

Оглавление

Введение.....	2
Способы интеграции с ARTA Synergy.....	2
Синхронизация/интеграция server-side.....	2
Прямая.....	2
Событийная.....	2
События ARTA Synergy.....	4
События пользователей.....	4
События реестров.....	4
Модуль, влияющий на ход исполнения маршрута.....	5
Блокирующий процесс.....	5
SQL Запрос.....	5
Приложение, работающее от имени пользователя по API.....	5
WEB-модуль, встроенный в ARTA Synergy.....	6
Дополнительный обработчик для стандартного процесса ARTA Synergy.....	7
Способы авторизации в ARTA Synergy.....	9
Авторизация по логину и паролю.....	9
Сессионная авторизация.....	9
Авторизация по ключам.....	9
Как задеплоить интеграционное приложение.....	11
Стандартные решения интеграционных задач.....	12
Синхронизация орг. структуры и пользователей.....	12

Введение

Данный документ представляет собой описание основных способов интеграции с ARTA Synergy.

Способы интеграции с ARTA Synergy

1. Синхронизация/интеграция server-side
 1. Событийная
 2. Прямая
2. Модуль, влияющий на ход исполнения маршрута
 1. Блокирующий процесс
 2. SQL запрос
3. Приложение, работающее от имени пользователя по API
4. WEB-модуль, встроенный в ARTA Synergy
5. Дополнительный обработчик для стандартного процесса ARTA Synergy

Синхронизация/интеграция server-side

Существует два основных подхода для интеграции с Synergy:

- Прямая интеграция — интеграционные модули разрабатываются с использованием API Synergy и интегрируемых систем. Синхронизация данных между системами и координация обмена между ними остаётся за разработчиком интеграционного модуля
- Событийная интеграция — когда какая-либо из подсистем Synergy генерирует различные события, связанные с какими-либо данными. Обработчики этих событий (на стороне Synergy) при необходимости преобразовывают данные событий и передают их интегрируемой системе через какой-либо транспортный уровень

Прямая

ARTA Synergy предоставляет API для доступа к своим функциям с помощью rest сервисов. Описание методов API можно посмотреть в [javadoc](#). Авторизация для всех методов API — Basic HTTP, подробнее о способах авторизации Способы авторизации в ARTA Synergy.

Событийная

Под «событием» мы будем подразумевать *сообщение* о каком-либо изменении в Arta Synergy, содержащее тип события и минимально необходимые для получения связанной с событием информации либо воздействия на Synergy *данные*. Обработчик события (или событий) — программный модуль, читающий сообщения о событиях из JMS Queue или JMS Topic и осуществляющий, при необходимости, доступ к экземпляру Synergy, сгенерировавшему сообщение, с помощью API Synergy.

Обработчик событий является отдельным от Arta Synergy приложением, которое может работать как на том же сервере приложений, что и Arta Synergy, так и на удалённом.

Кроме этого, обработчик события может иметь собственные конфигурационные файлы, необходимые для реализации целевого назначения.

Обработчик событий может обрабатывать как конкретное событие (например, `event.registers.formdata.add`), так и класс событий (например, `event.registers.*`).

Обработка события может происходить в 3 этапа:

1. Получение события
2. Получение и преобразование необходимых обработчику данных
3. Передача сформированного пакета данных далее (опционально)

ARTA Synergy генерирует событие в случае если для этого события настроены обработчики. Обработчики событий настраиваются в конфигурационном файле `jboss_home/standalone/configuration/arta/api-observation-configuration.xml`.

Сообщение, помещаемое в очередь JMS представляет собой экземпляр `javax.jms.TextMessage`. Тело сообщения зависит от типа события, его описание можно посмотреть ниже среди описаний типов событий. Каждое событие содержит свойство `api_event`, указывающее на тип события, вызвавшего его (содержимое тега `<event>event.registers.formdata.add</event>` в конфигурационном файле).

