. This process is described in this section.
- Using the Directory Server web console .
- Manually by creating schema files .

1.1. WORKFLOW OF A SCHEMA EXTENSION

Adding new schema elements requires:

1. Planning and defining unique object identifiers (OID) for the new schema. Directory Server recognizes schema elements by their OID, but you must manage the OIDs manually.
   An OID is a dot-separated number that identifies the schema element to the server. OIDs can be hierarchical with a base OID that can be expanded to accommodate different branches. For example, the base OID could be **1**, and there can be a branch for attributes at **1.1** and for object classes at **1.2**.

   IMPORTANT

   Even if not required, Red Hat recommends to use numeric OIDs for custom schemas for better forward compatibility and performance.

2. Request OIDs from the Internet Assigned Numbers Authority (IANA). For details, see https://pen.iana.org/pen/PenApplication.page.
3. Create a OID registry to track OID assignments and to ensure that no OID is used for more than one purpose. An OID registry is a list of all OIDs used in the directory schema including descriptions. Publish the OID registry with the custom schema.
4. Define the new attributes.
5. Define the object classes that contain the new attributes. However, never update the default schema. If you create new attributes, always add them to a custom object class.

Directory Server loads the schema when the instance starts. To load new schema files, restart the instance or initiate a reload task.

Keep the following rules in mind when customizing the Directory Server schema:

- Keep the schema as simple as possible.
- Reuse existing schema elements whenever possible.
- Minimize the number of mandatory attributes defined for each object class.
- Do not define more than one object class or attribute for the same purpose.
- Do not modify any existing definitions of attributes or object classes.

WARNING

Do not update or delete the standard schema to avoid compatibility problems with other directories or LDAP client applications.

1.2. HOW DIRECTORY SERVER MANAGES SCHEMA UPDATES IN A REPLICATION ENVIRONMENT

When you update the directory schema in the **cn=schema** tree, Directory Server stores the changes in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file, including a change state number (CSN).

Directory Server does not directly replicate the schema changes to other replicas. Schema replication starts when directory content is updated in the replicated tree. For example, if you update a user after modifying the schema, the supplier compares the CSN stored in the **nsSchemaCSN** attribute with the one on the consumer. If the value of the **nsSchemaCSN** attribute on the consumer is lower than the one on the supplier, Directory Server replicates the schema to the consumer. For a successful replication, all object classes and attribute types on the supplier must be a superset of the consumer's definition.

Example 1.1. Schema subsets and supersets

- On **server1**, the **example** object class allows the **a1**, **a2**, and **a3** attributes.
- On **server2**, the **example** object class allows the **a1** and **a3** attributes.

In the previous example, the schema definition of the **example** object class on **server1** is a superset of the object class on **server2**. During the validation phase, when Directory Server replicates or accepts the schema, the server retrieves the superset definitions. For example, if a consumer detects that an object class in the local schema allows less attributes than the object class in the supplier schema, Directory Server updates the local schema.

If the schema definitions are successfully replicated, the **nsSchemaCSN** attributes are identical on both servers and the schema definitions, such as object classes and attributes types, are no longer compared at the beginning of a replication session.

In the following scenarios, Directory Server does not replicate the schema:

- The schema on one host is a subset of the schema of another host.
  For example, the schema definition of the **example** object class on **server2** is a subset of the object class on **server1**. Subsets can also occur for attributes (a single-value attribute is a subset of a multi-value attribute) and attribute syntaxes.
- When definitions in supplier schema and consumer schema need to be merged.
- Directory Server does not support merging schemas. For example, if an object class on one server allows the **a1**, **a2**, and **a3** attributes and **a1**, **a3**, and **a4** on the other, the schemas are not subsets and cannot be merged.
- You use schema files other than **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif**. Directory Server enables you to add additional schema files to the **/etc/dirsrv/slapd-**

instance_name/schema/ directory. However, only the CSN in the **/etc/dirsrv/slapd-instance_name/schema/99user.ldif** file is updated. For this reasons, other schema file are used only locally and not automatically transferred to replication partners.

