univention
be open.

App Tutorial

Building Apps for the Univention App Center

univention
be open.

Table of Contents

Chapter 1. Apps and Univention App Center

Univention App Center provides a platform for software vendors and an easy-to-use entry point for Univention Corporate Server (UCS) users to extend their IT environment with business software. The App Center is part of the web-based UCS management system and gives an overview of available and installed apps. Its purpose is to present available business applications for UCS and simplify their installation. This allows their easy evaluation and fosters the purchase decision.

Apps are the content of the App Center and they consist of the business software and some meta data about the presentation in the App Center. Most of them come with an integration into UCS, e.g. the management system or the mailstack. The purpose of an app is to provide the business solution in a way that it is ready to use after the installation and that comes with a decent default configuration to offer a satisfying impression of the solution. The installation is non-interactive and is done by just a click. Furthermore, an app utilizes the benefits of UCS and the business solution.

The App Center infrastructure consists of two parts: The already mentioned frontend as part of the web-based UCS management system and the server side component that stores the app meta data and the app software packages in their own respective repositories. The server side infrastructure is operated by Univention. The technological basis for installation and updates of the apps is APT, the well known advanced package tool from Debian. Therefore, the app needs to consist of so-called Debian packages. The App Center frontend is responsible for the app's presentation. As soon as an app is clicked to be installed or updated, the App Center activates the respective repository and the further process is handed over to apt which takes care of the rest like for example dependency resolution.

The next sections explain how to prepare your business solution as app for UCS. It also outlines the integration possibilities and describes what to do by example. Let's go!

Chapter 2. Prepare the environment

Before you can start with the creation of an app for Univention App Center, you'll need to prepare your UCS environment. This section guides you through the necessary steps.

2.1. Download

Feedback

First of all, get yourself a copy of UCS free of charge at the Univention Website [https://www.univention.com/downloads/ucs-download/]. You can choose between an ISO image or a pre-installed virtual machine.

2.2. Initial setup

Feedback

Please refer to the UCS Quickstart Guide [https://docs.software-univention.de/quickstart-en.html] for the steps about installation and initial setup.

2.3. Activate the unmaintained repository

Feedback

UCS is a Linux distribution derived from Debian GNU/Linux. It behaves very similar and therefore software is installed from software repositories. UCS comes with the same packages as Debian (except the packages from the games section). The packages are provided through two repositories: maintained and unmaintained. Only the maintained repository is always activated by default.

To install your solution, you may need packages that are in the unmaintained repository. Please activate it:

```
ucr set repository/online/unmaintained='yes'
```

Note

Please remember the packages you need from unmaintained repository and provide the list later with your upload. Univention will copy those packages besides your app packages and make sure that the package dependencies from the unmaintained repository are met without prior activation by the user.

2.4. Install required packages for App development

Feedback

To build your software on UCS you will need to install build tools for Debian packages. The corresponding package can be installed with

```
univention-install build-essential debhelper
```

Depending on your app you may furthermore require several development libraries (e.g. ***libc-dev***, ***php5-dev***). For UCS integration packages, we recommend

```
# ucslint checks for common mistakes in a variety of files if enabled
# in debian/rules
#  see https://docs.software-univention.de/developer-
reference.html#misc:ucslint
univention-install ucslint
```

```
# univention-config-dev takes care of installing and registering UCR
# variables if enabled in debian/rules
#   see https://docs.software-univention.de/developer-
reference.html#chap:ucr
univention-install univention-config-dev
```

```
# If you are developing a UMC module to extend the management console,
# you will need
univention-install univention-management-console-dev
```

If you already have a source directory with working code

```
dpkg-checkbuilddeps
```

should list the missing build dependencies, if any.

Chapter 3. Package the software solution

3.1. Create or use Debian packages

Feedback

You as ISV already distribute your software solution in a certain way. Univention App Center makes heavy use of the Debian package manager `dpkg` and the technology around it. Therefore, it is required that the software is provided in the Debian package format and that it can be installed non-interactively, e.g. the user will not be asked any questions for software configuration. This step has to be moved to a later step following the package installation.

Please follow this checklist:

1. If your software is provided via `.deb` files, you already have Debian packages. Please install those packages on UCS for testing purpose and evaluate if the software works as expected.