Например:

```
<configuration>
  <listener>
    <queue>java:jboss/queues/Synergy/UsersQueue</queue>
    <event>event.users.*</event>
  </listener>
  <listener>
    <queue>java:jboss/queues/Synergy/RegisterCreateDocQueue</queue>
    <event>event.registers.formdata.add</event>
  </listener>
</configuration>
```

В этом примере настроены обработчики:

- 1) `java:jboss/queues/Synergy/UsersQueue` для всех событий класса `event.users.*`, т. е. всех событий связанных с пользователями: `event.users.account.change`, `event.users.formdata.change`, `event.users.account.add` и т. д.
- 2) `java:jboss/queues/Synergy/RegisterCreateDocQueue` для события добавления записи реестра `event.registers.formdata.add`.

Рассмотрим, например, код обработчика очереди `UsersQueue`

```
public class UsersMessagesListener implements MessageListener {

    public void onMessage(Message message) {
        //Получаем идентификатор пользователя, для которого
        //сгенерировано событие
        String userID = ((TextMessage) message).getText();
        //Получаем тип события
        String eventType = message.getStringProperty("api_event");

        //Выполнение действие по получению дополнительных данных через API
        //и прочих операций, зависящих от условий решаемой задачи
    }
}
```

Ниже описаны типы событий, которые могут быть сгенерированы ARTA Synergy.

События ARTA Synergy

События пользователей

Данные события генерируются при для каждого из нижеописанных случаев изменения *данных пользователей*:

- [event.users.account.change] Изменение данных полей *первичной карточки* пользователя, т. е. параметров его учётной записи:
 - Фамилия
 - Имя
 - Отчество
 - Логин
 - Код для показателей
 - e-mail
 - JID
 - Личная папка пользователя
- [event.users.formdata.change] Изменение данных *карточек пользователей на основе форм*, ассоциированных с ним посредством функциональности «Отдел кадров»
- [event.users.account.add] Добавление новой записи учётной записи пользователя (и связанными с ней файлами по формам «отдела кадров»)
- [event.users.account.delete] Удаление (пометка «удалённые») учётной записи пользователя (и связанных с ней файлов по формам «отдела кадров»)
- [event.users.contactdata.change] Изменение «контактных данных» пользователя — изменение/добавление записей раздела «Контакты» профиля пользователя (модуль «Сотрудники») следующих типов:
 - Skype
 - Рабочий телефон
 - XMPP
 - Адрес
 - Мобильный телефон
 - Почта
 - Телефон

Для всех событий типа event.users.* передаваемые данные — ID пользователя Synergy.

События реестров

Событие для реестра не генерируются самостоятельно и не имеют predeterminedных

названий. Для того чтобы для реестра было сгенерировано событие, необходимо в процесс активации/изменения/удаления реестра добавить процесс «Событие реестра» и указать в поле «Название» его название.

Название события должно начинаться со строки «event.registries.formdata.»». Для различных событий и для различных реестров могут быть указаны разные либо одинаковые названия событий в зависимости от целей решаемой задачи.

☒ Отображать при сохранении

Действия						
Название	Ответственный	Действие	Нагрузка	Возврат	Длительность	Формат
1 event.registries.formdata.change		Событие реестра	20%	Нет	8ч	Нет
Добавить этап						

Подробнее о настройке и использовании реестров можно узнать в Руководстве Методолога раздел 3.1.2 Реестры и Руководстве Пользователя раздел 7.2 Реестры.

Модуль, влияющий на ход исполнения маршрута

Блокирующий процесс

Блокирующий процесс предназначен для того, чтобы предоставить возможность в маршрут активации/изменения/удаления реестра вставить асинхронный вызов внешнего модуля. Основное отличие блокирующего процесса от события реестра (см. События реестров) заключается в том, что:

- при использовании блокирующего процесса маршрут реестра дожидается ответа о результате выполнения операции внешним модулем
- блокирующий процесс может завершиться положительно или отрицательно, что повлияет на дальнейшую работу маршрута (Если блокирующий процесс завершится отрицательно — процесс остановится, если положительно — то продолжит работу дальше)

Модуль реализующий блокирующий процесс должен представлять собой отдельное приложение, задеплоенное на jboss в соответствии с правилами, описанными в разделе Как задеплоить интеграционное приложение.

