RS:Core написание собственных расширений

# Концепция ПО

RS:Core является ПО, которое обеспечивает «прозрачное» взаимодействие с оборудованием ТСД с одной стороны, и предоставляет скриптовый язык BeanShell для написания расширений, которые решают конкретные задачи разработчиков на местах.

Т.о. расширение написанное на ТСД одной модели, может быть перенесено на ТСД другой модели с установленным RS:Core без каких-либо доработок. Основная задача расширений — предоставлять возможности системам, ограниченным в доступе к Android API, таким как 1C. Программисты, пишущие с использованием Android API, могут получить доступ к оборудованию ТСД через механизм интерфейсов. Поэтому, данный документ будет разбит на три части, в первой описана архитектура RS:Core, во второй написание расширений и использование их из 1С, в третьей описание методов интерфейса RS:Core.

# Архитектура RS:Core



RS:Coreсервис

Расширение

Расширение

ПользовательскоеПО на 1С

ПО с использованиемAndroid API

ВК RSDriver

Из приведенной выше схемы видно, что сервис RS:Core взаимодействует с оборудованием и обеспечивает интерфейс к нему для расширений. Приложение, написанное допустим на 1C, через внешнюю компоненту RSDriver может взаимодействовать с расширениями (но не с самим RS:Core), и получать необходимый ему функционал. Данный механизм выбран потому что RS:Core предоставляет слишком низкий уровень абстракции, что серьезно усложняет разработку на 1С и подобных системах (например, требуется работа со сложными типами данных, которые передать через границы процесса достаточно затруднительно и накладно в плане затрат рабочего времени). При использовании Android API эти же действия не требуют такого объема работ, поэтому для программ написанных на Android API доступны как вызовы расширений так и доступ к устройствам.

Расширения. Расширением называется модуль, написанный на языке BeanShell, который реализует функциональность предоставляемую RS:Core апи, и в свою очередь предоставляет «наружу» методы, которые могут быть вызваны из других процессов. Т.к. методы вызываются из других процессов, то соответственно передаваемые и возвращаемые типы данных допускаются только маршализуемые. Расширения могут быть как написаны пользователем, так и загружены из магазина расширений компании Rightscan.

# Расширения RS:Core, концепции и примеры

В физическом смысле, расширение является zip архивом, который содержит минимум два файла, \_\_info\_\_.json который описывает расширение и \_\_init\_\_.rcs, который является кодом расширения на языке BeanShell. Кроме них в архиве (будет именоваться в дальнейшем bundle) могут содержаться другие ресурсы, такие как иконки, текста и прочее.  
Разберем файл описания расширения \_\_info\_\_.json. Этот файл содержит JSON объект, описывающий расширение. Для него обязательны следующие поля:  
**Name** — имя расширения, используемое для вызовов. Имя не должно содержать символ : (двоеточие). Других ограничений на имя не накладывается;  
**Version** — версия расширения, записываемая в виде числа. Хорошим тоном является использование минимум трехзначных цифр, где разряд едениц описывает номер резиза, разряд десятков минорную версию, и остальные разряды — мажорную;  
**Description** — описание расширения. Это поле предназначено для описания функционала расширения.

Таким образом, пример \_\_info\_\_.json для расширения может выглядеть так  
{  
"Name":"RSExample",  
"Version":100,  
"Description":"Пример расширения RS:Core"  
}

Файл \_\_init\_\_.rsc является текстовым файлом, который содержит код расширения на BeanShell. Файл должен содержать создание экземпляра класса rs.core.api.CoreExtension который собственно и является расширением. Методы этого класса подробно описаны в javadoc документации, приложенной к данному документу, мы же разберем тут основные методы класса. Кроме этих файлов можно добавить файл icon.png содержащий конку расширения в фомате PNG. Рекомендуется размер 32x32 пикселя. Эта иконка будет отопражаться в приложении «RS:Core Интерфейс» для вашего приложения.

В принципе, класс не содерит каких-либо методов, обязательных к реализации, однако есть два метода, реализация которых желательна. Это метод onLoad и метод onUnload. Методы описаны следующим образом:  
void onLoad()  
void onUnload()

Метод onLoad вызывается RS:Core при загрузке расширения, когда уже все устройства проинициализированны. Метод onUnload вызывается, когда расширение выгружается из памяти (например, перед удалением). Этот метод **не вызывается** при выключении ТСД.

