ESAP CS通讯接口协议

Caland ESAP C-S Interface and Protocol

|  |  |
| --- | --- |
| **文档名称** | ESAP CS通讯接口协议 |
| **作者** | 苏智 |
| **类别** | Caland 软件开发详细设计文档 |
| **子类别** | 系统架构设计/接口定义/数据定义 |
| **摘要** | 本文档主要描述ESAP服务器与客户端之间的接口，及其需要了解的数据模型内容。  第一部分先是概述整体ESAP系统结构，从几个关键操作流程来总结所需要的接口。  第二部分接口的描述。  第三部分是接口中所涉及的各种数据包的JSON格式详细解析；着重体现了诸如Action的设计和描述等。  第四部分为建议的客户端采用的数据存储。该存储基于服务器端数据库设计，依赖于服务器端最终存储。 |

**修订历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **日期** | **主要修改人** | **审阅** | **摘要** |
| 1.0 | 2013/11/12 | 苏智 |  | 创建 |
| 1.01 | 2013/11/18 | 苏智 |  | 整理数据库和Action |
| 1.1 | 2013/11/28 | 李旭 |  | 修改部分报文，完成接口具体定义。 |
| 2.0 | 2014/4/24 | 苏智 |  | 重新整理数据库部分；数据包以JSON格式重新定义给出；增加通讯录相关数据；完善客户端操作逻辑介绍 |
| 2.1 | 2014/5/30 | 苏智 |  | 与Web产品沟通协调后调整，上传下载方式修订，流程优化调整 |
| 2.2 | 2014/9/9 | 苏智，朱鹏志 |  | Web产品更新功能，接口相应的调整 |
|  |  |  |  |  |
|  |  |  |  |  |

# 客户端流程描述

## 登录流程

客户端应当支持脱机登录功能（即联网不是必须的）。脱机登录功能则是基于“用户登录信息缓存”实现。在整个流程中，黄色的**“用户信息调用Login接口”**需要调用到服务器的接口。

用户登录信息缓存

服务器  
用户信息

启动

显示登录界面

从本地缓存获取用户信息

用户信息调用Login接口

检查缓存

检查联网

从用户输入获取用户信息

联网登录成功

分析调用结果

登录界面

A：允许更改用户

B：不准更改用户

缓存有默认登录用户信息

缓存有默认登录用户信息

有联网

无联网

鉴权通过

鉴权失败

清空用户登录缓存

写入用户登录缓存

脱机登录成功

准许脱机登录

准许

检查缓存

有缓存

不准许

准许脱机登录：用户信息缓存应当有一个可持久存储数据记录是否准许用户脱机登录。默认准许。

该部分逻辑后文称为“用户鉴权”

直接尝试同步操作

缓存无默认登录用户信息

图表 1登录流程逻辑图

## 同步流程

同步流程包括两个方面的内容。一个是切换到联网状态后客户端Action缓存的执行（在执行动作过程中，使用到了**“调用Action接口”**和**“调用Update接口”**两个接口），一个是文件上传下载的处理（下载使用固定的地址和文件ID参数即可；在上载过程中，使用到了 **“UploadComplete”**接口）。其中，Action和Update两个接口都要绑定用户鉴权操作。

同步开始

联机状态

调用Action接口

用户鉴权

同步结束

有Action

无Action

联机

脱机

联机

鉴权失败

文件上传下载过程

Action缓存

不准许脱机登录  
不准许更改用户

清除Action缓存

到登录流程的登录界面

调用Update接口

用户鉴权

更新除文件上下传以外数据

尝试联网

脱机

联机状态

到登录流程的登录界面

联机状态

脱机

联机

鉴权失败

不准许脱机登录  
不准许更改用户

成功

失败

同步失败

记录调用日志

图表 2同步操作逻辑图

同步操作开始以后，会有三种可能的结束情况：同步失败，同步结束，以及鉴权失败后回到登录流程的登录界面。

### 上传下载的处理方式

1. 为上传下载分别设置一个队列。可同时执行至少一个上传和一个下载任务。（可通过设置增加并发传输数量。）
2. 客户端如果是要从服务器获取一个文件新的版本下载，或客户端上传一个文件新的版本到服务器，均不可首先删除原版本文件，直到新版本文件传输完毕。
3. 上传文件文件完成后，需要单独调用一个**“UploadComplete”**接口来提交上传完成的信息。

server\_version > local\_version时，执行下载。  
下载完成，使local\_version = server\_version，physical\_filename被trans\_filename取代；  
如中途reset，客户端放弃trans\_filename，本地两个版本号信息保持不变。

physical\_filename

local\_version

server\_version

physical\_filename

version

trans\_filename

客户端

服务器

server\_version < local\_version时，执行上传。上传操作需要首先锁定文件（体现在upload接口中）  
上传完成，使server\_version = new\_version，new\_filename取代Physical\_filename，file解除锁定；  
如中途reset，客户端本地两个版本号信息保持不变，服务器放弃new\_filename和new\_version。

physical\_filename

local\_version

server\_version

physical\_filename

new\_version

version

new\_filename

客户端

服务器

## 执行动作流程

在执行动作过程中，使用到了**“调用Action接口”**和**“调用Update接口”**两个接口。这两个接口均需要首先进行用户鉴权，然后才能真正在服务器上执行。由于Update是联网Action的一个补充操作，这里需要注意不准许脱机登录的逻辑处理方式。