Запускается код модуля блокирующего процесса через очередь. При старте этапа маршрута, содержащего блокирующий процесс, в очередь добавляется сообщение, которое должен обработать модуль.

Конфигурация блокирующего процесса:

Для того чтобы добавить блокирующий процесс необходимо выполнить следующие действия:

1. Добавить процесс с в маршрут реестра в конфигураторе:

Название	Ответственный	Действие	Нагрузка	Возврат	Длительность	Форма
1 event.blocking.example		Блокирующий процес	20%	Нет		8ч Нет
Добавить этап						

Название процесса должно начинаться с **event.blocking.** и далее строка, характеризующая суть блокирующего процесса.

2. Создать очередь JMS для блокирующего процесса. Для этого необходимо в конфигурационный файл jboss (в стандартной установке это /opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml) в секцию <subsystem xmlns="urn:jboss:domain:messaging:1.2"> добавить:

```
<jms-queue name="ExampleQueue">
  <entry name="java:jboss/queues/Integration/ExampleQueue"/>
  <durable>true</durable>
</jms-queue>
```

3. Связать очередь и процесс через конфигурационный файл JBOSS_HOME/standalone/configuration/arta/api-observation-configuration.xml, добавив в него следующее

```
<listener>
  <queue>java:jboss/queues/Integration/ExampleQueue</queue>
  <event>event.blocking.example</event>
```

```
</listener>
```

Обратите внимание, что название блокирующего процесса, указанное в маршруте в конфигураторе должно быть равно значению тега в конфигурационном файле `api-observation-configuration.xml` (в данном примере: `event.blocking.example`) и название очереди должно совпадать со значением тега `queue` конфигурационного файла `api-observation-configuration.xml` (в данном примере: `java:jboss/queues/Integration/ExampleQueue`)

Сообщение передаваемое в очередь является экземпляром `TextMessage`. Содержимым сообщения является объект `json` с полями:

1. `dataUUID` — идентификатор данных по форме записи реестра
2. `executionID` — идентификатор блокирующего процесса
3. `documentID` — идентификатор документа реестра

После того как модуль обратится к внешней системе и выполнит необходимые действия, он должен вызвать метод `API Synergy` для того, чтобы вернуть результат выполнения процесса и продолжить работу маршрута. Для того чтобы это сделать необходимо вызвать метод `API kz.arta.synergy.samples.processes.blocking.ProcessesService#signalProcess`.

Примечание: В примере кода ниже разблокировка маршрута осуществляется в методе `onMessage`. Если время выполнения действия значительно или зависит от внешних факторов (например, доступность интегрируемой системы или необходимость ввода пользователем данных в интегрируемой системе), то разблокировка маршрута может произойти позже, в любой другой момент времени из другого метода, а сам метод `onMessage` должен завершиться без ошибок, «запомнив» переданные параметры.

Пример кода (обработка сообщения и разблокировка маршрута):