2. If your software is not provided via `.deb` files, Debian packages have to be created. Please follow the chapter packaging software [https://docs.software-univention.de/developer-reference.html#chap:packaging] in the UCS developer reference about how to create Debian packages.

3.2. Structure the App

Feedback

In most cases packages of an app for Univention App Center in principle consist of:

1. packages including the vanilla software solution of the ISV

2. packages with the integration of the software solution with Univention Corporate Server

For the ease of app maintenance it is recommended to provide the vanilla software from 1. in packages on their own, independent from UCS. This allows to theoretically use the packages on other Debian-based Linux Distributions like for example Debian GNU/Linux itself or Ubuntu.

The UCS specific part from 2. should be collected in a separate package. This package depends on the "main" package from 1. and therefore automatically installs all the other packages needed via the dependency resolution of the package manager.

Chapter 4. Integration with UCS

Univention Corporate Server (UCS) is not just an enterprise Linux distribution. With its focus on identity and infrastructure management it has a lot of information saved about the IT infrastructure environment, the user accounts and the groups and the system configuration, to name a few.

The most obvious integration makes use of the numerous user accounts stored in the UCS directory service. Apps using this information avoid double effort in user administration. They may technically make just a simple LDAP bind for user authentication. Or if the app needs certain user attributes in its own persistence layer (e.g. the database) they may be synchronised via the Listener-/Notifier mechanism. Furthermore, existing data can be extended with app specific attributes, e.g., shall a user be allowed to use the app or what role shall the user occupy for the app. The UCS management system can be extended by attributes and the information is usually stored in the directory service. It is even possible that certain values or their change may trigger certain actions. A third integration possibility is to hook up the app in existing solution stacks of UCS, for example the mail stack or the web server. The app will among others benefit from a working configuration and a higher communication security because of already present security certificates.

Those are just a few examples to outline the possibilities for the integration. There are many more. The guiding question for the integration should be: What information about the infrastructure, the configuration and identities does UCS offer that the app will benefit from and saves efforts for the administrator?

The following sections give an impression of several integration scenarios. Further information can be found in the UCS developer reference [https://docs.software-univention.de/developer-reference.html].

4.1. Read information from the directory service

Feedback

One primary element of the UCS management system is an LDAP directory in which the data required across the domain for the administration are stored. In addition to the user accounts and similar elements, the data basis of services such as DHCP is also saved there.

An LDAP directory has a tree-like structure, the root of which forms the so-called basis of the UCS domain. The UCS domain forms the common security and trust context for its members. An account in the LDAP directory establishes the membership in the UCS domain for users. Computers receive a computer account when they join the domain.

UCS utilises OpenLDAP as a directory service server. The directory is provided by the master domain controller and replicated on all domain controllers (DCs) in the domain. The complete LDAP directory is also replicated on a DC backup as this can replace the DC master in an emergency. In contrast, the replication on

DC slaves can be restricted to certain areas of the LDAP directory using ACLs (access control lists) in order to realize a selective replication.

The OpenLDAP server of UCS listens on port 7389 by default, not on 389. This is due to Samba 4 requiring port 389.

More information about the OpenLDAP server in UCS can be found in the manual [https://docs.software-univention.de/manual.html#domain:ldap].

4.1.1. Access the LDAP directory

Feedback

Sometimes software can use LDAP, but does not use the user accounts directly but is restricted to one specific user who then is used for further user authentication. This LDAP bind can be done by creating a app specific user in UDM. This should be done in a Join script via

```
ldap_base="$(ucr get ldap/base)"
APP='myapp'
PASSWORD='secret'
touch "/etc/$APP.secret"
chown root:root "/etc/$APP.secret" # or so
chmod 600 "/etc/$APP.secret"
printf '%s' "$PASSWORD" > "/etc/$APP.secret"
udm users/user create "$@" --position "cn=users,$ldap_base" \
 --set username="$APP-user" --set lastname="$APP-user" \
 --set password="$PASSWORD" --option ldap_pwd || die
```

Now you can configure your software accordingly. Here the DN will be `uid=`*`$APP-user`*`,cn=users,$ldap_base`.