Action  
队列缓存

服务器  
接口

用户操作开始

创建生成Action

将Action添加到队列缓存

联机状态

调用Action接口

用户鉴权

用户操作结束

这里的联网状态是之前联网（登录流程中检查）尝试后的一个结果，这里并不真正去检查是否可以联网

仅有1个Action

多于1个Action

联机

脱机

鉴权失败

鉴权失败

清空Action队列缓存

检查缓存队列

Action其后的Update  
不准许脱机登录  
不准许更改用户。

记录调用日志

到登录流程的登录界面

到登录流程的登录界面

调用Update接口

用户鉴权

更新客户端数据

联机状态

脱机

联机

完成用户该操作结果

清空Action队列缓存

图表 3用户操作逻辑图

### Update操作后处理

1. 获取到的TARGET数据包中，会包含可能对应的FILE PACKAGE。该数据应当与本地的对应FILE PACKAGE进行VERSION的比较，用来决定在同步操作中，是要上传还是下载。
2. FILE PACKAGE中包括了SIGN FLOW PACKAGE，SIGN PACKAGE，从而可能获得具有一些本地通讯录没有的联系人信息。故在TARGET处理过程中，也要注意通讯录的更新处理。

### Action的说明

**Action是客户端记录用户所有操作的一个大集合定义。包括以下操作：**

Create Action: 新建一条Target记录（文件或文件夹）

Delete Action: 删除一条Target记录（对于文件夹对象，需递归删除所有子项）

Rename Action: 重命名Target原文件名

Move Action: 移动Target，从一个父节点到新的父节点

**Update Request Action: 更新文件（仅对文件），通知服务器本地新的文件版本号，如果动作成功，则接续Upload接口完成**

SignPenNEW Action: 提交签名图片

signpendel Action: 删除签名图片

Lock Action: 请求锁定文件（仅对文件），该操作是进行Sign Action的前提。

Signset Action: 设置流程（仅对文件），将新的签名流程信息提交到服务器。由于一个签名流程会涉及多个文件，服务器应当不仅对提交设置流程的文件进行更新，也应当考虑所有在签名流程中涉及的文件。

**Sign Request Action: 文件签名（仅对文件），更新文件对应的签名流程中的某一个签名数据包，同时对该文件进行更新请求操作，如果动作成功，接续Upload接口完成**

Contactnew Action: 新建通讯录条目

Contactdel Action: 删除通讯录条目

Contactupdate Action: 更新通讯录条目

Share Action: 拷贝一个文件给他人。由于拷贝对象只是contact id，服务器需要判断对象是否是ESAP用户；是用户，使用复制功能（不是单纯的以Target引用方式，这有别于签名流程中的操作）给对方复制一个文件对象；不是用户，发送邮件提醒。

有关Action的操作，基于接口参数中的[Action Package](#_Action_Package)实现，请注意参考。

## 签名流程

**签名操作必须实时联网才能完成**，主要是避免多人签名操作中可能存在的冲突。签名操作涉及三个Action类型：SignSet和Sign，以及“请求锁定”操作（Lock Action）。该三个Action执行请参考“执行动作流程”中联机方式。

同时，签名之后文件因为已经修改，需要更新上传，该操作将需要调用Upload接口，在其后异步执行。有关签名操作，请详细参考后面的Package和Database相关设计。依据其所具备的数据属性，这里特此说明各个签名流程相关操作的实现建议。

操作：保存签名

操作：编辑签名流程

文件所有权

编辑文件签名流程

添加一个联系人到流程

编辑或删除联系人

用户操作开始

用户完成签署并保存文件

已有流程

用户操作结束

文件只读

准备上传相关文件

打开文件

请求锁定

具备

不具备

成功

失败

创建Sign Request Action

客户端动作

是

否

文件只读

创建SignSet Action

用户将要签署并保存文件

客户端动作

保存签名？

图表 4签名操作逻辑流程

### 签名流程基础概念

1. 每一个签名流程都依附于“一个或多个PDF文件”，以及“一个或多个签名对象”组成。签名对象是联系人的一种业务包装，使其具备记录签署信息的能力。其中，至少有一个签名对象是“发起人”，仅发起人才可以编辑该签名流程。
2. 签名发起人具备流程中所有的PDF文件的拥有权。（即不可将他没有拥有权的PDF文件添加到签名流程）
3. 签名流程中的多个签名对象具备不同的sequence序号值（从1开始）。而签名流程从0开始（表示没有人签署），此时拥有最小sequence序号的联系人可以申请文件的锁定权成功。在一个签名流程中，具有同样sequence数值的sign具备同等签名优先级。
4. 为了避免具有同等优先级签名的用户执行签名操作发生顺序冲突，签名操作必须联网。
5. 签名流程具备一个当前签署状态（并非最终签署状态）。分别为：0未签署 1已签署 2占用中 3拒绝签署。
6. 任何人想要签署一个文件，必须首先获取该文件的锁定权。详情参考“如何获取文件锁定权”。
7. 文件的所有者可以在一个文件没有相关的签名流程时，通过立即签名默认创建一个“单文件、单签名对象”的签名流程。随后该用户可以继续编辑签名流程，但是不可将“已签署”的文件或对象进行编辑，仅可为其追加后继文件或签署对象。
8. 签署人完成一个文件的签署后，需检查还有无其他文件需要签署，仅当所有文件都签署后，方将签署人对象中的签署时间填写以表示“签署完成”状态。

