icCFPatcher - утилита изменения метаданных CF файла конфигурации.

Колчанов А.В.
<kolchanov.s@infocentre.su>

Утилита icCFPatcher предназначена для автоматизации внесения однотипных изменений в метаданные конфигурации 1С файла *.cf. Для разборки и сборки CF файла используется сторонняя утилита: V8Unpack (v. 2.0). Скачать ее можно по адресу: http://infostart.ru/public/15695/. Вносимые изменения оформляются в виде скрипт-файла синтаксиса языка программирования Python. Утилита поддерживает только определенные изменения. Включение дополнительных изменения требует доработки программы. Утилита выполнена для удобства использования в виде мастера-визарда. Возможно кроссплатформенное использование программы (проверенно на Ubuntu Linux и Windows).

1. Инсталяция и деинсталляция (Windows)

Инсталляция:

1) Запустите cfpatcher-setup.exe

2) На первой странице инсталятора надо выбрать инсталлируемые пакеты (см. рис. 1.1).

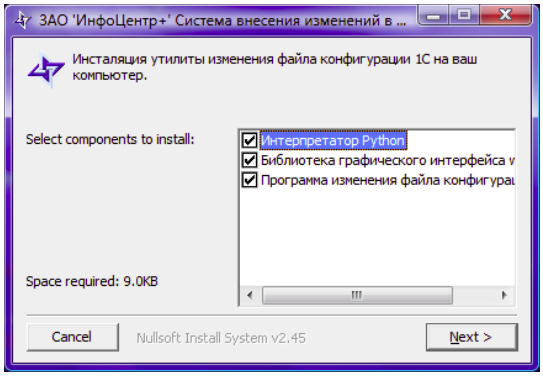

Рисунок 1.1: Запуск инсталляции

3) Выберите инсталляционную папку и следуйте инструкциям инсталлятора (см. рис. 1.2). По умолчанию инсталляционная папка C:\cf_patcher.

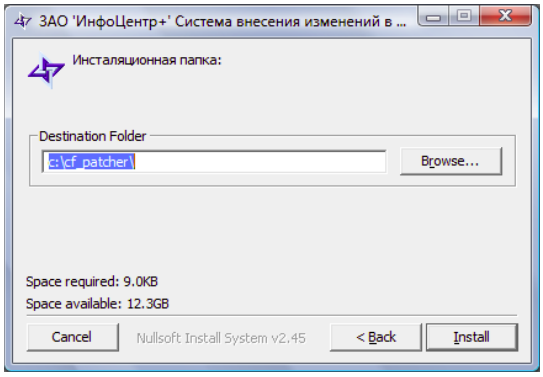

Рисунок 1.2: Выбор инсталляционной папки

4) По окончании инсталляции нажмите <Close> (см. рис. 1.3). На рабочем столе и в меню программ инсталлятором создаются ярлыки утилиты а также программы деинсталляции.

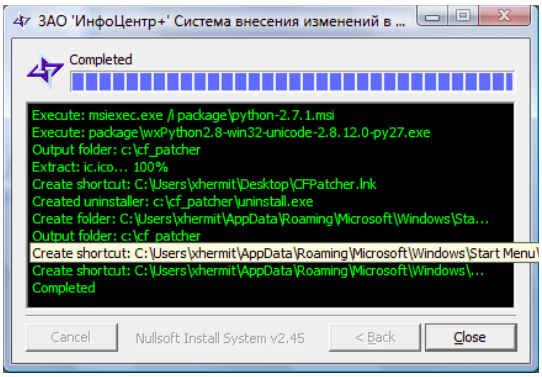

Рисунок 1.3: Окончание инсталляции

Деинсталляция:

1)Меню → Программы → icCFPatcher-uninstall (см. рис.1.4).