If more access is needed, it is also possible to use the machine account of the UCS system. Every computer joined into the UCS domain has certain permissions. Computers in `cn=dc,cn=computers,$ldap_base` (by default DC Master, DC Backup, DC Slave) can even access the (hashed) password attributes of users and computers. The password for the machine account is stored in `/etc/machine.secret` (readable by `root`). The machine DN can be found by

```
ucr get ldap/hostdn
```

The machine password rotates. This means the password changes over time. If your software needs to adapt, you may install a script (with executable bit set!) at `/usr/lib/univention-server/server_password_change.d/`*`xx$app`* with *`xx`* being two digits for ordering purposes with something like the following content:

```
#!/bin/sh
case "$1" in
prechange)
 # nothing to do before the password is changed
 exit 0
 ;;
nochange)
 # nothing to do after a failed password change
 exit 0
 ;;
postchange)
 # do something with /etc/machine.secret, e.g.
 cp /etc/machine.secret /etc/$app.secret
 # restart daemon after password was changed
```

```
 invoke-rc.d $app restart
 ;;
esac
```

4.1.2. Listener-/Notifier Mechanism

Feedback

The data regarding identity and infrastructure management is saved in LDAP. Apps that are not LDAP-aware can use this data nonetheless by registering handlers that trigger when certain data is changed (e.g. a user is created, the IP of a computer is changed). This may be useful if

1. Your software contains some kind of user authentication/authorization, but cannot connect to LDAP
2. Your software has its own database and the data should be in sync
3. Your software needs to reconfigure as soon as certain parameters of the network topology change

More details can be found in the Developer Reference [https://docs.software-univention.de/developer-reference.html#chap:listener].

A short example how to sync first name, last name, email of a user to a (theoretical) third-party database. This script is run every time a user is added, removed or any of these attributes change. As the email is unique (forced by UCS) and all three attributes are only single-valued (also forced by UCS), this may come down to:

```
name = "app_sync_users"
description = "always be in sync with UCS users"
filter = "(&(uid=*)(!(uid=*$))"
attributes = ["givenName", "sn", "mailPrimaryAddress"]

def handler(dn, new, old):
 if new and not old:
  add_user(new)
 elif not new and old:
  remove_user(old)
 elif new and old:
  modify_user(new, old)

def add_user(new):
 new_mailPrimaryAddress = new.get('mailPrimaryAddress', [''])[0]
 new_givenName = new.get('givenName', [''])[0]
 new_sn = new.get('sn', [''])[0]
 get_db_connection().add(new_givenName, new_sn, new_mailPrimaryAddress)

def remove_user(old):
 old_mailPrimaryAddress = old.get('mailPrimaryAddress', [''])[0]
 get_db_connection().remove(old_mailPrimaryAddress)

def modify_user(new, old):
 old_mailPrimaryAddress = old.get('mailPrimaryAddress', [''])[0]
 new_mailPrimaryAddress = new.get('mailPrimaryAddress', [''])[0]
 new_givenName = new.get('givenName', [''])[0]
 new_sn = new.get('sn', [''])[0]
 get_db_connection().modify(old_mailPrimaryAddress, new_givenName,
  new_sn, new_mailPrimaryAddress)

def get_db_connection():
 raise NotImplementedError()
```

4.2. Read configuration database

Feedback

UCS ships with a key-value store used to save parameters of the environment, the Univention Configuration Registry (UCR). It holds information about the local host (like hostname or network settings) and to some extent about the domain configuration (like the domainname or where the DC Master can be found). More details can be found in the Developer Reference [https://docs.software-univention.de/developer-reference.html#chap:ucr].

The values can be accessed easily in a script by using

```
hostname=$(ucr get hostname)
```

Notable variables include:

- `hostname`
- `domainname`
- `ldap/base`
- `ldap/master` (FQDN of the DC Master)
- `ldap/master/port` (Port for LDAP bind)
- `ldap/hostdn` (May be useful to connect to LDAP with the machine account (password in `/etc/machine.secret`))

You can use the database to store your own keys and values and use those in your script:

```
ucr set myapp/loglevel=5
```

It is also possible to set the variable only if it was not set before. This is generally preferred as it allows users to overwrite those values without having to fear that it is overwritten again.

```
ucr set myapp/loglevel?4
```