IMPORTANT

To enable Directory Server to automatically replicate the schema and to avoid duplicate schema definitions, store the custom schema in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file.

1.3. USING DSCONF TO CREATE A CUSTOM SCHEMA FOR AN ATTRIBUTE AND OBJECT CLASS

This procedure demonstrates how to use the **dsconf** utility to create a custom schema with:

- A single-valued attribute named **dateOfBirth** with OID **2.16.840.1.1133730.2.1.123** and syntax **Directory String** (OID **1.3.6.1.4.1.1466.115.121.1.15**)
- An object class named **exampleperson** without parent object class (**SUP top**), OID **2.16.840.1.1133730.2.1.99** that must contain the **dateOfBirth** attribute.

Procedure

1. Create the **dateOfBirth** attribute:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com schema attributetypes
add --oid="2.16.840.1.1133730.2.1.123" --desc="For employee birthdays" --
syntax="1.3.6.1.4.1.1466.115.121.1.15" --single-value --x-origin="Example defined"
dateOfBirth
```

2. Create the **exampleperson** object class:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com schema objectclasses
add --oid="2.16.840.1.1133730.2.1.99" --desc="An example person object class" --
sup="top" --must="dateOfBirth" examplePerson
```

3. Run a schema reload task:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com schema reload
```

Verification

- Monitor the **/var/log/dirsrv/slapd-*instance_name*/errors** file:
  - If the build succeeds, Directory Server logs:

```
[23/Sep/2021:13:47:33.334241406 +0200] - INFO - schemareload -
schemareload_thread - Schema reload task starts (schema dir: default) ...
[23/Sep/2021:13:47:33.415692558 +0200] - INFO - schemareload -
schemareload_thread - Schema validation passed.
[23/Sep/2021:13:47:33.454768148 +0200] - INFO - schemareload -
schemareload_thread - Schema reload task finished.
```

- If the build fails, Directory Server logs which step failed and why.

CHAPTER 2. CREATING A CUSTOM SCHEMA USING THE WEB CONSOLE

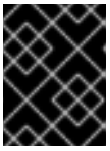
You can add custom attributes and object classes to Directory Server by extending the schema. You can extend the schema:

- Using the Directory Server web console. This process is described in this section.
- Using the dsconf utility on the command line .
- Manually by creating schema files .

2.1. WORKFLOW OF A SCHEMA EXTENSION

Adding new schema elements requires:

1. Planning and defining unique object identifiers (OID) for the new schema. Directory Server recognizes schema elements by their OID, but you must manage the OIDs manually.
   An OID is a dot-separated number that identifies the schema element to the server. OIDs can be hierarchical with a base OID that can be expanded to accommodate different branches. For example, the base OID could be **1**, and there can be a branch for attributes at **1.1** and for object classes at **1.2**.

   IMPORTANT

   Even if not required, Red Hat recommends to use numeric OIDs for custom schemas for better forward compatibility and performance.

2. Request OIDs from the Internet Assigned Numbers Authority (IANA). For details, see https://pen.iana.org/pen/PenApplication.page.
3. Create a OID registry to track OID assignments and to ensure that no OID is used for more than one purpose. An OID registry is a list of all OIDs used in the directory schema including descriptions. Publish the OID registry with the custom schema.
4. Define the new attributes.
5. Define the object classes that contain the new attributes. However, never update the default schema. If you create new attributes, always add them to a custom object class.

Directory Server loads the schema when the instance starts. To load new schema files, restart the instance or initiate a reload task.

Keep the following rules in mind when customizing the Directory Server schema:

- Keep the schema as simple as possible.
- Reuse existing schema elements whenever possible.
- Minimize the number of mandatory attributes defined for each object class.
- Do not define more than one object class or attribute for the same purpose.
- Do not modify any existing definitions of attributes or object classes.

WARNING

Do not update or delete the standard schema to avoid compatibility problems with other directories or LDAP client applications.

2.2. HOW DIRECTORY SERVER MANAGES SCHEMA UPDATES IN A REPLICATION ENVIRONMENT

When you update the directory schema in the **cn=schema** tree, Directory Server stores the changes in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file, including a change state number (CSN).

Directory Server does not directly replicate the schema changes to other replicas. Schema replication starts when directory content is updated in the replicated tree. For example, if you update a user after modifying the schema, the supplier compares the CSN stored in the **nsSchemaCSN** attribute with the one on the consumer. If the value of the **nsSchemaCSN** attribute on the consumer is lower than the one on the supplier, Directory Server replicates the schema to the consumer. For a successful replication, all object classes and attribute types on the supplier must be a superset of the consumer's definition.

Example 2.1. Schema subsets and supersets

- On **server1**, the **example** object class allows the **a1**, **a2**, and **a3** attributes.
- On **server2**, the **example** object class allows the **a1** and **a3** attributes.

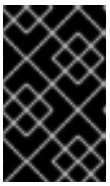
In the previous example, the schema definition of the **example** object class on **server1** is a superset of the object class on **server2**. During the validation phase, when Directory Server replicates or accepts the schema, the server retrieves the superset definitions. For example, if a consumer detects that an object class in the local schema allows less attributes than the object class in the supplier schema, Directory Server updates the local schema.

If the schema definitions are successfully replicated, the **nsSchemaCSN** attributes are identical on both servers and the schema definitions, such as object classes and attributes types, are no longer compared at the beginning of a replication session.

In the following scenarios, Directory Server does not replicate the schema:

- The schema on one host is a subset of the schema of another host.
  For example, the schema definition of the **example** object class on **server2** is a subset of the object class on **server1**. Subsets can also occur for attributes (a single-value attribute is a subset of a multi-value attribute) and attribute syntaxes.
- When definitions in supplier schema and consumer schema need to be merged.
- Directory Server does not support merging schemas. For example, if an object class on one server allows the **a1**, **a2**, and **a3** attributes and **a1**, **a3**, and **a4** on the other, the schemas are not subsets and cannot be merged.
- You use schema files other than **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif**. Directory Server enables you to add additional schema files to the **/etc/dirsrv/slapd-**

instance_name/schema/ directory. However, only the CSN in the **/etc/dirsrv/slapd-instance_name/schema/99user.ldif** file is updated. For this reasons, other schema file are used only locally and not automatically transferred to replication partners.



IMPORTANT

To enable Directory Server to automatically replicate the schema and to avoid duplicate schema definitions, store the custom schema in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file.

2.3. USING THE WEB CONSOLE TO CREATE A CUSTOM SCHEMA FOR AN ATTRIBUTE AND OBJECT CLASS

This procedure demonstrates how to use the web console to create a custom schema with:

- A single-valued attribute named **dateOfBirth** with OID **2.16.840.1.1133730.2.1.123** and syntax **Directory String** (OID **1.3.6.1.4.1.1466.115.121.1.15**)
- An object class named **exampleperson** without parent object class (**SUP top**), OID **2.16.840.1.1133730.2.1.99** that must contain the **dateOfBirth** attribute

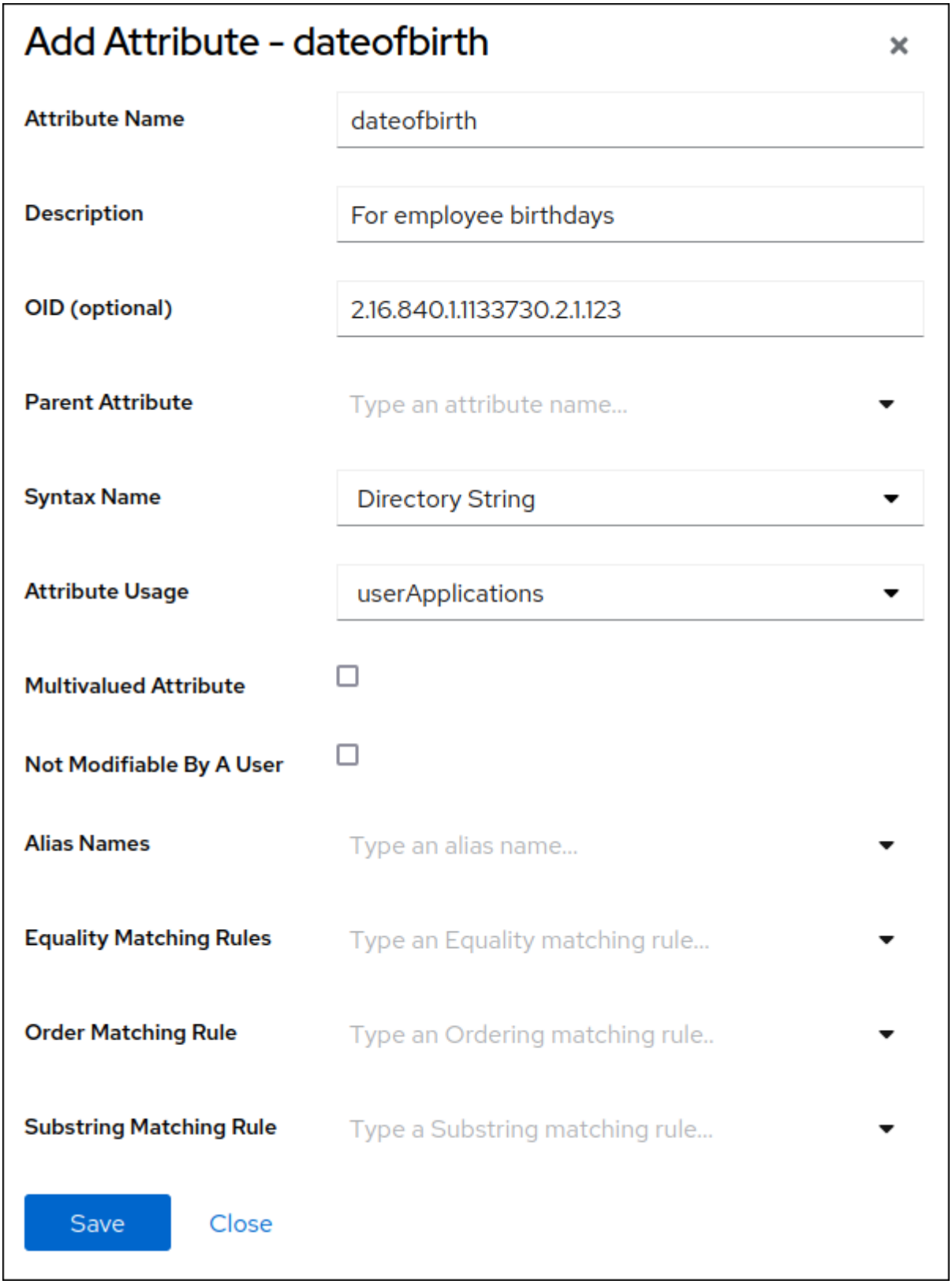
If you use the web console to update the schema, Directory Server automatically reloads the schema.

Prerequisites

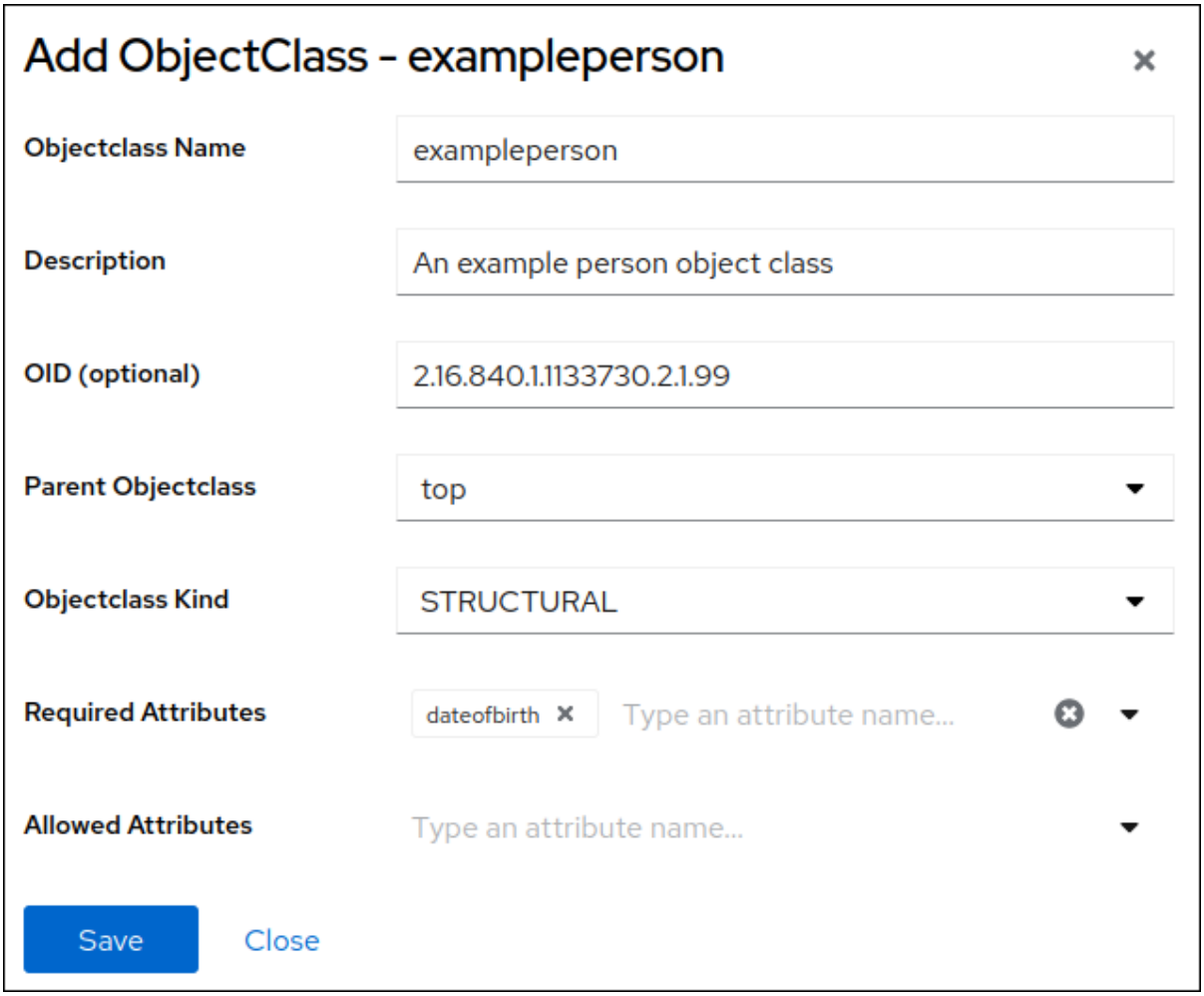
- You are logged in to the instance in the web console.

Procedure

1. Navigate to **Schema → Attributes**, and click **Add Attribute**.
2. Enter the settings of the attribute you want to add:



3. Click **Save**
4. Navigate to **Schema → Objectclasses**, and click **Add ObjectClass**.
5. Enter the settings of the object class you want to add:



6. Click **Save**

Verification

- Navigate to **Monitoring → Logging → Errors Log**.
  - If the build succeeds, Directory Server logs:

```
[23/Sep/2021:13:47:33.334241406 +0200] - INFO - schemareload - schemareload_thread - Schema reload task starts (schema dir: default) ...
[23/Sep/2021:13:47:33.415692558 +0200] - INFO - schemareload - schemareload_thread - Schema validation passed.
[23/Sep/2021:13:47:33.454768148 +0200] - INFO - schemareload - schemareload_thread - Schema reload task finished.
```

  - If the build fails, Directory Server logs which step failed and why.

CHAPTER 3. MANUALLY CREATING A CUSTOM SCHEMA FILE

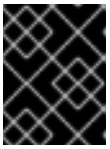
You can add custom attributes and object classes to Directory Server by extending the schema. You can extend the schema:

- Manually by creating schema files. The process is described in this section.
- Using the dsconf utility on the command line .
- Using the Directory Server web console .

3.1. WORKFLOW OF A SCHEMA EXTENSION

Adding new schema elements requires:

1. Planning and defining unique object identifiers (OID) for the new schema. Directory Server recognizes schema elements by their OID, but you must manage the OIDs manually.
   An OID is a dot-separated number that identifies the schema element to the server. OIDs can be hierarchical with a base OID that can be expanded to accommodate different branches. For example, the base OID could be **1**, and there can be a branch for attributes at **1.1** and for object classes at **1.2**.

   IMPORTANT

   Even if not required, Red Hat recommends to use numeric OIDs for custom schemas for better forward compatibility and performance.

2. Request OIDs from the Internet Assigned Numbers Authority (IANA). For details, see https://pen.iana.org/pen/PenApplication.page.
3. Create a OID registry to track OID assignments and to ensure that no OID is used for more than one purpose. An OID registry is a list of all OIDs used in the directory schema including descriptions. Publish the OID registry with the custom schema.
4. Define the new attributes.
5. Define the object classes that contain the new attributes. However, never update the default schema. If you create new attributes, always add them to a custom object class.

Directory Server loads the schema when the instance starts. To load new schema files, restart the instance or initiate a reload task.

Keep the following rules in mind when customizing the Directory Server schema:

- Keep the schema as simple as possible.
- Reuse existing schema elements whenever possible.
- Minimize the number of mandatory attributes defined for each object class.
- Do not define more than one object class or attribute for the same purpose.
- Do not modify any existing definitions of attributes or object classes.

WARNING

Do not update or delete the standard schema to avoid compatibility problems with other directories or LDAP client applications.

3.2. REQUIREMENTS FOR A SCHEMA FILE

Schema files use the LDIF format that define the **cn=schema** entry. Each attribute type and object class is added to this entry.

The following are the requirements for a schema file:

- The file must start with the following entry:

  ```
  dn: cn=schema
  ```

- A schema file can include attribute types or object classes or both of them.
- Object class definitions can use attributes defined in other schema files.
- Depending on which instances should use a custom schema file, store it in one of the following locations:
  - **/etc/dirsrv/slapd-*instance_name*/schema/** to make the schema file available to this specific instance
  - **/usr/share/dirsrv/schema/** to make the schema file available to all instances running on this host
- By default, Directory Server expects the custom schema in the **99user.ldif** file. If you use a different file name:
  - The name must be alphabetically lower than **99user.ldif**. For example, **99aaa.ldif** is ok, but **99zzz.ldif** is not.
  - The name must start with two digits and be higher than **01** because custom schema files must be loaded after the core schema files, which begin with **00** up to **98**
    Directory Server reads schema files in alphabetical order. Therefore, for example, if you store a definition **99user.ldif**, it will override definitions from standard files whose name begins with **00** and **01**.
- If you want to use a standard schema file from the **/usr/share/dirsrv/data/** directory, copy the file to **/etc/dirsrv/slapd-*instance_name*/schema/** or **/usr/share/dirsrv/schema/** depending on which instances should use the file. However, use a different file name in the destination directory. Otherwise, Directory Server renames the file during an upgrade and appends the **.bak** suffix.

Example 3.1. Example of a custom schema file