### 如何确定当前文件的签署状态：

1. 对sign\_flow\_package中的内容进行检查，如果此包为空，表示没有设置签名流程信息；
2. 如果已经设置流程，再检查其中current\_sequence是否为0，是0即表示尚未开始签名；
3. 如果已经开始签名，current\_sign\_id必然非空，且可以在signs列表中依据此ID找到对应的sign数据包，然后再检查该sign数据包的contact信息，contact中又可以找到对应的account\_ID（如果没有account\_id则说明该联系人尚未注册ESAP系统平台）。account\_id如果是自己即表示轮到自己签署，然后检查sign\_date和refuse\_date，获知签名或拒绝信息。
4. 遍历所有签署包，如果所有签署包都具备sign\_date属性，表示整个流程签署完成。
5. 无流程/有流程未开始签署/签署中（轮到我/没轮到我）/成功/失败；

### 如何获取文件的锁定权（服务器逻辑）

1. 前提：尚未被任何人锁定的文件（此为文件的默认状态），可以通过比较当前签名流程中的current\_sequence与签名对象的sequence来判断是否能够获得文件锁定权：
   1. 签名流程的current\_sequence数值与自己签名对象的sequence数值相同
   2. 具备签名流程的current\_sequence数值的所有签名对象都已经是签署状态（签署时间非空），我是current\_sequence+1
2. 执行锁定：检查当前签署状态current\_sign\_status，如果是（0未签署 1已签署）状态，即可请求锁定文件（即设置current\_sign\_id为自己，并且设置current\_sign\_status=2占用中）。锁定操作可能需要从服务器更新文件，该更新为一个下载过程，更新时会放弃任何本地已有文件的修改信息。
3. 失效：如果在约定时间内（比如60分钟）用户没有提交签署结果（即置current\_sign\_status=1已签署或者3拒绝签署）,则此刻服务器应当将当前签署状态current\_sign\_status置为0。而后，如果用户再尝试提交签名结果，则返回前提1)重来。
4. 以上1、2两步操作不能通过的话，拒绝用户请求，用户以只读方式访问文件。

### 文件锁定权在同帐户、不同客户端的情况下的处理方案

1. 一般来说，锁定该文件的用户，是处于签名过程中的人（通过签名信息可以查询获得）。但是通过签名操作锁定该文件的用户一般都会有一个操作时限（比如打开文件，锁定文件60分钟后仍未提交签名结果，操作时限是由服务器确定），过了时限，服务器会解除文件锁定（即将文件锁定者ID清空）。
2. 锁定该文件的用户提交了签名结果，但是文件处于上传过程中，该上传文件操作暂时中止，此刻服务器应当延长锁定时限（比如24小时）。
3. 对于同一个用户，可能的一种状况是，他在某一个客户端上锁定了文件，但是上载操作未完成，而后立刻又在另一个客户端上试图再操作该文件。由于文件的锁定者是其本人，客户端应当是允许他继续操作的，但是新的客户端没有这个“上载未完成”的文件版本，服务器收到请求时，需要清除之前上传未完成的内容。

# 接口方法定义

使用Form的Post方法提交以下HTTP链接完成操作。

## Login

**https://IPAddress:Port/esap\_base/rest/example/login**

参数：[Login Request/Reuslt Package](#_Login)

## Action

**https://IPAddress:Port/esap\_base/rest/actionlist/action**

参数：[Action Request/Reuslt Package](#_Action)

## Update

**https://IPAddress:Port/esap\_base/rest/contact/update**

参数：[Update Request/Reuslt Package](#_Update)

## 文件上传下载接口

### Download

**https://IPAddress:Port/esap\_base/rest/document/download\_file?id=**

### Upload

**https://IPAddress:Port/esap\_base/rest/document/upload**

**Request Header**

Method: POST  
Content-Type:multipart/form-data; boundary=……………… **Body**

([boundary]\r\n Content-Disposition: form-data; name="file\_upload"; filename="[filename]"\r\nContent-Type: application/%@\r\n\r\n[filedata\_parts][boundary]--\r\n)

# Package Reference

用于HTTP／HTTPS请求的接口，作为参数使用的数据包，均以JSON格式进行内容的定义。为了避免类型错误带来的异常影响，所有数值都是字符型（包括数字也用字符方式存储）

数据包分为事务包（Transaction Package）和实体包（Entity Package）两类。

Upload Complete  
Interface

Login Request

Action Request

Transaction Package

Update Request

Upload Complete

Login Request

Login Request

Login Request

Login Result

Login Result

Login Result

Upload Complete

Update Result

Action Result

Login Result

Action Interface

Update Interface

Login Interface

图表 5事务包：接口所使用的事务包模型。在每一个接口发送、返回数据的两步过程中，首先要定义事务包。事务包包括：发送数据（Request）包，返回数据（Result）包。每一个Request包都唯一对应一个Result包。在同步操作过程中，可以通过事务的连续、不可分割性质对应Request-Result；但同时考虑到客户端的数据处理潜在的异步处理、复杂性，每个事务包均有一个id字段（采用GUID），用以匹配Request和Result。

