

# Additional features

- Custom Applet structure for RemoteApp

# Custom Applet structure for RemoteApp

## What is an applet?

An applet is a set of files that describe the process of installing and launching an application on Microsoft RDS via RemoteApp. This is necessary for JumpServer to initiate an access session to this application, automatically log in, and hide the authorization parameters from the user.

## Applet Structure

Each applet must include the following files:

```
AppletName
├── i18n.yml
├── icon.png
├── main.py
├── manifest.yml
└── setup.yml
```

`main.py` - script for launching and logging into the application

`icon.png` - applet icon

`manifest.yml` - metadata, i.e., applet description

`setup.yml` - file describing the installation process

`i18n.yml` - file for translation into various languages

## File manifest.yml

Example based on the MySQL Workbench applet

The file `manifest.yml` contains general information about the applet and specifies its type and protocol.

```
# (required)
name: mysql_workbench8
display_name: "{{ 'MySQL Workbench' | trans }}"
comment: "{{ 'A tool for working with MySQL, to execute SQL and design tables' | trans }}"
# (required)
version: 0.1.1
# (required)
exec_type: python
```

```
# (required)
author: Eric
# general or web (required)
type: general
update_policy: always
edition: community
# (required)
tags:
  - database
# (required)
protocols:
  - mysqlworkbench

# translations into other languages
i18n:
  MySQL Workbench:
    en: MySQL Workbench
    zh: MySQL Workbench
    ja: MySQL Workbench
```

```
A tool for working with MySQL, to execute SQL and design tables:
en: A tool for working with MySQL, to execute SQL and design tables
zh: MySQL SQL
ja: MySQL SQL
```

## File setup.yml

The file setup.yml describes the parameters for installing the applet on the RDS server.

```
# software installation type - msi, exe, zip, manual
type: msi
# URL to download the software distribution or file name if the distribution is included with the applet
# archive
source: mysql-workbench-community-8.0.31-winx64.msi
# installation arguments
arguments:
  - /qn
  - /norestart
# installation directory
destination: C:\Program Files\MySQL\MySQL Workbench 8.0 CE
# path and name of the executable file
program: C:\Program Files\MySQL\MySQL Workbench 8.0 CE\MySQLWorkbench.exe
md5: d628190252133c06dad399657666974a
```

## Script main.py

**main.py** - the main script of the applet

## The application is launched by running the command:

```
python main.py base64_json_data
```

That is, the main.py script is launched, and the launch parameters are passed to it. The **base64\_json\_data** structure looks approximately as follows:

```
{
  "app_name": "mysql_workbench8",
  "protocol": "mysql",
  "user": {
    "id": "2647CA35-5CAD-4DDF-8A88-6BD88F39BB30",
    "name": "Administrator",
    "username": "admin"
  },
  "asset": {
    "asset_id": "46EE5F50-F1C1-468C-97EE-560E3436754C",
    "asset_name": "test_mysql",
    "address": "192.168.1.1",
    "protocols": [
      {
        "id": 2,
        "name": "mysql",
        "port": 3306
      }
    ]
  },
  "account": {
    "account_id": "9D5585DE-5132-458C-AABE-89A83C112A83",
    "username": "root",
    "secret": "test"
  },
  "platform": {
    "charset": "UTF-8"
  }
}
```

## Contents of main.py

```
import sys

from common import (block_input, unblock_input) # Import functions for blocking/unblocking input
from common import convert_base64_to_dict # Import function to convert Base64 string to a dictionary (array)
from app import AppletApplication # Import main application

def main():
    base64_str = sys.argv[1] # Get the Base64 string from command-line arguments
    data = convert_base64_to_dict(base64_str) # Convert Base64 string to a dictionary
    # The data dictionary contains all the parameters for launching the application: account, server
```

```

name, database name, etc., depending on the application type
    applet_app = AppletApplication(**data) # Pass dictionary data to the application launch function
    block_input() # Block user input
    applet_app.run() # Launch the application
    unblock_input() # Unblock user input
    applet_app.wait() # Wait for the application to complete

if __name__ == '__main__':
    try:
        main() # Launch the main function
    except Exception as e:
        print(e) # Output the error if it occurs