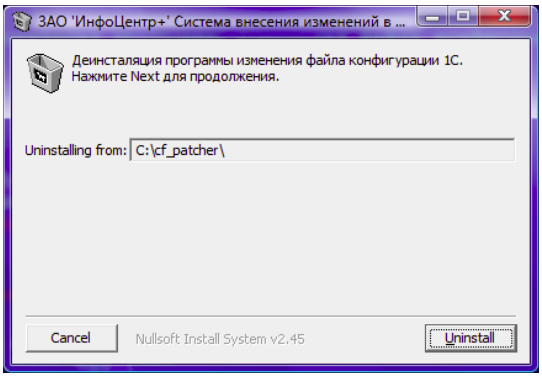

Рисунок 1.4: Деинсталляция

2)По окончании деинсталляции нажмите <Close> (см. рис. 1.5).

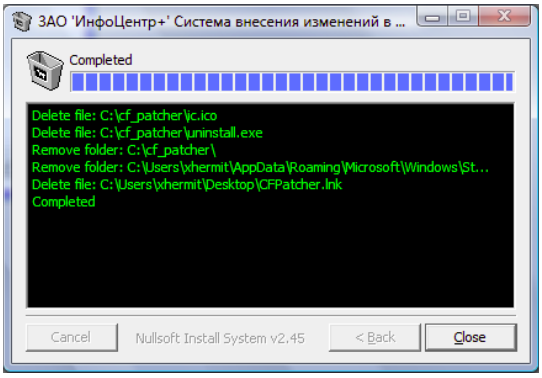

Рисунок 1.5: Деинсталляция

2. Инсталяция (Linux)

Под Linux для работы программы достаточно распаковать zip файл программы.

Для запуска утилиты необходимо установить кроме Python такие пакеты как Wine и wxPython.

3. Использование

Файловая организация программы:

```
V8Unpack20/ - директория утилиты V8Unpack
doc/        - документация
icpatcher/  - директория программы
img/        - изображения кнопок
scripts/    - скрипты изменений
start.bat   - запускающий скрипт для Windows
start.sh    - запускающий скрипт для Linux
```

Программа запускается с помощью **start.bat** (Windows) или **start.sh** (Linux). В верхней части окна программы указанна версия и дата сборки программы.

1) На первой странице необходимо выбрать CF файл конфигурации, который будем изменять (см. рис.3.1). Поле папки конфигурации, в которую будет происходить разворачивание, генерируется по умолчанию как **/путь/к/утилите/имя_файла_cf**. Папка конфигурации может быть изменена пользователем.