Target

File

SignFlow

Sign

SignPen

Action

Contact

Contact Item

Update Result

Action Request

Update Interface

Action Interface

Entity Package

图表6实体包：主要由Action(Request)和Update(Result)引用。而实体包是用于描述事务包中的复杂参数所用。比如Contact、Target、SignPen等数据包。实体包图中示意了各种包之间的关联（方块重叠处表示有对应关联）。

## Login

在Login接口定义中所传递的JSON数据包。该数据包具有特殊性质。在一个完整的接口数据传递过程中，必须要由两部分参数构成：首当其冲的即Login数据包，然后再是接口其他数据（对于单纯的Login接口则仅有Login数据包）。

### Login Request Package

@login\_package(request)  
{  
 “id”: “…”, // 标识该数据包的id，采用GUID  
 “alias”: “112262@qq.com”, // 登录用户名。可以是电子邮件，也可以是用户昵称  
 “type”: “0”, // 用户登录密码加密方式。 0：MD5加密 1：用户私钥加密  
 “password”: “CALAND”, // 用于登录验证的密码，该内容使用MD5加密或用户私钥加密  
 “requireCert”: “0”, // 是否向服务器请求ESAP平台证书。0：不请求 1：请求  
 “deviceId”: “”, // 设备标识（必备）  
 “verifyNumber”: ””, // 验证码，当设备标识没有注册时需要，否则返回结果为0  
 “deviceType”: “”, // 设备类型 0：未知 1：iPad 2：iPhone 3：Android 4：Win8平板 “cert”: “…”, // 用户的ESAP平台证书信息，使用MD5加密  
 “accountId”: “”, // 用户在ESAP平台的用户ID（UserID，整型）  
 “result”: “0” // 登录返回结果。0：需要验证码 1：鉴权成功 2：不存在用户名 3：密码错误 4：设备被拒绝  
}

注：password用于验证用户登录。如果不使用证书登录，password使用MD5进行加密；如果使用证书登录，使用用户私钥加密“**CALAND**”字符（全部大写）得到的密文放在password字段即可（服务器会使用该用户的公钥解析password，检验结果是否是“CALAND”）。

### Login Result Package

@login\_package(result)  
{  
 “id”: “…”, // 标识该数据包的id，采用GUID  
 “alias”: “112262@qq.com”, // 登录用户名。可以是电子邮件，也可以是用户昵称  
 “type”: “0”, // 用户登录密码加密方式。 0：MD5加密 1：用户私钥加密  
 “password”: “CALAND”, // 用于登录验证的密码，该内容使用MD5加密或用户私钥加密  
 “requireCert”: “0”, // 是否向服务器请求ESAP平台证书。0：不请求 1：请求  
 “deviceId”: “”, // 设备标识  
 “verifyNumber”: ””, // 验证码，当设备标识没有注册时需要，否则返回结果为0  
 “deviceType”: “”, // 设备类型 0：未知 1：iPad 2：iPhone 3：Android 4：Win8平板 “cert”: “…”, // 用户的ESAP平台证书信息  
 “accountId”: “”, // 用户在ESAP平台的用户ID（UserID，整型）  
 “result”: “0” // 登录返回结果。0：需要验证码 1：鉴权成功 2：不存在用户名 3：密码错误 4：设备被拒绝  
}

注：返回结果中有用户的ESAP平台证书信息，该证书由ESAP平台发布并提供给客户端使用。仅当请求包中require Cert为1时方返回数据。客户端根据本地缓存状况来决定是否要求服务器发放证书，该证书包括完整的的公、私钥数据。

**\*关于request和result格式：从上图可见，一对儿请求和结果的JSON包可以使用同一解析器进行解析，只是发送和接收时，关注的字段有所不同。在本文档中，灰色的字段表示可以此包内加入的数据，但是在处理时一般予以忽略。后文亦会做同样处理，供参考。**

## VERIFY

在Verify接口定义中所传递的JSON数据包，客服端请求绑定设备，服务器需向用户的手机或邮箱发送验证码，用以验证设备绑定是否成功.

### Verify Request Package

@verify\_package(request)  
{  
 “accountId”: “”, // 用户在ESAP平台的用户ID（UserID，整型）  
 “deviceId”: “”, // 验证码是对UserID和DeviceID的一个配对绑定，必须有这个参数  
}

### Verify Result Package

@verify\_package(result)  
{  
 “accountId”: “”, // 用户在ESAP平台的用户ID（UserID，整型）  
 “deviceId”: “”, // 验证码是对UserID和DeviceID的一个配对绑定，必须有这个参数  
 “verifyAddress”: “”, // 激活码发送到的地址，比如一个手机号码，135\*\*\*\*7732，或邮件等等  
 “result”: “0” // 验证返回结果。0：验证失败 1：验证成功  
}

## Action

在Action接口定义中所传递的JSON数据包，定义如下：

### Action Request Package

@action\_list\_package(request)  
{  
 "login": {@login\_package(request)}, // login头请求  
 "actions": [{@action\_package(request)}, …] // Action列表，由0~n个action包组成  
}

### Action Result Package

@action\_list\_package(result)  
{  
 "login": {@login\_package(result)}, // login头结果  
 "actions": [{@action\_package(result)}, …] // Action列表，由0~n个action包组成  
}

Action接口在客户端联网时，所有的Actions（即Action list）均只有1个Action，实时交互客户端和服务端数据；当客户端脱机操作时，方会一次性提交多个Action到服务器。