You do not need to register those variables anywhere, they are just saved. It is also possible to use theses variables in your installations scripts as environment variables, for example:

```
eval "$(ucr shell)"
echo "This UCS system has the FQDN $hostname.$domainname "\
"and the LDAP base is $ldap_base."
```

A very powerful ability of UCR is its usage in templates. You may ship files that are recreated when certain variables change. For example, your app's configuration file needs to be updated every time a locale of the system is added or removed. Say your main package (***myapp.deb***) ships `/etc/myapp.conf`:

```
# configuration of myapp
title=My App
locales=en_US.UTF-8:UTF-8
```

Your integration package (***univention-myapp.deb***) can ship this file, too:

```
@%@UCRWARNING=#@%@
# configuration of myapp
title=My App
locales=@%@locale@%@
```

This file shall trigger each time

```
ucr set locale=...
```

is called.

You need to add the file above in ***univention-myapp***'s directory at `conffiles/etc/myapp.conf`. Furthermore you need to add the following in `debian/rules`:

```
override_dh_auto_install:
 univention-install-config-registry
 dh_auto_install

%:
 dh $@
```

Now you need to tell the system when to recreate it. For this, create a file `debian/univention-myapp.univention-config-registry` with

```
Type: file
File: etc/myapp.conf
Variables: locale
```

And this should do the trick. Templates can even use a Python runtime to do more than just writing the exact content of certain UCR variables. See the Developer Reference for details.

4.3. Domain-Join and Unjoin

Feedback

Integration into the UCS domain works by writing into the domain wide LDAP directory. The package can only change something in the LDAP directory through a join script, otherwise the functionality is not guaranteed. Furthermore, the hostname and other basic configuration settings are first defined when the join script is executed.

A join script is just an executable script living in `/usr/lib/univention-install/`. The name is something like *xx*`$app.inst` (*xx* are two digits for ordering purposes). The file must have the executable permission bits set.

Join scripts are commonly used to (but of course not limited to):

- Create users, groups, etc, as well as modifying existing ones
- Registering an LDAP schema extension
- Extending the form for creating/modifying a user (or a computer, ...) by Extended Attributes
- Adding a service entry to the local host
- Configuring the app with parameters read from LDAP

Join scripts are normally run as `root`.

This example shows how to register a schema extension as well as adding widgets to the user form.

```
root@master:~# cat /usr/lib/univention-install/50app.inst
#!/bin/bash

# VERSION has to be set for external programs to parse
```

```
# join scripts will in general onyl be run once per VERSION
# so you need to increment this value when you are changing the script
VERSION="1"

. /usr/share/univention-lib/ldap.sh
. /usr/share/univention-join/joinscripthelper.lib

# this function of joinscripthelper.lib initializes some important
# variables as well as aborting if this script has already been run
joinscript_init

eval "$(ucr shell)"
SERVICE="My App"
APP="app"

# "$@" is IMPORTANT, because this includes parameters for LDAP bind
# Otherwise these functions will fail on systems != DC master
# An example schema file is in the section "Extend the UCS management
# system"
ucs_registerLDAPExtension "$@" --schema "/usr/share/$APP/$APP.schema"

# create a container where the extended attributes shall live
udm container/cn create "$@" \
 --ignore_exists \
 --position "cn=custom attributes,cn=univention,$ldap_base" \
 --set name="$APP" || die # if this fails, abort join script

# for more details, see the section "Extend the UCS management system"
udm settings/extended_attribute create "$@" \
 --ignore_exists \
 --position "cn=$APP,cn=custom attributes,cn=univention,$ldap_base" \
 --set module="users/user" \
 `# ...` \
 --set name="$APP-enabled" || die

# Best practice: Adds the service to the host. Then LDAP can be queried
# to lookup where the app is already installed. Also useful for unjoin
ucs_addServiceToLocalhost "${SERVICE}" "$@"

# when everything worked fine, tell the system that this VERSION does
# not need to be run again
joinscript_save_current_version
exit 0
```