```
dn: cn=schema
objectClasses: ( 2.16.840.1.1133730.2.1.123 NAME 'exampleperson' DESC 'An example
```

```
  person object class' SUP top STRUCTURAL MUST dateOfBirth X-ORIGIN 'user defined' )
attributeTypes: ( 2.16.840.1.1133730.2.1.99 NAME 'dateOfBirth' DESC 'For employee
  birthday' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'user defined' )
```

3.3. THE DEFINITION OF ATTRIBUTES IN CUSTOM SCHEMA FILES

You define attributes in schema files as values of **attributeTypes** attributes.

Example 3.2. Definition of an attribute

```
attributeTypes: ( 2.16.840.1.1133730.2.1.123 NAME 'dateOfBirth' DESC 'For employee birthday'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'user defined' )
```

The attribute definition contains the following components:

- A unique object identifier (OID) specified as a dot-separated number.
- A unique name in the form of **NAME *attribute_name***.
- A description in the form of **DESC *description***.
- The OID for the syntax of the attribute values in the form **SYNTAX *OID***. For details about the LDAP attribute syntaxes, see RFC 4517.
- Optional: The source where the attribute is defined.

3.4. THE DEFINITION OF OBJECT CLASSES IN CUSTOM SCHEMA FILES

You define object classes in schema files as values of **objectClasses** attributes.

Example 3.3. Definition of an object class