```

## Contents of app.py

App.py typically contains all the main code for launching the application with the required parameters, making it the most important and complex part when developing a new applet. It is easier to base it on one of the scripts of existing applets that are similar in structure/type to the new applet being developed.

```

import sys # Imports the sys module for working with system functions

if sys.platform == 'win32': # Checks if the operating system is Windows
    from pywinauto import Application # Imports the library for automating Windows GUI applications
    from pywinauto.controls.uia_controls import (
        ButtonWrapper, EditWrapper, MenuItemWrapper,
        MenuWrapper, ComboBoxWrapper, ToolbarWrapper
    )
    # Imports various controls for interacting with the application's GUI

from common import BaseApplication, wait_pid # Imports the base application class and a function
for waiting on processes

_default_path = r"C:\Program Files\MySQL\MySQL Workbench 8.0 CE\MySQLWorkbench.exe"
# Defines the default path to the MySQL Workbench application

class AppletApplication(BaseApplication): # Defines the application class inheriting from
BaseApplication

    def __init__(self, *args, **kwargs): # Initializes the application
        super().__init__(*args, **kwargs) # Calls the parent class constructor

        self.path = _default_path # Sets the application's path
        self.username = self.account.username # Retrieves the username from the account information
        self.password = self.account.secret # Retrieves the password from the account information
        self.host = self.asset.address # Retrieves the host address from the asset information
        self.port = self.asset.get_protocol_port(self.protocol) # Retrieves the port based on the protocol
        self.db = self.asset.spec_info.db_name # Retrieves the database name

```

```

self.pid = None # Placeholder for the application's process ID
self.app = None # Placeholder for the application object

def run(self): # Method to run the application
    app = Application(backend='uia') # Creates an application object using UI Automation
    app.start(self.path) # Starts the application using the specified path
    self.pid = app.process # Saves the application's process ID
    if not all([self.username, self.password, self.host]): # Checks if necessary parameters are
provided
        print('Missing required parameters') # Outputs an error message in Chinese ("Missing required parameters")
        return

    # Accesses the main MySQL Workbench window and the "Database" menu
    menubar = app.window(title="MySQL Workbench", auto_id="MainForm",
control_type="Window") \
        .child_window(title="Database", control_type="MenuItem")
    menubar.wait('ready', timeout=10, retry_interval=5) # Waits for the menu to be ready
    MenuItemWrapper(menubar.element_info).select() # Opens the "Database" menu

    # Selects the "Connect to Database" menu item
    cdb = menubar.child_window(title="Connect to Database", control_type="MenuItem")
    cdb.wait('ready', timeout=10, retry_interval=5) # Waits for the item to be ready
    MenuItemWrapper(cdb.element_info).click_input() # Clicks on "Connect to Database"

    # Inputs the host
    host_ele = app.top_window().child_window(title="Host Name", auto_id="Host Name",
control_type="Edit")
    EditWrapper(host_ele.element_info).set_edit_text(self.host) # Sets the host value

    # Inputs the port
    port_ele = app.top_window().child_window(title="Port", auto_id="Port", control_type="Edit")
    EditWrapper(port_ele.element_info).set_edit_text(self.port) # Sets the port value

    # Inputs the username
    user_ele = app.top_window().child_window(title="User Name", auto_id="User Name",
control_type="Edit")
    EditWrapper(user_ele.element_info).set_edit_text(self.username) # Sets the username value

    # Inputs the database name
    db_ele = app.top_window().child_window(title="Default Schema", auto_id="Default Schema",
control_type="Edit")
    EditWrapper(db_ele.element_info).set_edit_text(self.db) # Sets the database name

    # Clicks the "OK" button to confirm the connection
    ok_ele = app.top_window().child_window(title="Connection", auto_id="Connection",
control_type="Window") \
        .child_window(title="OK", control_type="Button")
    ButtonWrapper(ok_ele.element_info).click() # Clicks "OK"

    # Inputs the password

```

```
password_ele = app.top_window().child_window(title="Password", auto_id="Password",
control_type="Edit")
password_ele.wait('ready', timeout=10, retry_interval=5) # Waits for the password field to be
ready
EditWrapper(password_ele.element_info).set_edit_text(self.password) # Sets the password value

# Clicks "OK" to complete the connection
ok_ele = app.top_window().child_window(title="Button Bar", auto_id="Button Bar",
control_type="Pane") \
    .child_window(title="OK", control_type="Button")
ButtonWrapper(ok_ele.element_info).click() # Clicks "OK"

self.app = app # Saves the application object for further use

def wait(self): # Method to wait for the application to complete
    wait_pid(self.pid) # Waits for the process with the specified ID to complete
```