An unjoin script is more or less the same, except that it lives in `/usr/lib/univention-uninstall/` (and ends with `.uinst`). Its purpose is to be called after the app is uninstalled. After uninstallation, it might be appropriate to clean up those objects that have been added in the join script. Keep in mind that the app may be installed on different servers in the domain. So one must take care to not delete important objects when another host is still running this service.

```
root@master:~# cat /usr/lib/univention-uninstall/50app-uninstall.uinst
#!/bin/bash
VERSION=1
. /usr/share/univention-lib/ldap.sh
. /usr/share/univention-join/joinscripthelper.lib
```

```
joinscript_init

eval "$(ucr shell)"
SERVICE="My App"
APP="app"

# revert ucs_addServiceToLocalhost
ucs_removeServiceFromLocalhost "${SERVICE}" "$@"

# check whether this app is still installed elsewhere
if ucs_isServiceUnused "${SERVICE}" "$@"; then
 # revert other changes made by 50app.inst
 # just remove the container, the extended attribute is removed
 # automatically
 udm container/cn remove --dn \
  "cn=$APP,cn=custom attributes,cn=univention,$ldap_base"

 # DO NOT revert ucs_registerLDAPExtension "$@" --schema
 # schema extensions should be kept forever. If attributes defined
 # there were set during the time the app was installed
 # it may break LDAP if the attribute definition gets removed!
 # See http://sdb.univention.de/1274
fi

# revert joinscript_save_current_version - so that the join script
# would run again if the app is reinstalled
joinscript_remove_script_from_status_file app

exit 0
```

Now the scripts need to be packaged. Some steps have to be done in the `postinst`, `prerm`, `postrm` files of the package. There is a helper script that does that automatically. In `debian/rules`, add

```
override_dh_auto_install:
 univention-install-joinscript
 dh_auto_install

%:
 dh $@
```

The join script needs to lie in the root directory of the source code and has to be named after the package, e.g. `50univention-myapp.inst` and `50univention-myapp-uninstall.uinst`. If you need more control, just do not `univention-install-joinscript`, details what to do can be found in the Developer Reference [https://docs.software-univention.de/developer-reference.html#join:write].

4.4. Extend the UCS management system

Feedback

4.4.1. Add tabs and options

Feedback

The form for creating LDAP objects can be customized by apps. Technically this is done by writing special objects into LDAP. As such, customization is generally done in a join script. The objects are created with the Univention Directory Manager (UDM).

This example creates a checkbox in the users' form's tab "Advanced settings". This makes it possible to save whether the user should be allowed to use the app. The value has to be queried by the app afterwards.

```
APP="myapp"
SERVICE="My App"
# for more details, see
#   https://docs.software-univention.de/developer-reference.html#udm:ea
# "$@" is here because this should go into a join script and there
# passing the arguments of the script invocation to udm is necessary
udm settings/extended_attribute create "$@" \
 --ignore_exists \
 --position "cn=$APP,cn=custom attributes,cn=univention,$ldap_base" \
 --set module="users/user" `# extending users` \
 --set ldapMapping="${APP}Enabled" `# LDAP attribute from the schema` \
 --set objectClass="${APP}-user" \
 --set name="$APP-enabled" `# this is the name for UDM` \
 --set shortDescription="Allow $SERVICE" \
 --set longDescription="Whether this user shall be allowed ..." \
 --set translationShortDescription="\"de_DE\" \"$SERVICE erlauben\"" \
 --set translationLongDescription="\"de_DE\" \"Zeigt an, ob ...\"" \
 --set tabName="$SERVICE" `# This may create a new tab in the form` \
 --set translationTabName="\"de_DE\" \"$SERVICE\"" \
 --set tabAdvanced='0' \
 --set tabPosition='1' \
 --set syntax='TrueFalseUp' `# should be a CheckBox` \
 --set mayChange='1' \
 --set default='TRUE' || die
```

Note the

```
--set syntax='TrueFalseUp'
```

which semantically turns this attribute into a boolean field. Other syntax definitions exist, for example `string` or `ipAddress`. More examples can be found in the following file `/usr/share/pyshared/univention/admin/syntax.py`.

