1. 应用状态
   1. 只有当前应用是激活的状态，循环在跑；其他背景应用只接收管道传过来的应用切换和CALLfunc的管道事件并处理，其他事件一概屏蔽；
   2. 应用主循环采用类aie\_main的方式，在主循环中处理wait\_event事件；
2. 事件
   1. 定时器事件：在当前应用和背景应用中都允许打开，但是当接收到休眠广播事件后，关闭定时器；
   2. PIPE事件：在当前应用和背景应用中都必须允许；
      1. 背景应用中，只对ECHO,ACK,SWITCH APP,CALLFUNC命令感兴趣；
   3. KBD,MAG CARD,ICC CARD,BAR,COM,CLOCK等事件：在当前应用是允许的，背景应用不允许；
3. 应用切换
   1. 应用切换规则：
      1. 应用切换必须指定切入方的应用名称；
      2. 应用切换只允许在主控应用和子应用之间进行进行；
      3. 应用切换与通讯无关，无需重新关闭和打开通讯通讯设备；
   2. 应用切换的流程：
      1. 切出方检测到应用切换要求，发起ECHO指令；
      2. 切入方在指定时间内对切出方的ECHO指令进行确认，通过管道返回ACK指令，切入方继续等待wait\_event；
      3. 切出方在指定时间收到了对方的ACK指令，则释放资源，通过管道发送SWITCH APP指令，释放定时器，进入wait\_event（只响应管道事件中的ECHO,SWITCH APP和CALLFUNC指令）；
      4. 切入方接收到SWITCH APP指令后，打开资源，启动定时器，调用idle\_screen函数；设置wait\_event等待所有事件（PIPE,EVENT,TIMER,KEYBOARD,MAGIC CARD,ICC,BAR,COM PORT,CLOCK）并进入wait\_event
   3. 异常的处理机制：
      1. 切入方未收到ECHO指令：
         1. 切出方的等待ACK超时处理；切出方退回到调用处，调用失败
         2. 切入方继续其流程或继续wait\_event
      2. 切入方此时在处理其他事物，未能及时收到ECHO指令：
         1. 切出方的等待ACK超时，退回到切换处，切换失败
         2. 切入方处理完事物后，返回wait\_event，接收到ECHO指令，发生ACK指令；
         3. 切出方已经退出切换：
            1. 切出方退回到其他函数中，没有wait\_event，收不到ACK指令，不处理；
            2. 切出方退回到wait\_event中，收到ACK指令，但是不在SWTICH APP流程里，不发生SWTICH APP指令；
            3. 切出方再次进入SWTICH APP流程，在发送ECHO之前清空pipe时：

PIPE中的ACK被清除，无影响。

清空PIPE后收到第一个ACK，切出方无法识别是响应哪个ECHO指令，认为是第二次ECHO的响应，则发送SWITCH APP指令，切入方又发送ACK指令，切出方在发送SWITCH APP指令后，已经进入wait\_event，只对ECHO,SWITCH APP和CALLFUNC指令感兴趣。

* + 1. 切出方释放资源失败，则重新打开资源，返回到切出方应用wait\_event(此时未将定时器和事件重置)；
    2. 切入方打开资源失败，则不停尝试再次打开资源；（是否需要循环重试N次h后切换回主控或切出方？）
  1. 应用的状态转变：
     1. 切换前：
        1. 切出方：
           1. 事件：响应PIPE,TIMER,KBD,MAG CARD,ICC CARD,BEEPER,CLOCK
           2. 设备：占用并打开所有设备
           3. 定时器：开启
           4. 界面：显示IDLE SCREEN
        2. 切入方：
           1. 事件：响应PIPE
           2. 设备：所有设备被关闭
           3. 定时器：关闭
           4. 界面：无
     2. 切换后：
        1. 切出方：
           1. 事件：响应PIPE
           2. 设备：所有设备被关闭
           3. 定时器：关闭
           4. 界面：无
        2. 切入方：
           1. 事件：响应PIPE,TIMER,KBD,MAG CARD,ICC CARD,BEEPER,CLOCK
           2. 设备：占用并打开所有设备
           3. 定时器：开启
           4. 界面：显示IDLE SCREEN
  2. SWITCH APP需要的管道指令
     1. ECHO
     2. ACK
     3. SWITCH APP
  3. SWITCH APP需要传递给切入方的管道数据
     1. 切出方应用名？
     2. 切入方应用名？
  4. 主控和子应用的切换：
     1. 主控通过菜单或指定默认应用的方式切换到子应用；
     2. 子应用通过热键或超时方式切换到主控应用；
     3. 子应用只能切换到主控应用；

