Плагин Manage Applied Projects для управления инстансами и прикладными проектами DirectumRX

Введение

- **Инстанс RX** - именованный набор сервисов RX и пула и сайта в IIS.
- **Прикладной проект** - набор, состоящий из конкретного набора исходного кода (два и более репозиториев), базы данных и хранилища документов.

В практической деятельности прикладному разработчику приходится работать в следующих условиях:

- в оперативном доступе нужны несколько разных версий Directum RX, а также иметь возможность оперативно развернуть конкретный билд DirectumRX;
- если в тиражируемых решениях приходится разрабатывать SQL-запросы, то нужна возможность быстро переключаться между Microsoft SQL Server и PostgreeSQL;
- даже работая с одной версией Directum RX, нужна возможность оперативного переключения между разными прикладными проектами. Например, чтобы посмотреть/прорецензировать изменения из соседней ветке, которую не хотелось бы публиковать в БД текущего проекта;
- иногда возникает необходимость создать новый проект и хорошо бы это сделать без переустановки Directum RX;
- и т.д.

Типовой подход решения подобных ситуаций - создавать виртуальные машины под каждую версию Directum RX или даже под каждый прикладной проект. Такой подход не только требует существенных серверных ресурсов под виртуальные машины, но и довольно неудобен в практической работе. Например, на каждой виртуальной машине приходится устанавливать и поддерживать в актуальном состоянии необходимый инструментарий.

Штатный инсталлятор Directum RX заточен на установку первого и единственного инстанса Directum RX. Начиная с версии 4.2 DirectumLauncher позволяет:

- установить несколько копий Directum RX на одну машину;
- относительно простыми действиями создавать новые прикладные проекты и переключаться между ними.

Однако:

- для этого необходимо знать еще больше особенностей работы DirectumLauncher (многие из которых не документированы), а также в правильной последовательности выполнять нужные команды. В результате сложность таких действий и вероятность ошибки достаточно велики;
- установка Directum RX требует заполнения довольно значительного количества параметров, что требует заметного времени;
- каждый проект как минимум требует репозитория с исходниками стандартной прикладной разработки, а это 700-800 Mb.

Компонента Manage Applied Tools облегчает установку нескольких инстансов RX на одну машину, переключение установленных инстансов между разными прикладными проектами, а также упрощает ряд рутинных операций, с которыми сталкиваются прикладные разработчики. Происходит это за счет:

- использования в config.yml переменных, в которые выносятся значения, уникальные для инстансов или прикладных проектов;
- сохранения в специальных файлах значений, которые нужно вводить при установке инстанса;
- специальных скриптов, упрощающих установку нового инстанса Directum RX;
- сохранения в специальных файлах описания проектов значений переменных, уникальных для каждого прикладного проекта;
- реализации набор команд, позволяющих создавать, клонировать и менять прикладные проекты на основании описаний прикладных проектов;

Внимание! Компонента НЕ предназначена для управления продуктивными серверами.

Текущая версия совместима с Directum RX 4.2-4.8.

Демонстрация установки Directum RX с использованием Manage Applied Projects - см. Rutube, Youtube.

Подготовка к установке инстансов Directum RX

Использование компоненты предполагает, что:

1. Каждый инстанс Directum RX будет иметь своё имя
2. Все инстансы Directum RX будут устанавливаться внутри одного корневого каталога всех инстансов
3. Для всех инстансов будет использоваться один и тот же сертификат, который будет размещен в каком-то одном каталоге. Самое удобное место - корневой каталог инстансов.
4. Каталоги логов инстансов будут располагаться внутри корневого каталога
5. Базы данных проектов инстанса располагаются на одном сервере баз данных
6. Домашние каталоги проектов инстанса лежат внутри корневого каталога домашних каталогов инстанса (для каждого инстанса - свой каталог)
7. Репозитории с исходниками проектов инстанса лежат внутри корневого каталога исходников (для каждого инстанса - свой каталог)
8. Для каждого проекта будет создан конфиг проекта (очень упрощенная версия config.yml) и все конфиги будут располагаться внутри корневого каталога конфигов всех проектов

Подготовка к установке инстансов выполняется один раз для каждого рабочего места:

1. В Powershell, запущенном от имени администратора выполните команду `Install-Module powershell-yaml` (установка вспомогательной библиотеки), после успешной установки окно Powershell можно будет закрыть.
2. Установите Directum RX штатным инсталлятором. После установки скопируйте config.yml и папку с сертификатом, который используют сервисы для общения друг с другом. После этого Directum RX можно удалить - см. раздел "Удаление инстанса".
3. Подготовьте каталоги:

- корневой каталог инстансов. Например, `c:\rx_ver`. Внутри этого каталога будут создаваться каталоги для устанавливаемых инстансов;
- корневой каталог логов. Например, `c:\rx_logs`. Внутри этого каталога будут создаваться каталоги с логами каждого инстанса.
- создайте корневой каталог проектов. Например, `c:\rx`. В этом каталоге будем создавать файлы с описаниями прикладных проектов, а также каталоги для хранения исходников и данных прикладных проектов.

4. В корневом каталоге инстансов создайте каталог для хранения сертификата сервисов. Например, c:\rx_ver\data_protection. Скопируйте в каталог pfx- и cer-файлы сертификата, сохраненные на шаге №1.
5. Клонируйте репозиторий компоненты Manage Applied Tools
6. Скопируйте файлы update_config_before_install.yml и update_config_after_install.yml из каталога компоненты и отредактируйте их. Эти файлы будут использоваться в дальнейшем для корректировки config.yml устанавливаемых инстансов. Что следует учесть:
   - необходимо заполнить значения, отмеченные троеточием `...`
   - значения брать из config.yml, сохраненного на шаге №1.за исключением:
     - в свойстве DATA_PROTECTION_CERTIFICATE_FILE в update_config_before_install.yml укажите путь к pfx-файл, скопированному в каталог хранения сертификата сервисов
     - в параметре CONNECTION_STRING в update_config_before_install.yml параметр с именем базы данных (`initial catalog` для mssql и `database` для postgres) укажите БД, которая будет использоваться в ходе первоначальной установки инстанса. Например, `rx_install`. Эта БД будет удаляться и создаваться заново при установке каждого инстанса
     - в параметре home_path в update_config_before_install.yml указать каталог, который будет использоваться в качестве домашнего каталога во время установки инстанса. Например, `c:\rx\rx_install`.

Получится примерно так. Файл *update_config_before_install.yml*:

```
variables:
    host_fqdn: 'localhost'
    protocol: 'http'
    home_path: 'c:\rx\rx_install'
devstand_config: &devstand_config
    SAVE_NOCODE_SETTINGS_TO_SOURCES: 'true'
    DEV_STUDIO_CONFIG_PATH: '{{ instance_root_path }}\{{ instance_name
}}\etc\_builds\DevelopmentStudio\bin\_ConfigSettings.xml'
common_config:
    DATABASE_ENGINE: 'mssql'
    CONNECTION_STRING: 'data source=SQLSERVER;initial catalog=rx_install;user
id=sa;Password=sa_password'
    QUEUE_CONNECTION_STRING:
'virtualhost=rx;hostname=localhost;port=5672;username=admin;password=admin_passwor
d;Exchange=exchange_install'
    DATA_PROTECTION_CERTIFICATE_FILE: 'C:\rx_ver\data_protection\cert.pfx'
    DATA_PROTECTION_CERTIFICATE_FILE_PASSWORD: 'f4bc3790-11e4-45f2-b3de-
1acca662b8f4'
    MONGODB_CONNECTION_STRING: 'mongodb://admin:password@127.0.0.1:27017'
    AUTHENTICATION_USERNAME: 'Service User'
```

```
    AUTHENTICATION_PASSWORD: '11111'
services_config:
    DevelopmentStudio:
        COMPANY_CODE: 'OurCompany'
        UNIQUE_NAMES_IN_OVERRIDES: true
manage_applied_projects:
    postgresql_bin: 'C:\Program Files\PostgreSQL\14\bin'
    run_dds_after_set_project: 'True'
```