It is also possible to create own drop downs. The following example adds a combo box with two options "`Admin`" or "`User`"

```
udm settings/extended_attribute create "$@" \
 --ignore_exists \
 --position "cn=$APP,cn=custom attributes,cn=univention,$ldap_base" \
 --set module="users/user" \
 --set ldapMapping="${APP}Role" \
 --set objectClass="${APP}-user" \
 --set name="$APP-role" \
 --set shortDescription="Role in $SERVICE" \
 --set longDescription="Which role the user has for $SERVICE" \
 --set translationShortDescription="\"de_DE\" \"$SERVICE-Rolle\"" \
 --set translationLongDescription="\"de_DE\" \"Welche Rolle ...\"" \
 --set tabName="$SERVICE" \
 --set translationTabName="\"de_DE\" \"$SERVICE\"" \
 --set tabAdvanced='1' \
 --set tabPosition='1' \
 --set syntax="${APP}UserOrAdmin" \
 --set mayChange='1' \
 --set default='user' || die
```

The syntax is a Python class and needs to be defined in a separate file:

```
class myappUserOrAdmin(select):
 choices=[('user', 'User'), ('admin', 'Admin')]
```

This file needs to be registered in a join script:

```
ucs_registerLDAPExtension "$@" \
 --udm_syntax "/usr/share/$APP/${APP}_syntax.py"
```

4.4.2. LDAP schema extension

Feedback

The Extended Attributes are generally stored in LDAP as attributes not defined by default. A schema file needs to be created and registered for the Extended Attributes to actually work. See this section [http://www.openldap.org/doc/admin23/schema.html] for details of how to write a schema file.

The example above needs a file like this:

```
attributetype ( 1.3.6.1.4.1.10176.99998.xxx.1.1 NAME 'myapp-enabled'
 DESC 'My App allowed'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE
 )
attributetype ( 1.3.6.1.4.1.10176.99998.xxx.1.2 NAME 'myapp-role'
 DESC 'My App role'
 EQUALITY caseIgnoreMatch
 SUBSTR caseIgnoreSubstringsMatch
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE
 )
objectclass ( 1.3.6.1.4.1.10176.99998.xxx.0.1 NAME 'myapp-user'
 DESC 'My App user'
 SUP top AUXILIARY
 MUST ( cn )
 MAY ( myapp-enabled $ myapp-role )
 )
```

Note the "*xxx*" in the so called OIDs. You need a unique (worldwide!) identifier for your attributes and object classes. Either request one [http://pen.iana.org/pen/PenApplication.page] or (probably better) talk to us, as Univention has its own namespace and any schema extension can be defined "beneath" it.

This file needs to be registered in a join script:

```
ucs_registerLDAPExtension "$@" --schema "/usr/share/$APP/$APP.schema"
```

4.4.3. Build own UDM modules

Feedback

The Univention Directory Manager (UDM) is a collection of modules written in Python to add powerful capabilities around Python's LDAP bindings. In general, vendors will extend those existing modules using Extended Attributes. But if a completely new class of object shall be saved, a new UDM module may be useful, e.g., if the app manages buildings which cannot be simply "extended groups or containers".

When writing a UDM module, it is best to look out for existing modules that can be copied and customized for one's own needs. They live in `/usr/share/pyshared/univention/admin/handlers/`. The module `appcenter/app.py` is a rather simple module which shows how a basic module should look like:

```
svn co http://forge.univention.org/svn/dev/branches/ucs-4.0/ucs-4.0-0/\
management/univention-management-console-module-appcenter/\
```

```
udm/handlers/appcenter/
```

4.4.4. Build own UMC module

Feedback

The Univention Management Console (UMC) is the web based frontend which is used to administrate the UCS domain. It consists of separate modules and vendors may write such modules to further extend the Console. This may be a good idea if

- The app can be customized but currently lacks a frontend
- The app needs to be activated or manually configured to work properly

Note that many UCS users are used to UMC and the fact that everything can be configured in one place. So adding a UMC module may greatly enhance the user experience.