@Override

```
package kz.arta.synergy.samples.processes.blocking;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

/**
 * <p><Пример блокирующего процесса</p>
 */
public class BlockingQueueListener implements MessageListener {

    public void onMessage(Message message) {

        String dataUUID = null;
        String executionID = null;
        String documentID = null;

        if (!(message instanceof TextMessage)){
            return;
        }

        try {

            JsonFactory factory = new JsonFactory();
            JsonParser parser = factory.createJsonParser(((TextMessage)
message).getText());
            JsonToken token = null;

            while ((token = parser.nextToken()) != null) {
                if (token == JsonToken.FIELD_NAME) {
                    String fieldName = parser.getText();
                    parser.nextToken();
                    String value = parser.getText();
                    if (fieldName.equals("dataUUID")){
                        dataUUID = value;
                    } else if (fieldName.equals("executionID")){
                        executionID = value;
                    } else if (fieldName.equals("documentID")){
                        documentID = value;
                    }
                }
            }

            //Выполнение каких-либо действий
            .....

            //Разблокировка маршрута

            String address = "http://127.0.0.1:8080/Synergy";
            String login = "1";
            String password = "1";
            String signal = "got_agree";
            boolean isSuccess = false;
        }
    }
}
```



```

        try {
            URL url = new URL(address + "/rest/api/processes/signal?signal=" +
signal + "&executionID=" + executionID + "&param1=resolution&value1=sigal_is_" +
signal);

            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json; charset=utf-8");

            String encoded = Base64.encode((login + ":" + password).getBytes());
            conn.setRequestProperty("Authorization", "Basic " + encoded);

            String output;
            StringBuffer result = new StringBuffer();

            BufferedReader br = new BufferedReader(new
InputStreamReader((conn.getInputStream())));

            while ((output = br.readLine()) != null) {
                result.append(output);
            }

            conn.disconnect();

            JsonFactory factory = new JsonFactory();
            JsonParser parser = factory.createJsonParser(result.toString());
            JsonToken token = null;

            while ((token = parser.nextToken()) != null) {
                if (token == JsonToken.FIELD_NAME) {
                    String fieldName = parser.getText();
                    token = parser.nextToken();
                    if (fieldName.equals("errorCode") &&
parser.getText().equals("0")){
                        isSuccess = true;
                    }
                }
            }
        } catch (Exception exc){
            logger.error(exc.getMessage(), exc);
        }

        } catch (Exception exc){
            logger.error(exc.getMessage(), exc);
        }
    }
}