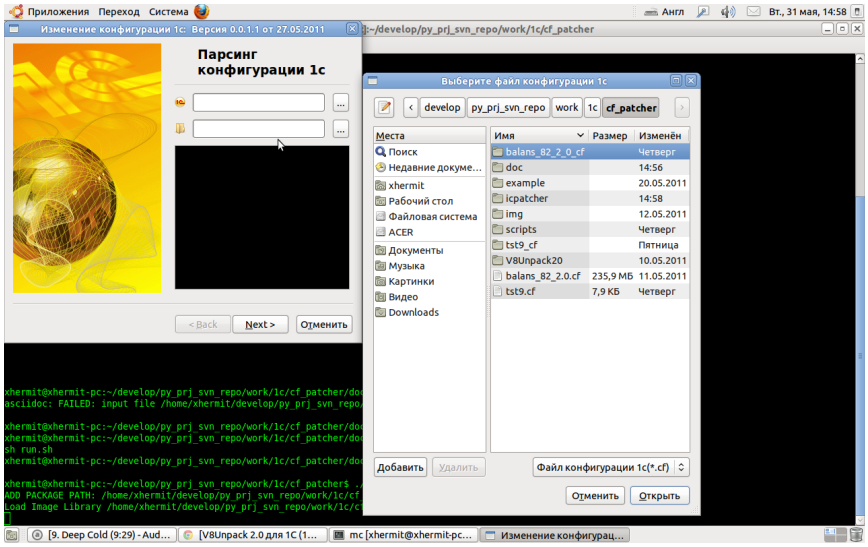

Рисунок 3.1: Страница1. Выбор CF файла конфигурации

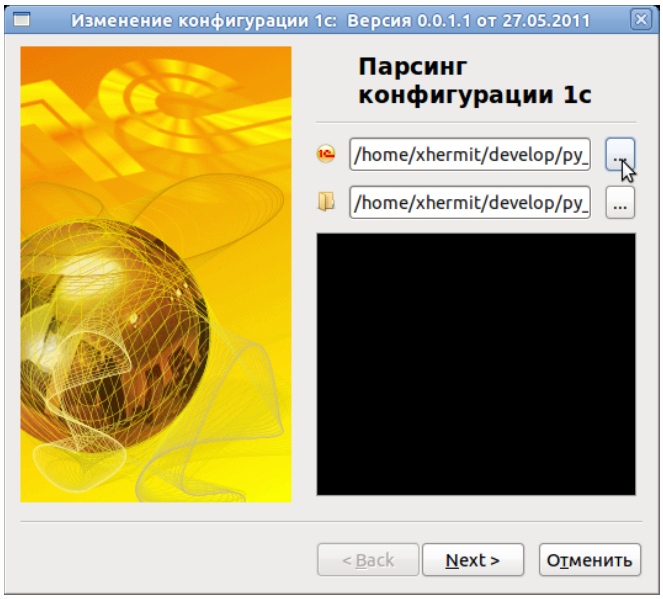

Рисунок 3.2: Страница1. Выбор CF файла конфигурации

Если фвйл конфигурации или папка конфигурации не указаны, то после нажатия на кнопку <Next> будет произведен переход на заключительную страницу мастера (см. рис.3.8). Дальнейшая работа утилиты не возможна.

2) На второй странице мастера можно отметить те метаобъекты, которые будут подвергнуты изменению. (см. рис.3.3) Если объекты не отмечаются, то считается что выбор объектов производится в изменяющем скрипте по мнемоническому правилу, заданному с помощью регулярного выражения отбора из имен метаобъектов.

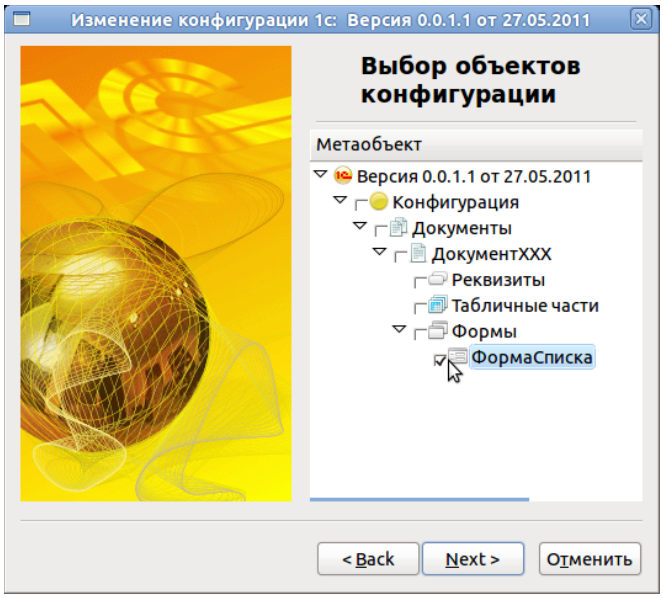

Рисунок 3.3: Страница2. Выбор метаобъектов для изменения

3) На третьей странице мастера пользователь выбирает последовательность внесения изменений в конфигурацию с помощью списка скриптов (см.рис. 3.4). Скрипты находятся в папке scripts и создаются пользователем.