UMC modules are very versatile (both the JavaScript based frontend part as well as the Python backend part) and can be used for nearly anything. This guide cannot cover everything there is about UMC modules. One starting point may be

```
univention-install univention-management-console-dev
umc-create-module --help
```

Or you use UMC and look out for modules that to some extend do what you are trying to accomplish and copy the source code, e.g.

```
svn co http://forge.univention.org/svn/dev/branches/ucs-4.0/ucs-4.0-0/\
/management/univention-management-console-module-top
```

4.5. Further integration scenarios

Feedback

4.5.1. Firewall settings

Feedback

In the default setting, all incoming ports are blocked by the UCS firewall. Every package can provide rules, which free up the ports required. In this example the port 6644 is opened for TCP and UDP. It can be run in the `postinst` script or in the join script:

```
# configure firewall
univention-config-registry set \
 security/packetfilter/package/"$APP"/tcp/6644/all="ACCEPT" \
 security/packetfilter/package/"$APP"/tcp/6644/all/en="$APP" \
 security/packetfilter/package/"$APP"/udp/6644/all="ACCEPT" \
 security/packetfilter/package/"$APP"/udp/6644/all/en="$APP"
[ -x "/etc/init.d/univention-firewall" ] &&
 invoke-rc.d univention-firewall restart
```

4.5.2. Serving a web application

Feedback

UCS comes with a running Apache HTTP Server used by the UMC server as a proxy. This means that apps cannot use port 80/443 easily: It is already used. Apps can use Apache, though, by shipping a file `/etc/apache2/sites-available/`*`$APP`*. Apache can then act as a proxy to the app's server (running on a different port).

```
# minimal
ProxyPass /$APP/ http://127.0.0.1:$APP_PORT/
ProxyPassReverse /$APP/ http://127.0.0.1:$APP_PORT/
```

The site needs to be enabled by a line in the `postinst` of the package:

```
a2ensite "$APP"
```

It is highly recommended to use Apache for it is the service with port 80/443. While it is possible to just let the app respond to requests on port say, 8080, many firewalls will block the app without taking further actions. One prominent example are the default security rules of the Amazon Web Services. The app may not be accessible without using Apache as a proxy!

4.5.3. Setting links to the web interface in `/ucs-overview`

The start page of any UCS system (`http://$hostname/`) lists available services on this server, notably UMC. If an app provides a web interface, this can be listed, too. The easiest way is by stating this in the `ini` file:

```
WebInterface=/$APP/
#WebInterfaceName=... # defaults to Name=
# one of the two categories in /ucs-overview.
#   "services" (default) or "admin"
#UCSOverviewCategory=services
```

If *WebInterface* is given in the `ini`, the App Center takes care of the integration on the overview site. But sometimes this is not powerful enough. This level of "automated integration" cannot handle ports other than 80/443 (as it will always use the current port which is 80 or 443) and cannot add more than one link. If a deeper level is required, this should be done in the `postinst` and `postrm` scripts of the integration package using UCR:

```
# postint

#DEBHELPER#

# ucs/web/overview/entries/service/... or
# ucs/web/overview/entries/admin/...

export P="ucs/web/overview/entries/service"
ucr set \
 "$P/$APP"/description/de="Description of link to $APP (German)" \
 "$P/$APP"/description="Description of link to $APP (English)" \
 "$P/$APP"/icon="/url/to/icon/$APP.png" \
 "$P/$APP"/label/de="Headline of link to $APP (German)" \
 "$P/$APP"/label="Headline of link to $APP (English)" \
 "$P/$APP"/link="https://$hostname.$domainname:$APP_PORT/webinterface/"
 \
 "$P/$APP"/priority=xx-digits-for-sorting-or-just-dont-set
```

```
# postrm

#DEBHELPER#

ucr unset \
 ucs/web/overview/entries/service/"$APP"/description/de \
 ucs/web/overview/entries/service/"$APP"/description \
 ucs/web/overview/entries/service/"$APP"/icon \
 ucs/web/overview/entries/service/"$APP"/label/de \
 ucs/web/overview/entries/service/"$APP"/label \
 ucs/web/overview/entries/service/"$APP"/link \
```

```
ucs/web/overview/entries/service/"$APP"/priority
```

4.5.4. Using PostgreSQL or MySQL

Feedback

When your application uses PostgreSQL, your package should depend on ***univention-postgres*** and you need to ship a file in `/etc/postgresql/9.1/main/pg_hba.conf.d/` or, maybe even better, in `/etc/univention/templates/files/etc/postgresql/9.1/main/pg_hba.conf.d/` (see UCR):

```
local $app_db_name $app_db_user md5
```

When your application uses MySQL, you may access the admin password by reading `/etc/mysql.secret`. A dependency on the package ***mysql-server*** is enough as we patch the Debian package.