Файл *update_config_after_install.yml*:

```
variables:
    purpose: 'назначение проекта'
    database: 'база данных'
    home_path_src: 'корневой каталог исходников'
logs_path:
    LOGS_PATH: 'c:\rx_logs\{{ instance_name }}'
common_config:
    DATABASE_ENGINE: 'mssql'
    CONNECTION_STRING: 'data source=SQLSERVER;initial catalog={{ database }};user
id=sa;Password=sa_password'
    QUEUE_CONNECTION_STRING:
'virtualhost=rx;hostname=localhost;port=5672;username=admin;password=admin_passwor
d;Exchange=Exchange_{{ instance_name }}'
    WIDGETS_ORLEANS_SILO_MONGO_GRAIN_STORAGE_DATABASE_NAME: 'GRAIN_{{ database
}}'
    WIDGETS_ORLEANS_SILO_MONGO_CLUSTERING_DATABASE_NAME: 'CLUSTERING_{{ database
}}'
services_config:
    DevelopmentStudio:
        SERVICE_RUNNER_CONFIG_PATH: '{{ instance_root_path }}\{{ instance_name
}}\etc\_{{ instance_name }}\_services_config\ServiceRunner\_ConfigSettings.xml'
        GIT_ROOT_DIRECTORY: '{{ home_path_src }}'
```

Если необходимо иметь возможность устанавливать инстансы Directum RX на разные диски (например, для более эффективного использования дисков) или для работы с разными серверами БД, то необходимо сделать дополнительные пары файлов *update_config_before_install.yml* и *update_config_after_install.yml*.

Установка инстанса Directum RX

Установка первого и последующих инстансов Directum RX выполняется одинаково. Последовательность шагов:

1. Перед установкой инстанса:
    - выберите имя устанавливаемого инстанса. Как правило, в имени инстанса имеет смысл отразить версию Directum RX. Например, для установки Directum RX 4.5.30 имя инстанса можно сделать `4530`.
    - подберите свободный порт(ы) для будущего инстанса. Проверить свободен тот или иной порт можно командой

```
netstat -an | findstr /i :номер_порта
```

Если порт свободен, то команду ничего не выведет в консоль. Если занять, то в консоли будет примерно так:

```
  TCP    0.0.0.0:2086           0.0.0.0:0              LISTENING
  TCP    [::]:2086              [::]:0                 LISTENING
```

- убедитесь, что в корневом каталоге инстансов отсутствует каталог с именем, равным имени будущего инстанса. Например, `c:\rx_ver\4530`.

2. Запустите установку инстанса скриптом install_instance.ps1 передав ему все необходимые параметры. Например:

```
powershell D:\rx-manage-applied-
projects\src\ManageAppProjects\map_plugin\install_instance.ps1 ^
                  -rx_instaler_dir_path c:\distr\rx4530 ^
                  -instance_name 4530 ^
                  -port 1085 ^
                  -instance_root_dir_path c:\rx_ver ^
                  -map_plugin_path "D:\rx-manage-applied-
projects\src\ManageAppProjects\map_plugin" ^
                  -cfg_before_install_path
c:\rx_ver\update_config_before_install.yml ^
                  -cfg_after_install_path
c:\rx_ver\update_config_after_install.yml
```

Особенности установки:

- убедитесь, что в корневом каталоге инстансов отсутствует каталог с именем, равным имени будущего инстанса. Например, `c:\rx_ver\4530`
- по умолчанию компонента Redis не устанавливается. Но будет выполнена проверки наличия необходимых версий SDK и предложено самостоятельно принять решение о необходимости установки этой компоненты.
- для непосредственной установки Directum RX запускается DirectumLauncher с перезаполненным config.yml. Для установки необходимо переключить в режим "Установка"
- для корректного завершения установки необходимо закрыть окно с консолью, которое открывается при запуска DirectumLauncher
- устанавливается минимально необходимый для запуска DevelopmentStudio набор компонент. Точный состав зависит от устанавливаемой версии Directum RX.
- zip-файлы компонент не копируются в папку инстанса
- компонента WebHelp.zip не устанавливается- она занимает очень много места на диске, а справка доступна на club.directum.ru.
- сразу устанавливается компонента Manage Applied Project в варианте plugin (подробнее см. раздел "Установка компоненты Manage Applied Projects)

3. Создайте первый конфиг проекта. Рекомендации:

- в качестве префикса в имени конфига использовать имя инстанса - это позволит легко определять какие конфиги для какого инстанса созданы. Например, `4530_BoxOnly.yml`.