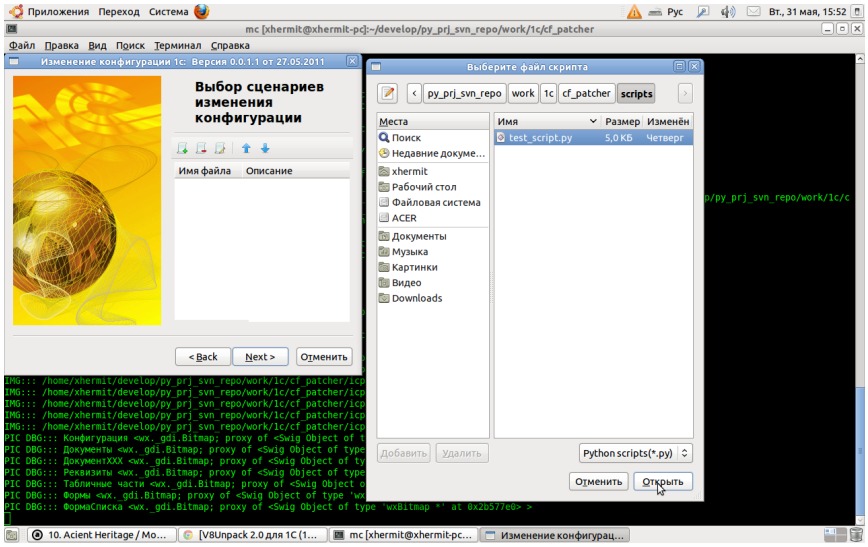

Рисунок 3.4: Страница3. Выбор скриптов изменения

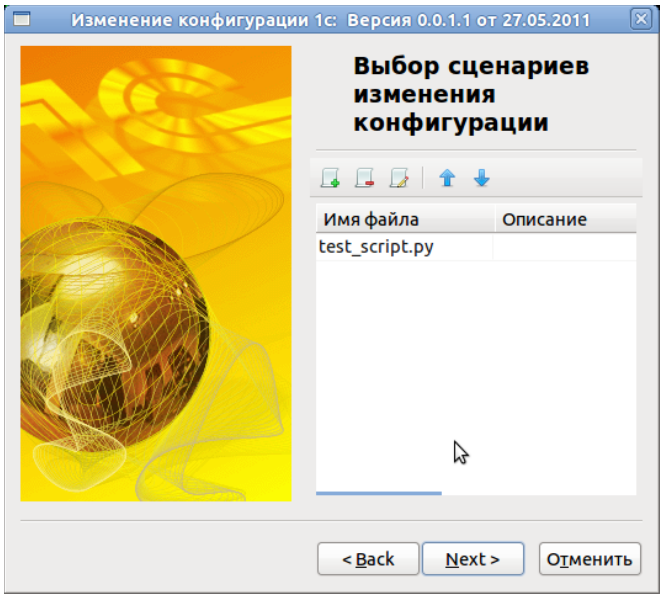

Рисунок 3.5: Страница3. Выбор скриптов изменения

4) Четвертая страница мастера предлагает запустить изменения конфигурации (см. рис. 3.6). Изменения по умолчанию будут происходить в папке конфигурации. Пользователь может сменить папку и тогда конфигурации перед изменением будет скопирована в новую папку.

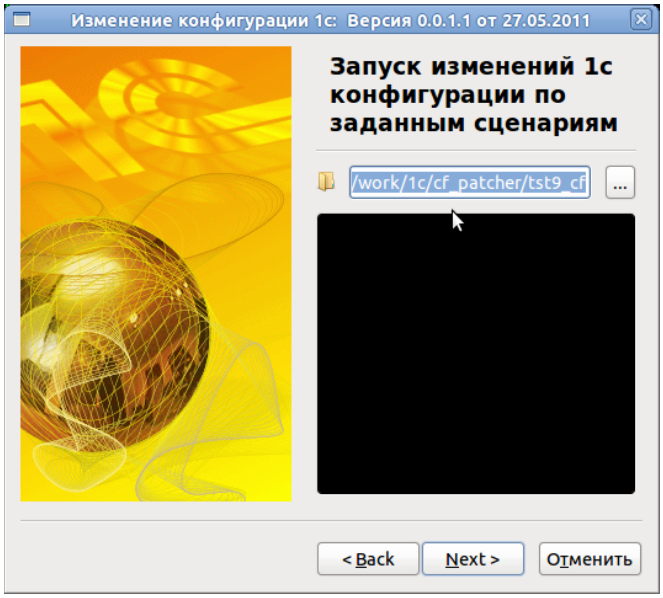

Рисунок 3.6: Страница4. Запуск изменений

5) После внесенных изменений в конфигурацию необходимо произвести сборку нового CF файла с изменениями (см. рис.3.7). Имя файла по умолчанию генерируется как **имя файла_patch_год_месяц_дата_часы_минуты_секунды.cf**. При необходимости пользователь может сменить имя CF файла.