```

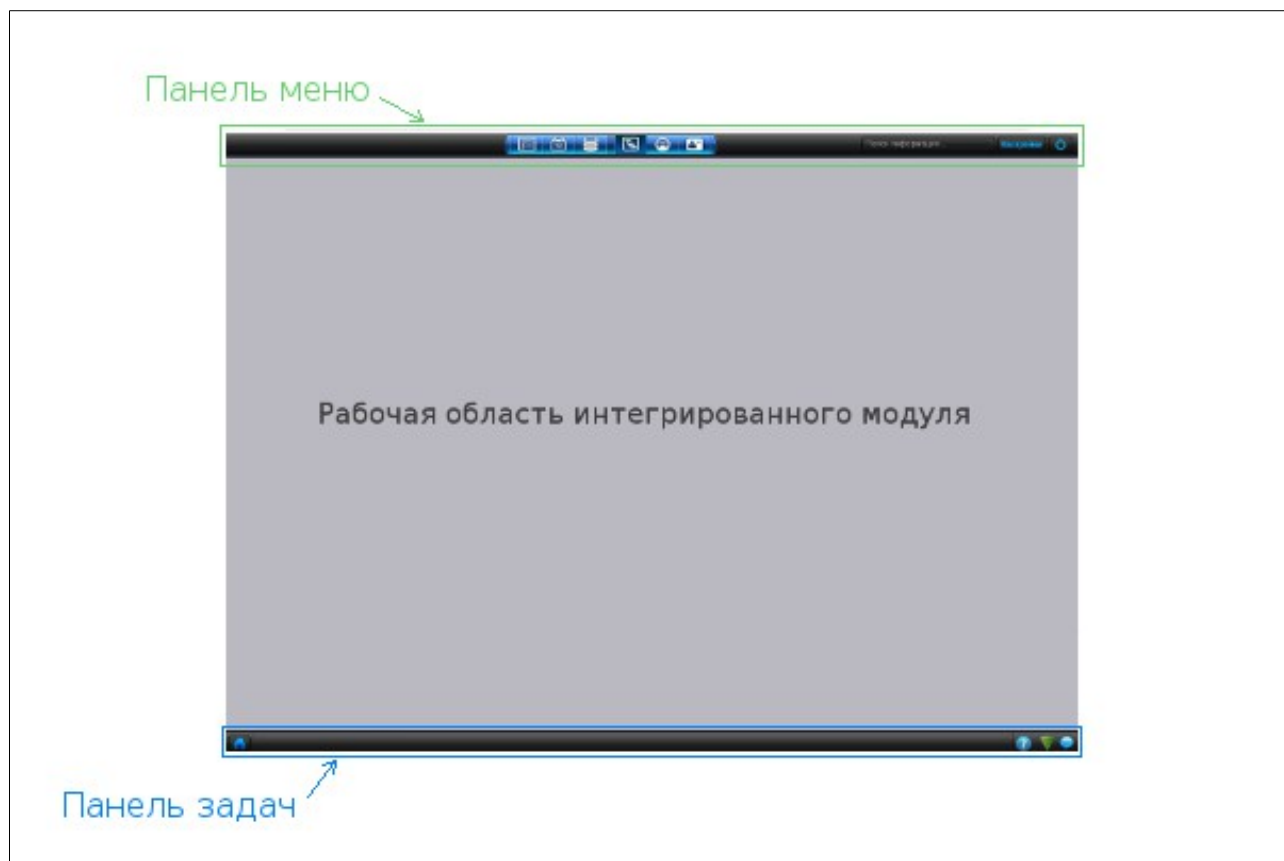
SQL Запрос

Приложение, работающее от имени пользователя по API

Пример: мобильное приложение

WEB-модуль, встроенный в ARTA Synergy

Web приложение внешнего модуля открывается в iframe в окне основного приложения. При этом рабочая область внешнего модуля занимает всю область страницы кроме панели меню и панели задач:



На данный момент не существует пользовательского интерфейса для добавления нового модуля, поэтому это необходимо сделать с помощью SQL запроса. Необходимо добавить запись в таблицу `outer_modules`:

- `id` — идентификатор модуля, должен совпадать с идентификатором вашего проекта в репозитории проектов (см. [общие правила к разработке модулей](#))
- `name_ru`, `name_kz`, `name_en` — название модуля на русском, казахском и английском языках соответственно
- `url` — адрес приложения
- `description` — описание модуля
- `active` — активен ли модуль, 1/0.

Для реализации SSO приложений, ARTA Synergy при загрузке интегрированного модуля будет в строку URL добавлять два параметра:

1. `locale` — локаль авторизованного пользователя
2. `sso_hash` — hash сумма для идентификации пользователя.
3. `host` — адрес, с которого загружено приложение Synergy

Например, если `url` приложения `http://host:port/plans_module`, то запрашиваться будет URL `http://host:port/plans_module?locale=locale_value&sso_hash=sso_hash_value`

Интегрированный модуль должен будет получить из URL параметр `sso_hash` и запросить по API у ARTA Synergy информацию об авторизованном пользователе (идентификатор, имя). Если метод API возвращает информацию о пользователе, это

подтверждает, что данный пользователь действительно авторизован с данного хоста, в данном браузере.

Далее строка `sso_hash` будет использована для Сессионная авторизация и вызова API Arta Synergy.

В ARTA Synergy реализована возможность обращения к ее модулям по относительной ссылке. Такая же возможность существует для внешних web модулей. Переход по ссылке вида:

```
#submodule=outer&outerModuleID=значение_из_таблицы_outer_modules&прочие_параметры_по_желанию_модуля
```

активирует в Synergy заданный модуль и передаст ему заданные в url-е параметры (параметры `locale`, `ssh_hash`, `host` так же будут переданы, несмотря на то что они отсутствуют в ссылке).