Любой метод расширения доступен для вызова из RSDriver, или через сервисы. Методы, не рекомендуемые к вызовам «извне» должны начинаться с символов \_\_ (два подчеркивания).

Так же в классе CoreExtension доступна константа DEVICE которая является экземпляром класса rs.core.DeviceDriver и содержит объекты работы с различными устройствами, реализованными в ТСД. Любое устройство наследовано от класса rs.core.hw.HWDevice. Этот класс реализует методы, одинаковые для всего оборудования. Т.к. многое оборудование способно сгенерировать данные в любой момент времени (например нажатие кнопки на клавиатуре или обнаружение NFC устройства), то механизм работы с оборудованием в большинстве своем случаем следующее:

* расширение создает экземпляр класса rs.core.api.RSEventHandler, который будет получать сигналы от оборудования;
* расширение регистрирует его как слушатель событий оборудования через вызов метода registerHandler объекта оборудования;
* при возниконовении события (сканирование штрихкода, нажатие клавиши и т. д.) ввызывается метод onEvent слушателя, в который передается объект оборудования вызвавшего события, и данные события (зависят от оборудования)
* по окончании работы, или в методе onUnload расширение обязано отключить слушатель с помощью метода unregisterHandler оборудования

Ниже приведен пример обработки событий нажатий клавиш, который по нажатию показывает сообщение с кодом нажатой клавиши  
new CoreExtension() {  
 kbReceiver = new RSEventHandler() {  
 onEvent(sender, data) {  
 if(sender == DEVICE.keyboard())   
// в случае клавиатуры в data передается экземпляр класса android.view.KeyEvent  
 if( data.getAction() == data.ACTION\_DOWN)  
 showToast("нажата "+data.getKeyCode());  
 }  
 };  
 void onLoad() {  
 DEVICE.keyboard().registerHanler(kbReceiver);  
 }  
 void onUnload() {  
 DEVICE.keyboard().unregisterHanler(kbReceiver);  
 }  
}

Однако, расширение «само в себе» не представляет никакого интереса для разработчика (хотя, в некоторых случаях оно может оказаться полезным, если реализует например функционал включения/отключения сканера баркодов по какой-либо кнопке на ТСД). Поэтому, добавим в наше расширение метод, который будет включать/выключать сообщения по команде из другого приложения. Для этого реализуем метод enableEcho, который будет принимать параметр типа boolean.   
  
new CoreExtension() {  
 kbReceiver = new RSEventHandler() {  
 onEvent(sender, data) {  
 if(sender == DEVICE.keyboard())   
// в случае клавиатуры в data передается экземпляр класса android.view.KeyEvent  
 if( data.getAction() == data.ACTION\_DOWN)  
 showToast("нажата "+data.getKeyCode());  
 }  
 };  
 void enableEcho(on) {  
 if(on)  
 DEVICE.keyboard().registerHanler(kbReceiver);  
 else  
 DEVICE.keyboard().unregisterHanler(kbReceiver);  
 }  
 void onLoad() {  
 }  
 void onUnload() {  
 DEVICE.keyboard().unregisterHanler(kbReceiver);  
 }  
}

Для проверки расширения нужно собрать bundle. Для этого упаковываем файлы \_\_info\_\_.json и \_\_init\_\_.rsc в zip архив с произвольным именем и расширением bundle. Допустим example.bundle. Этот файл нужно поместить в каталог /sdcard/RS на устройстве, и в приложении «RS:Core Интерфейс» выполнить перезагрузку расширений. Вы увидите, что ваше расширение успешно загружено. Теперь можно обратиться к нему. Для этого не нужно писать какую-либо программу, доступ к расширениям возможен через броадкаст сообщение **rs.core.tools**. Необходимо передать два стороковых параметра в вызов, это **Call**, содержащий имя вызова в формате *расширение:метод*, и **Params**, содержащий параметры, упаковывнные в JSON массив. В случае вызова метода без параметров Params может отсутствовать. Т.о. тестовый вызов с помощью утилиты am будет выглядеть так:  
am broadcast -a rs.core.tools -e Call "RSExample:enableEcho" -e Params "[true]"  
для включения эхотеста и   
am broadcast -a rs.core.tools -e Call "RSExample:enableEcho" -e Params "[false]"  
для выключения.