- первым рекомендуется создать проект, в котором будет только стандартная версия Directum RX. После её настройки - создания оргструктуры, создания пользователей и т.п., этот проект можно будет использовать как основу для созданию новых проектов.

Создать конфиг можно командой

```
do map generate_empty_project_config  c:\rx\4530_BoxOnly.yml
```

После чего его следует заполнить. Рекомендации:

- в корневом каталоге проектов создайте каталог с именем, равным именем инстанса. Например, `c:\rx\4530`.
- внутри созданного каталога создайте два каталога:
  - `d` - корневой каталог для домашних каталогов всех проектов инстанса
  - `s` - корневой каталог для репозиториев всех проектов инстанса
- в корневом каталоге для домашних проектов инстанса создайте каталог с именем проекта. Например, `c:\rx\4530\BoxOnly`.
- имя БД должно содержать как имя инстанса, так и имя проекта. Например, `rx4530_BoxOnly`.

Получится примерно так (имя конфига `4530_BoxOnly.yml`):

```
# ключевые параметры проекта
variables:
    # Назначение проекта
    purpose: 'Directum RX 4.5.30 (чистая коробка)'
    # БД проекта
    database: 'rx4530_BoxOnly'
    # Домашняя директория, относительно которой хранятся все данные сервисов.
    # Используется только в конфигурационном файле.
    home_path: 'c:\rx\4530\d\BoxOnly'
    # Корневой каталог с репозиториями проекта
    home_path_src: 'c:\rx\4530\s\'
# репозитории
services_config:
    DevelopmentStudio:
        REPOSITORIES:
            repository:
            -   '@folderName': 'BoxWork'
                '@solutionType': 'Work'
                '@url': ''
            -   '@folderName': 'Box'
                '@solutionType': 'Base'
                '@url': ''
```

4. Создайте проект:

```
do map create_project c:\rx\4530_BoxOnly.yml --
package_path=C:\rx_ver\4530\etc\_builds\DirectumRX\DirectumRXbase.dat --
need_import_src=True
```

В результате будет создан новый проект, в него будет принята стандартная прикладная разработка и приняты стандартные шаблоны документов.

Создание второго проекта в инстансе

Второй (третий, четвертый и так далее) проекты можно создавать также командой `do map create_project`. Но удобнее наполнить проект BoxOnly минимально необходимыми данными - создать оргструктуру, пользователей, учетные записи, а потом новые проекты создавать командой `do map clone_project` - см. соответствующий раздел.

При этом следует учитывать:

- рекомендуется в разных проектах переиспользовать каталоги с исходниками. В первую очередь это касается каталога Box с исходниками стандартной версии Directum RX. Это позволяет существенно экономить место на диске
- есть два подхода к подключению исходников к DevelopmentStudio - через подключение репозиториев и через импорт пакета разработки. По возможности стоит придерживаться варианта с подключением репозиториев. Если же требуется именно импортировать пакет разработки, то, в большинстве случаев, целесообразно создать отдельный каталог в корневом каталоге исходников, прописать его в конфиге проекта. При этом следует учитывать, что DevelopmentStudio импортирует пакет в первый по списку репозиторий подходящего типа.

Пример конфига проекта для изменения решения "Повторяющиеся поручения", при этом на базовом слое установлено не только стандартная прикладная разработка, но и решение "Прикладные константы". Имя конфига `4530_RecurringActionItems.yml`:

```
# ключевые параметры проекта
variables:
    # Назначение проекта
    purpose: 'Шаблон решения Повторяющиеся поручения'
    # БД проекта
    database: 'rx4530_RecurringActionItems'
    # Домашняя директория, относительно которой хранятся все данные сервисов.
    # Используется только в конфигурационном файле.
    home_path: 'c:\rx\4530\d\RecurringActionItems'
    # Корневой каталог с репозиториями проекта
    home_path_src: 'c:\rx\4530\s\'
# репозитории
services_config:
    DevelopmentStudio:
        REPOSITORIES:
            repository:
            -   '@folderName': 'rx-template-recurringactionitems'
                '@solutionType': 'Work'
                '@url': 'https://github.com/DirectumCompany/rx-template-
recurringactionitems'
            -   '@folderName': 'rx-template-settings'
                '@solutionType': 'Base'
                '@url': 'https://github.com/DirectumCompany/rx-template-settings'
            -   '@folderName': 'Box'
                '@solutionType': 'Base'
                '@url': ''
```

Порядок создания нового проекта и переключения на него:

1. Клонируйте (или создайте) каталоги с репозиториями проекта
2. На основе ранее созданного конфига проекта создайте новый и заполните его.