```
objectClasses: ( 2.16.840.1.1133730.2.1.99 NAME 'exampleperson' DESC 'An example person
object class' SUP top STRUCTURAL MUST dateOfBirth X-ORIGIN 'user defined' )
```

The object class definition contains the following components:

- A unique object identifier (OID) specified as a dot-separated number.
- A unique name in the form of **NAME *attribute_name***.
- A description in the form of **DESC *description***.
- The superior (parent) object class for this object class in the form **SUP *object_class***. If there is no related parent, use **SUP top**.
- The word **STRUCTURAL** defines the type of entry to which the object class applies. Any entry must belong to at least one **STRUCTURAL** object class. **AUXILIARY** means that it can apply to any entry.

- A list of required attributes, preceded by the **MUST** keyword. To include multiple attributes, enclose the group in parentheses and separate the attributes with a [command]`$ ` (dollar sign and space).
- A list of optional attributes, preceded by the **MAY** keyword. To include multiple attributes, enclose the group in parentheses and separate the attributes with a [command]`$ ` (dollar sign and space).

Only the name and OID is required, and other settings depend on the needs of the object class.

Additional resources

- Section 4.2 in RFC 4512

3.5. HOW DIRECTORY SERVER MANAGES SCHEMA UPDATES IN A REPLICATION ENVIRONMENT

When you update the directory schema in the **cn=schema** tree, Directory Server stores the changes in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file, including a change state number (CSN).

Directory Server does not directly replicate the schema changes to other replicas. Schema replication starts when directory content is updated in the replicated tree. For example, if you update a user after modifying the schema, the supplier compares the CSN stored in the **nsSchemaCSN** attribute with the one on the consumer. If the value of the **nsSchemaCSN** attribute on the consumer is lower than the one on the supplier, Directory Server replicates the schema to the consumer. For a successful replication, all object classes and attribute types on the supplier must be a superset of the consumer's definition.

Example 3.4. Schema subsets and supersets

- On **server1**, the **example** object class allows the **a1**, **a2**, and **a3** attributes.
- On **server2**, the **example** object class allows the **a1** and **a3** attributes.

In the previous example, the schema definition of the **example** object class on **server1** is a superset of the object class on **server2**. During the validation phase, when Directory Server replicates or accepts the schema, the server retrieves the superset definitions. For example, if a consumer detects that an object class in the local schema allows less attributes than the object class in the supplier schema, Directory Server updates the local schema.

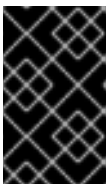
If the schema definitions are successfully replicated, the **nsSchemaCSN** attributes are identical on both servers and the schema definitions, such as object classes and attributes types, are no longer compared at the beginning of a replication session.

In the following scenarios, Directory Server does not replicate the schema:

- The schema on one host is a subset of the schema of another host.
  For example, the schema definition of the **example** object class on **server2** is a subset of the object class on **server1**. Subsets can also occur for attributes (a single-value attribute is a subset of a multi-value attribute) and attribute syntaxes.
- When definitions in supplier schema and consumer schema need to be merged.

- Directory Server does not support merging schemas. For example, if an object class on one server allows the **a1**, **a2**, and **a3** attributes and **a1**, **a3**, and **a4** on the other, the schemas are not subsets and cannot be merged.

- You use schema files other than **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif**. Directory Server enables you to add additional schema files to the **/etc/dirsrv/slapd-instance_name/schema/** directory. However, only the CSN in the **/etc/dirsrv/slapd-instance_name/schema/99user.ldif** file is updated. For this reasons, other schema file are used only locally and not automatically transferred to replication partners.



IMPORTANT

To enable Directory Server to automatically replicate the schema and to avoid duplicate schema definitions, store the custom schema in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file.

3.6. MANUALLY CREATING A CUSTOM SCHEMA FILE FOR AN ATTRIBUTE AND OBJECT CLASS

If you want to manually create a custom schema, store it in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file. Using a different file name is possible, but causes drawbacks, such as schema definitions stored in other files are replicated, but then stored in **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** on the replica. See How Directory Server manages schema updates in a replication environment.

This procedure adds:

- A single-valued attribute named **dateOfBirth** with OID **2.16.840.1.1133730.2.1.123** and syntax **Directory String** (OID **1.3.6.1.4.1.1466.115.121.1.15**)

- An object class named **exampleperson** without parent object class (**SUP top**) that must contain the **dateOfBirth** attribute.

Procedure

1. Add the following content below the **dn: cn=schema** entry in the **/etc/dirsrv/slapd-*instance_name*/schema/99user.ldif** file:

```
attributeTypes: ( 2.16.840.1.1133730.2.1.123 NAME 'dateOfBirth' DESC 'For employee
  birthday' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'user defined'
)
objectClasses: ( 2.16.840.1.1133730.2.1.99 NAME 'exampleperson' DESC 'An example
  person object class' SUP top STRUCTURAL MUST dateOfBirth X-ORIGIN 'user defined' )
```

2. Run a schema reload task:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com schema reload
```

Verification steps:

- Monitor the **/var/log/dirsrv/slapd-*instance_name*/errors** file:

  - If the build succeeds, Directory Server logs:

```
[23/Sep/2021:13:47:33.334241406 +0200] - INFO - schemareload -
schemareload_thread - Schema reload task starts (schema dir: default) ...
[23/Sep/2021:13:47:33.415692558 +0200] - INFO - schemareload -
schemareload_thread - Schema validation passed.
[23/Sep/2021:13:47:33.454768148 +0200] - INFO - schemareload -
schemareload_thread - Schema reload task finished.
```

- If the build fails, Directory Server logs which step failed and why.

CHAPTER 4. VALIDATING THE SYNTAX OF EXISTING ATTRIBUTE VALUES

With syntax validation, the Directory Server checks if an attribute value follows the rules of the syntax provided in the definition of that attribute. The Directory Server records the results of syntax validation tasks in the **/var/log/dirsrv/slapd-instance_name/errors** file.

Manual syntax validation is required if:

- You have the syntax validation disabled in the **nsslapd-syntaxcheck** parameter.

NOTE

Red Hat recommends that syntax validation should not be disabled.

- You migrate data from a server with disabled or without syntax validation.

4.1. CREATING A SYNTAX VALIDATION TASK USING THE DSCONF SCHEMA VALIDATE-SYNTAX COMMAND

With the **dsconf schema validate-syntax** command, you can create a syntax validation task to check every modified attribute and ensure that the new value has the required syntax.

Procedure

- To create a syntax validation task, enter:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com schema validate-syntax -f
'(objectclass=inetorgperson)' ou=People,dc=example,dc=com
```

In the example output, the command creates a task that validates the syntax of all values in the **ou=People,dc=example,dc=com** sub-tree which match the **(objectclass=inetorgperson)** filter.

4.2. CREATING A SYNTAX VALIDATION TASK USING ACN TASK ENTRY

The **cn=tasks,cn=config** entry in the Directory Server configuration is a container entry for temporary entries used by the server for managing tasks. You can initiate a syntax validation operation by creating a task in the **cn=syntax validate,cn=tasks,cn=config** entry.

Procedure

- To initiate a syntax validation operation, create a task in the **cn=syntax validate,cn=tasks,cn=config** entry as follows:

```
# ldapadd -D "cn=Directory Manager" -W -p 389 -H ldap://server.example.com -x

dn: cn=example_syntax_validate,cn=syntax validate,cn=tasks,cn=config
objectclass: extensibleObject
cn: cn=example_syntax_validate
basedn: ou=People,dc=example,dc=com
filter: (objectclass=inetorgperson)
```

In the example output, the command creates a task that validates the syntax of all values in the **ou=People,dc=example,dc=com** sub-tree that is similar to the **(objectclass=inetorgperson)** filter. When the task completes, Directory Server deletes the entry from the directory configuration.

Additional resources

- Configuration and schema reference