### Action Package

@action\_package(request)  
{  
 "id": “”, // 标识GUID  
 “timestamp”: “”, // 时间标签  
 "version": “”, // 版本信息**①** "category": "undefined", // Action的类型**③**  
 "orgData": {}, // Action参数1**③**  
 "newData": {}, // Action参数2**③**  
 “actionResult”: “0” // 返回结果。响应状态。0-拒绝，1-接受，2-挂起**②**  
}

1. 仅当action的target属性（target节点ID）对应的target为文件类型（type＝2）时有意义。该节点内容为一非负整数标记处理文件的版本信息。
2. 判断Action结果时，该Action无论是否执行成功，客户端都应予以删除不再次使用。但要注意，状态为2时（挂起），则表示该Action操作已经被服务器认可，但仍需执行某些后继操作（比如上传、下载等异步操作）。客户端需要继续准备调用其他接口才可以真正的完成所有操作。

@action\_package(result)  
{  
 "id": “”, // 标识GUID  
 “timestamp”: “”, // 时间标签  
 "version": “”, // 版本信息**①**  
 "category": "undefined", // Action的类型**③**  
 "orgData": {}, // Action参数1**③**  
 "newData": {}, // Action参数2**③**  
 “actionResult”: “0” // 返回结果。响应状态。0-拒绝，1-接受，2-挂起**②**  
}

1. 根据Action Category的不同，对应的orgData和newData数据对应不同的实体包(entity package)分别如下表所示。主要是四类实体类型：分别为Contact类型、Target类型、以及Sign类型、SIgnPen类型。

|  |  |  |  |
| --- | --- | --- | --- |
| **category** |  | **orgData** | **newData** |
| undefined | 未定义 | － | － |
| create | 新建 | － | target数据包 |
| delete | 删除 | 原target对应id | － |
| rename | 重命名 | target原文件名 | target数据包 |
| move | 移动 | target原parentId | target数据包 |
| **update\_request** | **更新文件（仅对文件）** | **－** | **target数据包** |
| lock | 锁定文件（仅对文件） | 原target对应id | － |
| signset | 设置流程（仅对文件） | 原target对应id | signFlow数据包 |
| **sign\_request** | **文件签名（仅对文件）** | **原target对应id** | **sign数据包** |
| signpennew | 更新签名图片 | － | pen数据包 |
| signpendel | 删除签名图片 | 原pen对应id | － |
| contactnew | 新建通讯录条目 | － | contact数据包 |
| contactdel | 删除通讯录条目 | 原contact对应id | － |
| contactupdate | 更新通讯录条目 | 原contact对应id | contact数据包 |
| share | 拷贝一个文件给他人 | 目标contact\_id | target数据包 |

### Contact Package

@contact\_package  
{  
 “id”: “”, // 联系人ID  
 “familyName”: “”, // 联系人的姓氏  
 “personName”: “”, // 联系人的名字  
 “gender”: “0”, // 联系人的性别 0：男性 1：女性  
 “lastTimeStamp”: “”, // 更新最后时间  
 “contactItems”: [{@contact\_item}, …] // 联系人的详细条目列表  
}

### Contact Item Package

@contact\_item\_package  
{  
 “id”: “”, // 条目ID  
 “accountId”: “”, // 可能对应的平台账户ID  
 “type”: “”, // 条目类型 0：邮件 1：电话 2：文本  
 “title”: “”, // 条目标题  
 “content”: “”, // 条目内容  
 “major”: “0” // 是否主要联系方式 0：否 1：是  
}

### Target Package

@target\_package  
{  
 “id”: “”, // Target ID  
 “parentId”: “”, // Target父节点ID  
 “displayName”: “”, // 显示名称  
 “type”: “2”, // 0：系统文件夹 1：文件夹 2：文件  
 “file”: {@file\_package}, // 对应文件包 （type=2时有效）  
}

### File Package

@file\_package  
{  
 “id”: “”, // File ID  
 “ownerAccount”: “”, // 拥有者Account ID  
 “lockerAccount”: “”, // 锁定者Account ID  
 “localVersion”: “1”, // 文件本地版本号信息。每次签署提交、修改文件都应使版本+1  
 “serverVersion”: “”, // 文件服务器版本信息。该信息由服务器返回提供  
 “storeType”: “0”, // 0：未加密 1：对称加密 2：非对称加密 3：联网签署  
 “type”: “1”, // 0：未知 1：PDF文件 2：文本 3：图像  
 “signFlow”: {@sign\_flow\_pakage} // 对应签名流程（type=1时有效）  
}

### Sign Flow Package

@sign\_flow\_package  
{  
 “id”: “”, // Sign Flow ID  
 “starterAccount”: “”, // 签名发起人账户ID  
 “currentSequence”: “1”, // 当前签名流程中走到的顺序号，0：未开始 1+：进行到的号  
 “currentSignId”: “”, // 当前激活的签名包ID  
 “currentSignStatus” “”, // 当前激活签名包的签署状态0未签署1已签2占用3拒签  
 “signs”: [{@sign\_package}, …] // 该流程所有的sign节点  
}