3. Создайте копию проекта

```
do map clone_project c:\rx\4530_BoxOnly.yml
c:\rx\4530_RecurringActionItems.yml
```

Будет создана копия проекта и предложено переключиться на него.

Вернуться на проект `BoxOnly` можно будет командой:

```
do map set c:\rx\4530_BoxOnly.yml
```

Командой `dds_wo_deploy` можно запустить DevelopmentStudio без фактического переключения на проект

```
`do map dds_wo_deploy  c:\rx\4530_BoxOnly.yml`
```

Это позволяет открыть исходники другого проекта одновременно с текущим. При этом будет заблокирована возможность публикации разработки.

Установка компоненты Manage Applied Projects

Установить можно двумя способами:

1. Как компоненту Directum Launcher
2. Как plugin Directum Launcher

Вариант с plugin удобнее, при наличии нескольких инстансов RX - обновление компоненты будет подхватываться сразу во всех инстансах RX.

Установка как компоненты Directum Launcher

1. Скачайте ManageAppProjects.zip из https://github.com/DirectumCompany/rx-manage-applied-projects/releases.
2. Перейдите в папку Directum Launcher и выполните команду:

```
do.bat components add_package <путь к файлу>\ManageAppProjects.zip
```

Чтобы обновить компоненту, установленную таким способом, её нужно будет сначала удалить:

```
do.bat components delete map
```

Установка как plugin Directum Launcher

1. Клонируйте репозиторий компоненты:

   `git clone https://github.com/DirectumCompany/rx-manage-applied-projects`
2. Перейдите в папку Directum Launcher и выполните команду

```
do.bat install_plugin <путь к репозиторию>\src\ManageAppProjects\map_plugin\
```

Чтобы обновить компоненту, установленную таким образом достаточно вытянуть из репозитория обновления.

Удаление инстанса

Для удаления инстанса воспользуйтесь скриптом remove_instance_rx.ps1. Например

```
powershell D:\rx-manage-applied-
projects\src\ManageAppProjects\map_plugin\remove_instance_rx.ps1 c:\rx_ver 4530
```

При выполнении этой команды:

- будут остановлены сервисы инстанса
- удалены пул приложений и веб-сервер инстанса
- удален каталог инстанса

Данные инстанса и конфиги проектов не удаляются.

Формирование дистрибутивов решений

Зачастую подготовка дистрибутивов решения выглядит несколько сложнее, чем просто выгрузка пакета разработки:

- пакетов разработки в дистрибутиве может быть несколько
- у решения может быть несколько вариантов дистрибутивов для разных целей и с разным набором пакетов разработки
- кроме пакетов разработки в дистрибутив нужно скопировать и другие файлы

Чтобы сделать этот процесс проще можно создать конфиг с описанием дистрибутивов и одной командой формировать все дистрибутивы.

Создать конфиг с описанием дистрибутивов

Создать конфиг с описанием дистрибутив можно командой

```
do map generate_empty_distributions_config путь_к_создаваемому_конфигу
```

В результате будет создан шаблон конфига

```
# Название проекта
project: ''

# mtd-файл, из которого берется номер текущей версии
mtd_for_version: '....Solution.Shared\Module.mtd'

# XML-конфиги, которые используются для формирования пакета разработки в процессе
увеличения версии решения
devpacks_for_increment_version:
-   config: ''

# Файлы и каталоги, которые копируются в каждый дистрибутив
to_every_set:
-   'src': ''
```

```
    'dst': ''

# Описание дистрибутивов
distributions:
    # идентификатор дистритутива
-   'id': ''
    # описание сути дистрибутива
    'comment': ''
    # папка дистрибутива, создается внутри папки версии решения
    'folder_name': ''
    # Значимая часть имени zip-архива с дистрибутивом. Если указать пустую строку
- архив не создается
    'zip_name': 'Образец '
    # Пакеты разработки, которые нужно поместить в дистрибутив
    'devpacks':
    -   'config': '.xml'
        'result': '.dat'
    # Уникальные файлы, которые нужно поместить в конкретный дистрибутив
    'files':
    -   'src': ''
        'dst': ''
```