Дополнительный обработчик для стандартного процесса ARTA Synergy

Цель данного вида интеграции — дать возможность проверить возможность запуска стандартный процесса и при необходимости прервать его.

Стандартный функционал платформы ARTA Synergy дает возможность запретить отправку документов на согласование, утверждение если количество уровней орг. структуры между отправителем и получателем, превышает некоторое настроенное значение. Но в некоторых компаниях существуют более сложные правила, ограничивающие возможность отправки документов/работ. В этих случаях необходима разработка данного обработчика.

Обработчик может быть применён к процессам:

- «согласование» (agreement-single)
- «утверждение» (approval-single)
- «ознакомление» (acquaintance-single)
- «работа» (assignment-single)
- «общий процесс при запуске по формам» (common-process-by-form)
- «отправка документа» (send-document)
- «отправка документа по форме» (send-document-by-form)

Обработчик представляет собой Java-класс, реализующий интерфейс

`kz.arta.synergy.integration.api.bp.StartHandlerIF`.

Данный интерфейс находится в библиотеке `integration-api.jar`, которую можно взять здесь:

`svn://@scm.forge.arta.local/svnroot/synergy/trunk/build/artifacts/integration-api.jar`

Интерфейс содержит два метода:

- `makeDecision` — проверяет возможно ли выполнение процесса
- `getResolution` — возвращает текст, который должен быть записан в ход исполнения

Более подробную информацию о полях методов можно посмотреть в java-doc к этим методам, которые доступны в **`integration-api.jar`** (библиотека содержит и скомпилированные классы и исходный код).

Установка обработчика для процесса осуществляется с помощью конфигурационного файла `${jboss.server.config.dir}/arta/process-handlers-configuration.xml`, имеющего следующий

формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-handlers-configuration
  xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/arta
http://www.arta.kz/xml/ns/ai/process-handlers-configuration
.xsd">
  <handlers>
    <handler>
      <process>agreement-single</process>
      <handler-providers>kz.arta.ext.Trk</handler-providers>
    </handler>
  </handlers>
</process-handlers-configuration>
```

На каждый процесс может быть указано несколько классов обработчиков (handler-providers) (через пробел) — они будут выполнены в указанном порядке.

Обработчики выполняются последовательно до тех пор пока метод makeDecision одного из них не вернет false, после этого процесс прерывается.

Библиотеку, содержащую обработчик необходимо скопировать в папку /opt/synergy/jboss/standalone/deployments/Synergy.ear/lib.

После копирования библиотеки обработчика и изменения файла process-handlers-configuration.xml jboss необходимо перезапустить.

Примечание. Процесс common-process-by-form запускает процессы agreement-single, approval-single, acquaintance-single, assignment-single (подпроцессы). Поэтому, если обработчик будет запрещать выполнение подпроцесса и при этом разрешать выполнение процесса common-process-by-form, то подпроцессы все равно будут прерваны. Аналогично, если выполнение common-process-by-form разрешено, а выполнение подпроцесса запрещено, подпроцессы будут прерваны.

Способы авторизации в ARTA Synergy

API ARTA Synergy доступно только авторизованным пользователям. Тип авторизации — BASIC HTTP. Методы API выполняются от имени того пользователя, который авторизован. Имеются следующие типы авторизации:

Авторизация по логину и паролю

Авторизация пользователя по его логину и паролю приемлема в тех случаях, когда приложение может знать текущий логин и пароль пользователя, например:

- Приложение предоставляет альтернативный интерфейс к некоторым модулям Synergy (мобильное приложение, десктопный клиент для хранилища)
- Приложение представляет собой server-side утилиту для синхронизации, для которого создан выделенный пользователь и его логин и пароль хранятся в конфигурационном файле на сервере.

Для реализации данного типа авторизации надо передать в запросе заголовок:

Authorization, со значением «Basic » + Base64(login + «:» + password). Например:

Логин	Administrator
Пароль	123456
Значение заголовка	Basic QWRtaW5pc3RyYXRvcjoxMjM0NTY=

Сессионная авторизация

Сессионная авторизация используется для встроенных WEB модулей. При сессионной авторизации также используется тип — BASIC HTTP, но в качестве логина пользователя необходимо использовать значение «\$session» и в качестве пароля — полученное значение sso_hash.

Таким образом заголовок *Authorization* должен иметь значение:

«Basic » + Base64(«\$session:» + sso_hash). Например:

Значение sso_hash	D3RONfC52dtJO5XgDyn5qUMv
Значение заголовка	Basic JHNlc3Npb246RDNST05mQzUyZHRKTzVYZ0R5bjVxVU12

Авторизация по ключам

Модуль, который хочет авторизоваться от имени какого-либо пользователя таким способом, должен сгенерировать для него ключевую пару, обеспечив сохранность закрытого ключа. Затем модуль сохраняет получивший открытый ключ для пользователя в Synergy, используя следующий вызов API:

```
kz.arta.synergy.server.api.rest.person.PersonService#generateUserAuthKey
```

Этот вызов назначает ключ тому пользователю, от имени которого выполняется, поэтому для использования.

Если ранее для данного пользователя был сгенерирован другой ключ, то предыдущий автоматически становится недействительным.

Создать ключ можно только для существующего WEB-модуля, так как для этого требуется идентификатор приложения. Если у вас нет необходимости разрабатывать WEB модуль, но есть необходимость в использовании авторизации по ключам можно создать такой модуль на уровне БД и отключить его использование в административном приложении SynergyAdmin для всех элементов орг. структуры.

Использование этого ключа для авторизации аналогично использованию сессионного ключа. Тип авторизации Basic HTTP, в качестве логина пользователя надо использовать строку «\$key», в качестве пароля — полученный с помощью API ключ.

Таким образом заголовок *Authorization* должен иметь значение:

«Basic » + Base64(«\$key:» + значение_ключа). Например:

Значение ключа	MS03Y2Q0ZGU3YS0zYjRkLTQ2NjgtYWlyOC0zZDI1YzgxZGNmOGZfMjAxMy0xMC0zMSAxNzo0Mg==
Значение заголовка	Basic JGtleTpNUzAzWTJRMFpHVtNZUzB6WWpSa0xUUTJOamd0WVdJeU9DMHpaREkxWXpneFpHTm1PR1pmTWpBeE15MHhNQzB6TVNBeE56bzBNZz09

Как задеплоить интеграционное приложение

ARTA Synergy работает на сервере приложений JBossAS7. Интеграционное приложение может представлять собой jar-файл либо war-файл либо их комбинацию.

Если приложение является одиночным файлом, его можно задеплоить, скопировав в директорию `jboss_home/standalone/deployments`. Если приложение состоит из нескольких файлов, необходимо создать *.ear приложение.

Если приложение имеет зависимости на внешние библиотеки и они находятся в модулях JBoss-a (`jboss_home/modules`) необходимо использовать их, прочие зависимости — помещать внутрь приложения.

В целях безопасности работы приложения Synergy и сервера приложений категорически запрещается помещать артефакты интеграционного модуля в приложение Synergy.ear и изменять состав модулей (`jboss_home/modules`).

Стандартные решения интеграционных задач

Синхронизация орг. структуры и пользователей

Under construction

Данная задача заключается в синхронизации подразделений, должностей, пользователей с внешней системой. В качестве главной системы может быть как ARTA SYNERGY, так и внешняя система.

Основным вопросом в реализации синхронизации является установление соответствия между сущностями внешней системы и ARTA Synergy. Этот вопрос решается в каждом случае индивидуально. При его решении можно прийти к тому, что набор дефолтных полей ARTA Synergy недостаточен. В этом случае необходимо составить карточки с дополнительными полями для должностей, подразделений, пользователей.