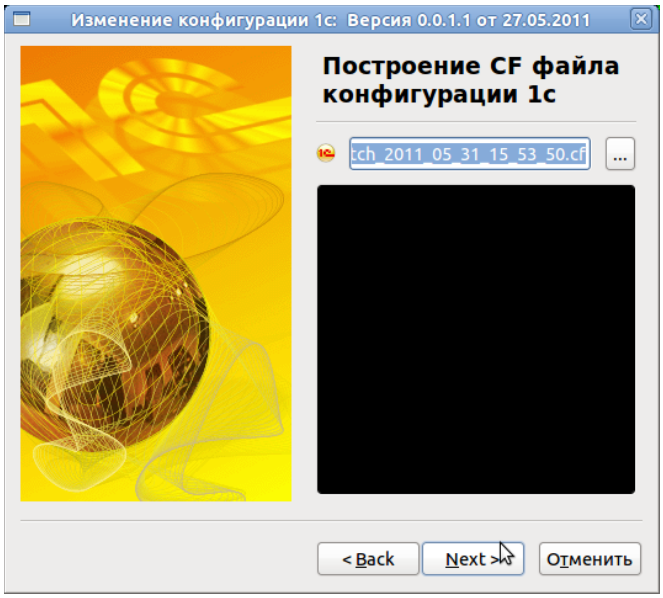

Рисунок 3.7: Страница5. Сборка CF файла конфигурации 1С

6) Об успешном окончании работы программы оповещает заключительная страница (см. рис. 3.8). Выход из программы производится по кнопке <Finish>.

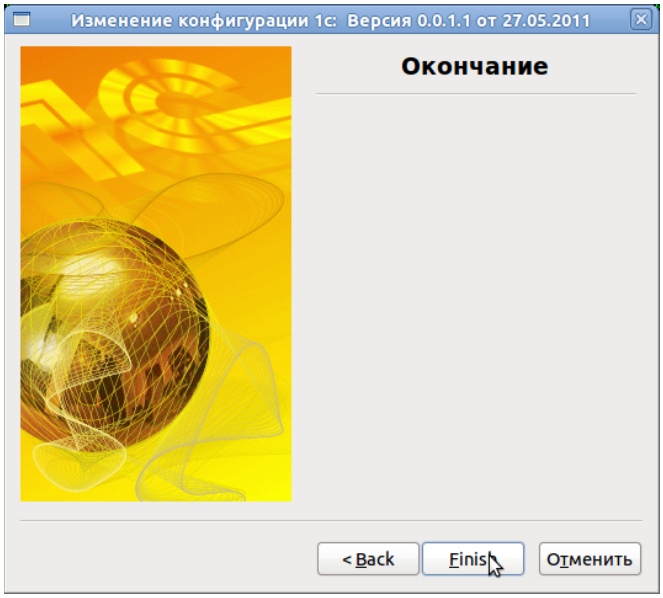

Рисунок 3.8: Страница6. Окончание

4. Написание скриптов изменений

Пример скрипта

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Тестовый скрипт.
"""

#Для каждого объекта "Документ", который не имеет префикса "иц", с формой списка необходимо сделать следующее:

#1. Добавить реквизиты формы: ДеревоКолонки, ДеревоОтборы, ДеревоПериоды.
#   Тип - ДеревоЗначений.

addCF(u'РеквизитФормы',u'ДеревоКолонки',Type_=u'ДеревоЗначений',
    NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')
```

```
addCF(u'РеквизитФормы',u'ДеревоОтборы',Type_=u'ДеревоЗначений',
    NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')
addCF(u'РеквизитФормы',u'ДеревоПериоды',Type_=u'ДеревоЗначений',
    NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')

#2. В модуль формы добавить:

#Процедура ПередОткрытием(Отказ, СтандартнаяОбработка)
#       ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_УстановитьЭлементы(ЭтаФорма);
#КонецПроцедуры

#Процедура ОтборНажатие(Элемент)
#       ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_Нажатие(ЭтаФорма,Элемент);
#КонецПроцедуры