Особенности заполнения конфига описания дистрибутивов:

- `mtd_for_version` - относительный (от корня репозитория) путь к mtd-файлу основного solution-решения. Из этого mtd-файла будет браться номер версии решения
- `devpacks_for_increment_version` - xml-файл конфигурации пакета разработки (см. в справке про невизуальный режим экспорта разработки), который будет использоваться когда нужно будет увеличить номер версии решения. В этом конфиге должны содержаться все solution решения и параметр `IncludeAssemblies` должен иметь значение `true`.
- `to_every_set` - список файлов, которые будут скопированы в каждый дистрибутив.
    - `src` - относительный или абсолютный путь к файлу/каталогу, который нужно скорпировать.
    - `dst` - относительный (от папки дистрибутива) путь назначения
- `distributions` - список описаний дистрибутивов, которые нужно создать
    - `folder_name` - имя папки с конкретным дистрибутивом
    - `zip_name` - имя архива с дистрибутивом. К этому имени будет добавлен номер версии.
    - `devpacks` - список пакетов разработки, которые нужно добавить в дистрибутив.
        - `config` - относительный (от корня репозитория) или абсолютный путь к xml-файлу конфигурации пакета разработки
        - `result` - относительный (от папки дистрибутива) путь к файлу пакета разработки, который нужно создать
    - `files` - список файлов, которые будут скопированы в конкретный дистрибутив.
        - `src` - относительный или абсолютный путь к файлу/каталогу, который нужно скорпировать.
        - `dst` - относительный (от папки дистрибутива) путь назначения

Пример заполнения файла

```
# Название проекта
project: 'Пример прикладного решения'

# mtd-файл, из которого берется номер текущей версии
mtd_for_version: 'DirRX.SampleSolution\DirRX.SampleSolution.Shared\Module.mtd'

# XML-конфиги, которые используются для формирования пакета разработки в процессе
увеличения версии решения
devpacks_for_increment_version:
-   config: 'build\devpack_SampleSolution.xml'

# Файлы и каталоги, которые копируются в каждый дистрибутив
to_every_set:
-   'src': 'doc'
    'dst': 'doc'
-   'src': 'data\Templates'
    'dst': 'Templates'

# Описание дистрибутивов
distributions:
    # идентификатор дистритутива
-   'id': 'implementation'
    # описание сути дистрибутива
    'comment': 'Для использования на проектах внедрения'
    # папка дистрибутива, создается внутри папки версии решения
    'folder_name': 'SampleSolution_implementation'
    # Значимая часть имени zip-архива с дистрибутивом. Если указать пустую строку
- архив не создается
    'zip_name': 'Образец решения (для проектов внедрения)'
    # Пакеты разработки, которые нужно поместить в дистрибутив
    'devpacks':
    -   'config': 'build\devpack_SampleSolution.xml'
        'result': 'SampleSolution.dat'
    -   'config': 'build\devpack_SampleSolution_debug.xml'
        'result': 'SampleSolution_debug.dat'
    # Уникальные файлы, которые нужно поместить в конкретный дистрибутив
    'files':
    -   'src': 'build\distributions\SampleSolution_implementation\readme.md'
        'dst': 'readme.md'
    -   'src': 'build\distributions\SampleSolution_implementation\doc.md'
        'dst': 'doc.md'

    # идентификатор дистритутива
-   'id': 'evaluation'
    # описание сути дистрибутива
    'comment': 'Для передачи в ознакомительных целях - только бинарники, без
исходников.
Текст комментария может занимать несколько строк'
    # папка дистрибутива, создается внутри папки версии решения
    'folder_name': 'SampleSolution_evaluation'
    # Значимая часть имени zip-архива с дистрибутивом. Если указать пустую строку
- архив не создается
    'zip_name': 'Образец решения (ознакомительная версия)'
    # Пакеты разработки, которые нужно поместить в дистрибутив
    'devpacks':
```

```
    -   'config': 'build\devpack_SampleSolution_binonly.xml'
        'result': 'SampleSolution_evaluation.dat'
    # Уникальные файлы, которые нужно поместить в конкретный дистрибутив
    'files':
    -   'src': 'build\distributions\SampleSolution_evaluation\readme.md'
        'dst': 'readme.md'
```