Chapter 5. Provide the App

Until now you should have your software solution packaged as Debian package(s) including a separate package taking care of the UCS integration. To finish the app, you'll need to add the meta data for the App Center and upload it to Univention.

Note

Starting with UCS 4.0, only 64 bit installation images are provided. Univention does support 32 bit for at least UCS 4.*x*, though. When using a 32 bit UCS 3.2, one may update to UCS 4.0. It is therefore recommended (but not required) to provide the app for amd64 and i386. If i386 shall not be supported, one may specify `SupportedArchitectures=amd64` in the ini file, see Section 5.1.

For the archive to be uploaded, the following directory structure is recommended:

- `metadata/`
- `packages/`
  - `all/`
  - `amd64/`
  - `i386/`
- `readme`

Put your packages in the appropriate subdirectories below `packages/`.

5.1. Create the App meta data

The Debian packages take care of the installation of your software solution on UCS through the Debian package manager. But, the App Center does not know what to present to the user. This gap is filled with the App meta data comprising of text information like description, website, contact, visual information like a logo and optional screenshots, optional detailed information for the users in several readme files.

Please provide the following information together with packages:

1. A text file in the `.ini` format including information like description, several website links, contact information, conflicting apps, etc. Please refer to the Developer Reference [https://docs.software-univention.de/developer-reference.html#app:iniFile] for a template and the description of every attribute.

2. A product logo in PNG format, size: 50x50 pixels, with transparent background.

The `.ini` file has the attribute *ID*. Simply name the `.ini` file and the product logo after that ID:

```
myapp.ini
myapp.png
```

Put those files below the `metadata/` directory.

Note

The meta data contains the attributes *NotifyVendor* and *NotificationEmail*. If you want to receive daily email notifications upon the installation of your app, please set them appropriately. The email address here may differ from the contact address. If set to `True` the user will be informed about the delivery of such a notification before installing the app.

Note

Note to sales: You as independent software vendor are responsible for the contacts and it is up to you how to organize the follow-up. Try to contact the users very fast. The experience shows that it makes sense to organize a follow up within one week.

A detailed explanation about the notifications can be found in the Developer Reference [https://docs.software-univention.de/developer-reference.html#app:notification].

5.2. Create optional App meta information

Feedback

You may add optional app meta data information. Please refer to the Optional application files section in the developer reference for the choice of files.

1. Screenshot of your solution: The filename of the screenshot is given in the app meta data with the attribute *Screenshot*. Recommended name: *myapp*`_screenshot.png` (you may also provide a `jpg`).

2. License agreement: If you require the user to confirm a license agreement before installation, provide the file `LICENSE_AGREEMENT_EN` and `LICENSE_AGREEMENT_DE`.

3. `README` files: You may add different readme files depending on action taking place. For example, you may present text before installation or before update only. Please refer to the Optional application files section [https://docs.software-univention.de/developer-reference.html#app:optionalFiles] for the set of possible readme files.

Please use simple HTML in all those files and split the text into reasonable paragraphs. Copy the files below the readme directory in the recommended structure. The file names have to match the definitions.

5.3. Upload the App

Feedback

Finally, upload the whole app according to the following steps:

1. Take the above directory structure, create an archive, for example `tar.gz` or `zip`.

2. Upload the archive to https://upload.univention.de/ and remember the upload-id shown there.

3. Send the upload-id via email to `<appcenter@univention.de>`.

Congratulations! That's it, you are finished for the moment.

Chapter 6. What happens next?

After you sent the upload-id to Univention, a Univention employee will extract your files and copy them to the Test App Center. You'll receive a short note about how to activate the Test App Center for final testing of your app. After the app passed the automatic tests at Univention concerning the packages and you as vendor gave your written approval, the app will be published in the App Center. Due to feedback and further communication the approval process may last a couple of days. Further uploads may be necessary to fix issue that have been found during this iterating process.

Note

The automatic tests run after the packages have been copied into the Test App Center, only cover UCS core functionality, e.g., whether LDAP still works after adding a schema file. They do not test the App itself.