#Процедура ОтборАктивизация(Элемент)
#       ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_Активизация(ЭтаФорма,Элемент);
#КонецПроцедуры

#Процедура ДокументСписокПередНачаломДобавления(Элемент, Отказ, Копирование)
#       ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_ДобавлениеНовогоОбъекта(ДокументСписок, ЭтаФорма, Элемент, Отказ, Копирование,
#КонецПроцедуры

updateCF(u'МодульФормы',u'''
Процедура ПередОткрытием(Отказ, СтандартнаяОбработка)
        ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_УстановитьЭлементы(ЭтаФорма);
КонецПроцедуры

Процедура ОтборНажатие(Элемент)
        ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_Нажатие(ЭтаФорма,Элемент);
КонецПроцедуры

Процедура ОтборАктивизация(Элемент)
        ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_Активизация(ЭтаФорма,Элемент);
КонецПроцедуры

Процедура ДокументСписокПередНачаломДобавления(Элемент, Отказ, Копирование)
        ицРаботаСоСправочникомВидыДокументов.ИспользованиеОтбора_ДобавлениеНовогоОбъекта(ДокументСписок, ЭтаФорма, Элемент, Отказ, Копирование)
КонецПроцедуры
''',NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')

#3. В табличное поле формы, в событие "ПередНачаломДобавления" вписать процедуру "ДокументСписокПередНачаломДобавления"

#Формы спика у документа может не быть. В этом случае ничего для такого документа не делать.

addCF(u'СобытиеФормы',
      Name_=u'ДокументСписок.ПередНачаломДобавления',Value_=u'ДокументСписокПередНачаломДобавления',
      NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')

#4. При добавлении процедуры "ПередОткрытием" необходимо в свойствах формы в событие "ПередОткрытием"
#задавать имя процедуры-обработчика "ПередОткрытием".

addCF(u'СобытиеФормы',
      Name_=u'.ПередОткрытием',Value_=u'ПередОткрытием',
      NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')
```

Рассмотрим скрипт более детально.

Все комментарии начинаются с симвоола #.

Следующая секция указывает какой интерпретатор нужен для исполнения скрипта.

```
#!/usr/bin/env python
```

Указание кодировки скрипта

```
# -*- coding: utf-8 -*-
```

Описание скрипта

```
"""
Тестовый скрипт.
"""
```

Вызов функции добавления

```
addCF(u'РеквизитФормы',u'ДеревоКолонки',Type_=u'ДеревоЗначений',
    NameFilter_='.Документы.<\A(?!иц*)>.Формы.ФормаСписка')
```

В скрипте может присутстствовать несколько функций по изменению конфигурации: **addCF** - функция добавления элемента в конфигурацию **delCF** - функция удаления элемента из конфигурации **replaceCF** - функция замены одного

элемента конфигурации на другой, если элемент не найден, то замены произведено не будет **updateCF** - функция обновления одного элемента другим, если элемент не найден в конфигурации, то происходит добавление нового элемента

В каждой вызываемой функции первым аргументом является имя элемента, затем значение элемента и далее дополнительные параметры.

Также важен аргумент **NameFilter_**, который задает правила выбора изменяемых метаобъектов из конфигурации. Значения выбираются по иерархии метаобъектов в дереве конфигурации. Разделителем является точка. Например: **.Документы.<ic([0-9a-zA-Z])**>.Формы.ФормаСписка* - читается как "Из конфигурации, из всех документов, выбрать те, которые начинаются с *ic* и в своем имени имеют цифры и латинские буквы. И у этих документов выбрать из всех форм формы списка" Значение между символами < и > задает регулярное выражение применяемое к имени метаобъекта для его выбора для изменения. Регулярные выражения задаются согласно синтаксиса регулярных выражений языка Python. Если этот аргумент не задан, то подразумевается, что пользователь сам в мастере выбирает метаобъекты для изменения (см. рис. 3.3).

Для отладки регулярных выражения можно пользоваться сторонней утилитой PyReditor. http://sourceforge.net/projects/regexeditor/.

Last updated 2011-06-03 12:08:20 KRAST