Собрать дистрибутивы решения

Сборка дистрибутивов выполняется командой

```
do map build_distributions путь_к_файлу_описания_дистрибутивов
путь_к_папке_назначения путь_к_репозиторию_решения --increment_version
```

где:

- `путь_к_файлу_описания_дистрибутивов` - путь к подготовленному файлу с описанием дистрибутивов
- `путь_к_папке_назначения` - путь к папке в которой будет создана папке с номером версии, внутри которой будут созданы папки с дистрибутивами
- `путь_к_репозиторию_решения` - путь к репозиторию решения
- `--increment_version` - необязательный параметр, указывающий на то, нужно или нет увеличить номер версии решения

Экспортировать пакет разработки

Команда `export_devpack` может использовать для выгрузки одиночного пакета разработки. Например, для установки на тестовый стенд

```
do map export_devpack имя_файла_конфига_пакета_разработки имя_пакета разработки
```

Особенности поведения команд

Полный список команд можно посмотреть командой `do map help`

У команд `set`, `clone_project` и `create_project` есть общее в поведении:

- параметр `--confirm`, который определяет будет или нет выводиться запросы на подтверждение действий пользователя. При указании `--confirm=False` запросы выводиться не будут. По умолчанию - true.
- параметр `--rundds`, который определяет будет или нет предложено запустить DevelopmentStudio после выполнения команды. значение по умолчанию определяется параметром `run_dds_after_set_project` в config.yml. Если параметр в config.yml не указан, то будет false.

set - переключиться на проект

В результате будет скорректирован config.yml, обновлены конфиги сервисов и компонент и перезапущены сервисы RX.

```
do map set <имя файла с описанием проекта>
```

create_project - создание нового прикладного проекта

Чтобы создать новый прикладной проект необходимо:

1. Подготовить файл описания проекта
2. Выполнить команду `do map create_project <файл-описания-проекта>`

В результате выполнения:

- будет создана новая БД
- будет создан новый домашний каталог проекта
- если в параметре `--package_path` указан пакет разработки
    - разработка из пакета будет принята в БД и выполнена инициализация
    - будут импортированы стандартные шаблоны
    - если указан параметр `--need_import_src`, то сразу будут импортированы исходные коды из того же пакета разработки.

clone_project - создание копии прикладного проекта

В некоторых случаях создать новый проект удобнее копирование существующего. Например, чтобы безопасно проверить изменения в ветке проекта.

Чтобы сделать копию проекта необходимо:

1. Подготовить файл описания проекта
2. Выполнить команду `do map clone_project <файл-описания-проекта-источника>  <файл-описания-проекта-назначения>`

В результате выполнения:

- будет создана копия БД
- будет создана копия домашнего каталога проекта

Особенности работы:

- при использовании Microsoft SQL Server:
    - создается полная копия исходной БД с режимом COPY_ONLY, чтобы не нарушать политику резервного копирования
    - копия БД создается в том же каталоге, где расположена сама БД и удаляется после восстановления из неё копии БД
- при использовании PostgreSQL:
    - копия БД парой выполняется парой утилит `pg_dump | psql` и для корректной работы требует настройки файла паролей pgpass.conf

dds_wo_deploy - запустить DevelopmentStudio для просмотра/редактирования исходников указанного проекта без возможности публикации

Запускает DevelopmentStudio для просмотра/редактирования исходников указанного проекта (параметр `--project_config_path`) без возможности. Полезна для ситуаций, когда требуется открыть исходники сразу нескольких проектов.