**当前用户(start\_Account)如何确定通过signs和files属性共享签名文档给他人：**

当前用户选择其拥有的一个PDF文档发起签名（创建签名流程）后，signs中至少创建其自己一个用户的sign，files也至少包含一个文件。而后他可以选择更多的人、更多的文件（单人签名、单个文件也都是可以存在的）加入到流程。注意，用户发起签署时，仅能选择owner\_account为自己的文件。

服务器接收到这样一个sign\_flow\_package时，需要遍历signs和files，为sign中的每一个account\_id生成target引用到每一个file（当前用户已经拥有这些target，不需要重复创建）。

### Sign Package

@sign\_package  
{  
 “id”: “”, // Sign ID  
 “sequence”: “1”, // 该签名包的的顺序号，从1开始  
 “signDate”: “”, // 签署时间  
 “refuseDate”: “”, // 拒签时间  
 “signerAccountID”: “”, // 对应的签名人的帐号ID（如果有）  
 “signerName”: “”, // 对应的签名人显示名称**①**  
 “signerAddress”: “” // 对应的签名人联系方式**①**  
}

1. 根据服务器传来的sign数据包，可以获取签名人信息包。签名人来自于发起人的通讯录，但是并不维系完整的通讯录信息，仅保存一个显示名称和联系地址（一般选择该联系人的首要地址，如果存在）；当明确知道该联系人也是ESAP平台用户时，还会包含一个Signer Account ID。接收到签名信息的用户，如果该联系人具有Account ID并且该Account ID在当前用户通讯录中已经存在，需要执行数据合并操作；如果该联系人的signer Address与当前用户通讯录中某联系人相同，则认为是同一个人，但是不执行数据合并操作，客户端显示该签署人时，应当以当前用户通讯录中的数据为标准显示（别人说这个文件要跟“老刘”签名，“老刘”在我的通讯录中叫“小刘”，那么我这边还是显示“小刘”。如果该联系人ID和majorEmail都不存在，则在当前用户通讯录中新建一条联系人信息，该联系人ID需要重新生成。

### Sign Pen Package

@sign\_pen\_package  
{  
 “id”: “", // 签名图ID  
 “cert”: “”, // 签名图关联的证书信息  
 “rule”: “0”, // 签名图表达的含义。 0：同意（默认） 1：拒绝  
 “picUrl”: “” // 存在服务器的URL地址  
}

相比较文件的上传下载需要额外判断Action返回结果中的文件版本号，然后再另调用接口完成，签名图的上传下载，直接在Action后（上传）和Update后（下载）进行，无需检查版本。故签名图没有更新操作，用户要抛弃一个旧的签名图，只能删除后重建。sign pen packagE只有在Update操作获得全部签名图后会有picURL字段，用以立即下载签名图。客户端需要自行判断是否要更新签名图片的逻辑。

## Update

所有的Action操作，用户仅能向服务器提出“动作请示”，然而，实际执行结果，未必会如客户端所愿，这是因为服务端要处理多客户端的请求，这些请求可能是同一个用户在不同平台上发起的，逻辑上存在一定矛盾的。服务器处理Action需要遵循一定的准则。

最终数据的结果应当通过Update接口，从服务器返回实际的最新结果。

在Update接口中定义的JSON数据包如下。

### Update Request Package

@update\_package(request)  
{  
 “login”: {@login\_package(request)}, // Login头请求  
 “type”: “0”, // 请求类型。 0：全部更新 1：更新通讯录和文档 2：更新签名图  
 “contact”: {@contact\_update(request)}, // 请求的通讯录内容  
 “target”: {@target\_update(request)}, // 请求的文档内容  
 “pen”: {@sign\_pen(request)} // 请求的签名图内容  
}

### Update Result Package

@update\_package(result)  
{  
 “login”: {@login\_pacakage(result)}, // Login头请求  
 “type”: “0”, // 请求类型。 0：全部更新 1：更新通讯录 2：更新文档  
 “contact”: {@contact\_update(result)}, // 请求的通讯录内容  
 “target”: {@target\_update(result)} // 请求的文档内容  
 “pen”: {@pen\_update(result)} // 请求的签名图内容  
}

### Contact Update Package

@ contact\_update  
{  
 “updateId”: “”, // 指定要更新的联系人ID  
 “contacts”: [{@contact\_package}, …] // 返回数据  
}

Request时只需要提供updateId字段（如果不提供，表示要求获取整个通讯录更新）

Result的包中只需要包括contacts字段及其内容即可。

### Target Update Package

@ target\_update  
{  
 “targetId”: “”, // 指定要更新的对象ID  
 “requireType”: “0”, // 更新结构的标识**①**  
 “targets”: [{@target\_package}, …] // 返回数据  
}

Request时需要提供targetId和requireType字段。不提供targetId表示要更新当前用户根目录

Result的包中只要包括targets字段及其内容即可。

1. requireType默认为0。如果Target是文件对象，则无须考虑这个字段；如果是文件夹，各个取值的含义是：  
   0：以递归方式请求指定target之下其所有子对象（不含当前Target）  
   1：仅获取当前Target之下一层的所有子对象（不含当前Target）  
   2：仅获取当前Target之下一层的文件夹对象（不含当前Target）  
   3：仅获取当前Target对象