Особенности работы:

- создает временный файл аналогичный config.yml с пустыми параметрами LOCAL_WEB_RELATIVE_PATH, LOCAL_SERVER_HTTP_PORT и SERVICE_RUNNER_CONFIG_PATH в секции DevelopmentStudio (это блокирует возможность публиковать разработку);
- на основе временного config.yml создает временный файл аналогичный _ConfigSettings.xml для DevelopmentStudio;
- запускает DevelopmentStudio передавая ему созданных аналог _ConfigSettings.xml;
- имена временных файлов выводятся в лог;
- после закрытия DevelopmentStudio временные файл удаляются.

generate_empty_project_config - создать заготовку для файла описания проекта

Создаст пустой шаблон файла описания проекта.

clear_log - удаление старых файлов логов

По умолчанию RX настроен так, что каждый день компоненты и сервисы RX создают новые файлы логов. На рабочем месте прикладного разработчика, как правило, важны текущие логи, а старые не имеют особого значения. Для удаления логов можно использовать команду

```
do map clear_log
```

По умолчанию удаляются логи старее 3-х дней. Используя параметр `--limit_day` можно изменить глубину удаляемых логов. Удаление всех логов, кроме сегодняшних:

```
do map clear_log  --limit_day=1
```

current - информация о текущем проекта

Посмотреть ключевую информацию по текущему проекту:

```
do map current
```

url - получить url подключения к веб-доступу текущего проекта

```
do map url
```

Используя команду `do map url | clip` ссылку можно сразу скопировать в буфер обмена.

check_config - посмотреть содержимое файла описания проекта

```
do map check_config
```

check_sdk - проверить наличие необходимых версий .Net

```
do map check_sdk
```

Проверить наличие и соответствие версий для git и dotnet.

run_script - запустить python-скрипт в контексте DirectumLauncher

```
do map run_script --script_filename=script_example.py
```

Позволяет запустить внешний python-скрипт в контексте DirectumLauncher.

В скрипте будет доступны:

- все импортированные в map_installer.py функции, модули
- предопределенная переменная self_map - ссылка на экземпляр класса ManageAppliedProject, через который вызывается команда `run_script`.
- дополнительные параметры, которые будут доступны в коде скрипта в качестве локальных переменных. Пример вызова скрипта с доп.параметрами:

Пример скрипта - см. script_example.py.

```
do map run_script --script_filename=script_example.py arg1=value1 arg2=value2
```

Особенности разработки скриптов

1. Проверка передан или нет параметр для скрипта:

```
# проверить передан ли параметра arg1
if "arg1" not in locals():
    log.error("Пропущен обязательный параметр arg1")
    sys.exit(-1)
```

2. Использование переменных и методов экземпляра класса ManageAppliedProject, в контексте которого выполняется скрипт

```
# отобразить путь к config.yml текущего экземпляра RX
log.info(f'Путь к config.yml: {self_map.config_path}')
# вызвать метод класса ManageAppliedProject
self_map.check_sdk(need_pause=False)
```

3. Использование функций, определенных в map_installer.py

```
# вызвать функцию, определенную в map_installer.py
log.info(f'Версия RX: {_get_rx_version()}')
```

4. Переменные, доступные в основном теле скрипта, недоступны в функциях, определенных в скрипте. Например, при выполнении скрипта

```
var1 = "value"
def f1():
    log.info(f'  f1() run')
    print(var1)
f1()
```

будет ошибка `NameError: name 'var1' is not defined.`

Для исправление необходимо нужные переменные передать в качестве параметра в функцию:

```
var1 = "value"
def f1(var1):
    log.info(f'  f1() run')
    print(var1)
f1(var1)
```

5. Функциии и классы, определенные в основном теле скрипта, доступны в самом скрипте, но не доступны для вызова друг в друге. Например, при выполнении скрипта

```
def f1():
    log.info(f'  f1() run')
def f2():
    log.info(f'  f2() run')
    f1()
f1()
f2()
```

вызов функции f1() из основного тела скрипта отработает корректно, а при вызове f1() внутри f() будет ошибка `NameError: name 'f1' is not defined.`
Для исправление необходимо функции и классы определять внутри функции-обвертки, а в теле скрипта вызывать функцию обвертку:

```
def main_func():
    def f1():
        log.info(f'  f1() run')
    def f2():
        log.info(f'  f2() run')
        f1()
    f1()
    f2()
main_func()
```