### Pen Update Package

@ pen\_update  
{  
 “pens”: [{@sign\_pen\_package}, …] // 需要更新的签名图列表  
}

签名图更新无需指定要更新的ID，一定是完整更新。

## Upload Complete Package

Upload Complete接口用于客户端通知服务器一个文件已经上传成功。定义的JSON数据包为

### Upload Request Package

@upload\_complete\_package(request)  
{  
 “login”: {@login\_pacakage(request)}, // Login头请求  
 “completeType”: “0”, // 完成的上传类型。0：文件上传 1：签名图上传  
 “completeId”: “”, // 完成的上传数据的ID  
 “completeUrl”: “”, // 上传完成对应的Url，供服务器找到对应数据  
}

### Upload Result Package

作为一个状态通知，该操作不需要特定的结果返回。

@upload\_complete\_package(result)  
{  
}

# Database Reference

这一部分是数据库设计的参考。所有的功能基于以下给出的数据设计。由于后台服务的复杂行，该部分的数据未必与服务器承载的数据库格式完全一致，但这部分数据可以视为服务器数据的一个重要子集。

关于当前设计局限性声明：以通讯录为例，当前客户端的数据设计并不是最臻完善的状态。

比如，目前的设计中的通讯录并不是万全，用户姓、名区分，但没有昵称、西方人名中的MIDDLE NAME等更复杂的划分，也不存储联系人个性头像，客户端可以根据性别使用两种默认头像，也可以仅作本地的头像保存信息（即在网络通讯中不传输头像图片信息）

## Client Account

客户端用户表——用以保存当前登录的用户

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| name | string | - | 用户名 |  |
| password | string | - | 用户密码 |  |
| major\_email | string | - | 用户首要电子邮件地址 | 该邮件地址也可以用于登录； |
| cert | string | null | 用户证书信息 | 可考虑暂时仅使用用户公钥数据 |

该表格仅存储最小必要的用户信息

## Client Contact

客户端联系人表——用以保存基于用户视角的地址本

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| account\_id | unique\_identifier | - | 拥有该联系人信息的用户 | Foreign Key |
| family\_name | string | null | 该联系人的姓氏 |  |
| person\_name | string | null | 该联系人的名字 | 即First Name |
| gender | int | 0 | 该联系人的性别 | 0：男性  1：女性 |
| last\_timestamp | datetime | null | 联系人最后执行更新时间 | 供Action参考使用 |
| record\_status | int | 0 | 该联系人的存储状态 | 0：已经删除  1：有效 |

## Client Contact Item

客户端联系人条目表——用于存储联系人的详细信息

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| Id | unique\_identifier | - |  | Primary Key |
| contact\_id | unique\_identifier | - | 拥有该信息条目的联系人 | Foreign Key |
| account\_id | unique\_identifier | null | 该信息条目对应的用户账户 | Foreign Key**①** |
| type | int | 0 | 条目类型 | 0：email  1：电话号码  2：文字信息 |
| title | string | - | 条目标题 | 描述条目的属性 |
| content | string | - | 条目内容 | 描述条目的数值 |
| major | bit | 0 | 是否是主要条目 | 0：非主要条目**①**  1：主要条目 |

1. 每个用户的联系人信息都有可能是另一个甚至多个ESAP用户，这取决于其联系条目中的联系地址，是否对应了系统的联系人账户account\_id；当account\_id非空时，即证明拥有该联系条目的联系人至少是一个ESAP用户；用户可以自行在一个联系人的多个联系方式中，选择是ESAP平台账户的地址中的一个作为主要联系方式，通过major字段确定。当一个联系人没有任何一个联系方式是ESAP平台账户的时候方可选择其他联系方式作为首要地址，并通过这个地址发起签署和邀约。

## Client Target

客户端对象表——用以保存对应用户的私人文件夹及文件的目录结构。

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| parent\_id | unique\_identifier | null | 父节点的ID | Foreign Key |
| account\_id | unique\_identifier | null | 所属用户的ID | Foreign Key |
| file\_id | unique\_identifier | null | 对应文件实体的ID | Foreign Key**①** |
| display\_name | unique\_identifier | “new target” | 显示在界面上的名称 | 根据树的结构和节点显示名称可以用递归方法生成完整路径 |
| last\_timestamp | datetime | null | 对象最后执行更新时间 | 供Action参考使用 |
| type | int | 2 | 用以标识文件或文件夹 | 0：系统文件夹（不可修改，仅由系统预先内置，参考Client Target System）  1：用户自定义文件夹  2：文件 |
| create\_time | datetime | null | 目标创建时间 | 与物理文件保持一致**①** |
| update\_time | datetime | null | 目标最后更新时间 | 与物理文件保持一致**①** |
| record\_status | int | 0 | 删除标记位 | 0：已经删除  1：正常 |

1. file\_id, create\_time, update\_time三个字段，仅当type=2（文件目标）时方有效

## Client Target System

这里罗列了Client Target表格中，固定设置好的系统文件夹类型对象实际数据。未在表格中的数据均取默认值。

| **id** | **parent\_id** | **account\_id** | **display\_name** | **type** | **record\_status** |
| --- | --- | --- | --- | --- | --- |
| 311BAE9-7449-45BD-855A-F33E1E534A70 | null | null | 收件箱 | 0 | 1 |
| 1DC5F472-16AD-4C93-A849-F05A4ED5BC5C | null | null | 已发送 | 0 | 1 |
| D1245350-4373-46B9-899B-A148F5C71CB0 | null | null | 草稿箱 | 0 | 1 |
| 7D0273A8-09CD-483E-B5C1-81A7B9AA52AF | null | null | 已删除 | 0 | 1 |
| 00000000-0000-0000-0000-000000000000 | null | null | 我的合同**①** | 0 | 1 |

1. 客户端实现时，应仅允许用户在“我的合同”文件夹之内创建自定义文件夹。

## Client File

客户端文件表——用以保存各种文件对象实体。

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| owner\_account\_id | unique\_identifier | - | 文件所有者ID | Foreign Key**①** |
| **locker\_account\_id** | **unique\_identifier** | **null** | **当前锁定文件者ID** | **Foreign Key②** |
| total\_size | int | 0 | 文件实际大小 | 单位为字节数 |
| download\_size | int | 0 | 已下载字节数 | 单位为字节数 |
| upload\_size | int | 0 | 已上传字节数 | 单位为字节数 |
| physical\_filename | unique\_identifier | null | 文件实际名称 | 文件存储在磁盘上的实际位置和名称。**③** |
| trans\_filename | unique\_identifier | null | 传输中的文件名字 | **③** |
| server\_version | int | 1 | 文件在服务器上的版本号 | 基于1增长的整数 |
| local\_version | int | 1 | 本地文件的版本号 | 基于1增长的整数 |
| version\_guid | unique\_identifier | null | 每次修改文件的随机号 | **④** |
| store\_type | int | 0 | 文件存储的类型 | 0：普通未加密存储  1：对称加密存储  2：UKEY非对称加密  3：联网签署 |
| file\_type | int | 0 | 文件类型的枚举 | 0：未知类型  1：PDF文件  2：纯文本文件  3：图片文件 |
| sign\_flow\_id | unique\_identifier | null | 对应的签名流程ID | Foreign Key**⑤** |
| sign\_status | int | 0 | 当前签署状态 | 0：没有签名状态**⑤**  1：所有人都未签名  2：部分人已经签署  3：所有人都已经签署 |
| record\_references | int | 0 | 删除标记位 | 0：没有任何Target引用  1+：被Target引用的数量 |

1. 文件也有一个所有者id。当一个文件使用Target引用方式（一般是通过发起签署方式Sign Set Action）共享给多人时，服务器上该文件只有一个拷贝，未发生复制，仅所有者可以对其发起签署。如果要拷贝给其他用户一个文件（Share Action，需要执行服务器上文件对象的复制。服务器上的文件和客户端文件之间采用“同步”的概念，我们认为两者是“同一个”文件。
2. 文件有一个锁定者ID。锁定发生的情况有两种：某个客户端进行签约声称的锁定，以及上传文件到服务器发生的锁定。
3. physical\_filename可使用GUID作为文件名，该对象没有所有者属性，用户访问需要有Target引用。trans\_filename则是一个中间状态暂存文件的字段。对于服务器来说，一个文件upload未完成时，文件先放在这个字段，仅当上传完毕，才使用该字段对应文件取代physical\_filename对象；对于客户端，该字段应当是对应download未完成的文件。即**上传、下载操作未完成时，不应当先删除原有文件。**
4. verison\_guid是一个文件版本产生的随机号，用以区别不同客户端可能产生的同一个版本数字。它仅具备一个区别作用，以避免同一个用户在多个客户端将版本号相同但实际内容不同的文件作为断点续传的方式上传到服务器。
5. sign\_flow\_id给出了该文件绑定的签名流程对象。通过签名流程对象可以获取具体的签名状态；sign\_status则是一个概要性的状态信息。实际应用中，可能需要通过这两个字段以及当前用户的信息综合判断实际的文件签名状态。

## Client Sign Flow

签名流程表——用以描述一个签名流程，一个签名流程由多个签名组组成。

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| current\_sign\_id | unique\_identifier | - | 当前进行到的签名对象 | Foreign Key |
| current\_sign\_status | int | 0 | 当前签名对象的签名状态 | 0：未签署  1：已签署  2：占用中  3：拒绝签署 |
| current\_sequence | int | 0 | 当前进行到的签名序号 |  |
| record\_status | int | 0 | 删除标记位 | 0：已经删除  1：有效 |

## Client Sign

签名对象表——用以标识一个签名实例对象。

| **Field** | **Data type** | **Default Value** | **Description** | **Usage** |
| --- | --- | --- | --- | --- |
| id | unique\_identifier | - |  | Primary Key |
| sign\_flow\_id | unique\_identifier | - | 所属签名流程ID | Foreign Key |
| sequence | int | 1 | 该签名的顺序号 | 同序号为一组 |
| sign\_displayname | string | - | 该签名块对应人的显示名称 |  |
| sign\_address | string | - | 该签名块对应的联系人地址 | Foreign Key |
| sign\_date | datetime | null | 同意签署日期 |  |
| refuse\_date | datetime | null | 拒绝签署日期 |  |
| sign\_account\_id | unique\_identifier | null | 该签名块对应的联系人的ESAP账户ID | 要执行签署或者拒绝签署，该字段不能为空**①** |

1. 如果该sign\_account\_id为空，说明该签名对象对应的联系人尚不是ESAP平台用户。该联系人注册成功之前，该签名实例对象不可能sign\_status=1。一旦该联系人注册用户成功，服务器应执行一系列的操作，给所有联系人对应的Sign对象的sign\_account\_id进行赋值更新）。