1. CALLFUNC：
   1. CALLFUNC规则：
      1. CALLFUNC必须指定被调用方的应用名称；
      2. CALLFUNC无论调用方还是被调用方的原因导致的执行失败或成功，都应回到调用方；
      3. CALLFUNC允许在各个应用之间进行调用，只需指定被调用方的应用名称（允许子应用间的互相调用；主控应用对子应用的调用；子应用对主控应用的调用）；
   2. CALLFUNC流程：
      1. 调用方发起ECHO指令；
      2. 被调用方在指定时间内对调用方的ECHO指令进行确认，通过管道返回ACK指令，被调用方继续等待wait\_event；
      3. 调用方在指定时间收到了对方的ACK指令，则释放资源，通过管道发送CALLFUNC指令和具体的调用数据,并等待wait\_event（只等待PIPE消息中的CALLFUNC RESULT指令，其他事件和指令忽略）。
      4. 被调用方接收到CALLFUNC指令后，开启资源，根据函数定义表调用应用相关处理函数；
      5. 被调用方处理完对应函数后，释放资源，并将函数处理结果和返回数据通过CALLFUNC RESULT返回给调用方；
      6. 调用方开启资源，返回调用结果；
   3. CALLFUNC采用管道确认机制来保证请求正确发送到被调用方：
      1. 首先调用方发起ECHO指令；（之前清管道，防止过期数据干扰）
      2. 被调用方在指定时间内对调用方的ECHO指令进行确认，通过管道返回ACK确认指令；
      3. 如果调用方在指定时间内没有接收到被调用方的ACK指令则认为此调用无效，退回到原应用中。（是否需要循环重试N次？）
   4. 异常的处理机制：
      1. 被调用方未收到ECHO指令：
         1. 调用方的等待ACK超时处理；调用方退回到调用处，调用失败
         2. 被调用方继续其流程或继续wait\_event
      2. 被调用方此时在处理其他事物，未能及时收到ECHO指令：
         1. 调用方的等待ACK超时，调用方退回到调用处，调用失败
         2. 被调用方处理完事物后，返回wait\_event，接收到ECHO指令，发生ACK指令；
         3. 调用方已经退出调用：
            1. 调用方退回到其他函数中，没有wait\_event，收不到ACK指令，不处理；
            2. 调用方退回到wait\_event中，收到ACK指令，但是不在CALLFUNC流程里，不发生CALLFUNC指令
            3. 调用方再次进入CALLFUNC流程，在发送ECHO之前清空pipe时：

PIPE中的ACK被清除，无影响。

清空PIPE后收到第一个ACK，调用方无法识别是响应哪个ECHO指令，认为是第二次ECHO的响应，则发送CALLFUNC指令，被调用方又发送ACK指令，调用方在发送CALLFUNC指令后，只对CALLFUNC RESULT感兴趣，其他的接收到后抛弃。

* + 1. 调用方释放资源失败，则重新打开资源，返回到调用处，不发送CALLFUNC指令；
    2. 被调用方打开资源失败，则关闭所有资源，发送CALLFUNC RESULT指令，返回结果为失败；
    3. （被调用方无法接收到CALLFUNC指令 || 被调用方返回的CALLFUNC RESULT指令，调用方未接收到）调用方等待CALLFUNC RESULT无果，导致调用方死等：（？？？？？）
  1. 被调用方在CALLFUNC时，可能处于两种状态：
     1. Wait\_event等待状态：此时接收到调用方发过来的ECHO，立刻回应ACK指令；并继续进入Wait\_event等待状态，等待执行CALLFUNC指令；
     2. 应用在处理其他事物过程中：当调用方发过来的ECHO指令不能被立即处理；当应用处理完其他事物返回到wait\_event状态后，虽然收到ECHO指令并回应ACK，但是因为调用方已经因为超时退出CALLFUNC处理流程，因此不会再发生CALLFUNC指令。所以被调用方需要继续返回到wait\_event中。
  2. CALLFUNC需要传递给被调用方的管道数据
     1. 调用方应用名？
     2. 被调用方应用名？
     3. 附件数据及长度
  3. CALLFUNC RESULT需要传递给调用方的管道数据
     1. 调用方应用名？
     2. 被调用方应用名？
     3. 附件数据及长度
  4. 当前应用通过管道传递CALLFUNC的请求，CALLFUNC传递无类型型数据及有效数据长度，具体的数据内容和格式由应用自己定义，但不能超过管道所能承载的最大长度。
  5. CALLFUNC需要的管道指令
     1. ECHO
     2. ACK
     3. CALLFUNC
     4. CALLFUNC RESULT

1. 设备管理
   1. 此处所指的设备不包括通讯库占用的设备；
   2. 应用无需知道和关系与设备相关的操作；
   3. 当前机型可用的设备由库来判断哪些设备是可用的；
   4. 设备只允许当前应用使用；
   5. 在应用切换时，切出方需要关闭所有设备，切入方需要打开所有设备；
2. 管道
   1. 管道操作：
      1. 管道能够读写的数据缓冲设置为2048（对于要传递超过管道限制的数据，可以采用共享文件或其他方式）
      2. 发送之前清管道，防止过期数据干扰
      3. 在接收到管道数据时，判断接收方的taskid是否与管道指令要求的一致；
      4. 对于有时间戳的指令，判断是否已经超过有效期，如果逾期，则抛弃不响应
   2. 管道数据：
      1. 发起方管道ID
      2. 发起方应用taskid
      3. 接收方应用taskid
      4. 管道指令
      5. 时间戳有效期（事务型指令为有效期tick，广播型值为0）
      6. 附加数据及长度
   3. 管道指令分类：
      1. 事物型指令：指令在一段时间内是有效的，超过这个时间再响应可能会造成混乱。
         1. ECHO
         2. ACK
         3. SWITCH APP
         4. CALLFUNC
      2. 广播型指令：无论何时收到指令都有效，不要求立即响应或者不要求响应
         1. BROADCAST
         2. LOGPRINT
         3. POWER SAVE
   4. 管道创建：
      1. 应用启动时创建同名管道；
      2. 全部采用消息型管道；
   5. 管道方式的处理：
      1. 对于多对一的方式，采用匿名管道的方式（如LOGPRINT）
      2. 对于一对多的方式，采用有名管道遍历的方式（如BROADCAST）
3. 省电模式
   1. 省电模式的进入：
      1. 省电模式由内部应用主循环处理；
      2. 省电模式的触发时间内部定义，主控设置G1的\*POW=1；
      3. 省电模式的触发条件：1.终端类型；2.当前应用通知主控发起休眠广播事件；
      4. 所有应用（含当前应用和背景应用）接收到广播事件后，将定时器取消；
   2. 省电模式的取消和当前应用状态恢复：
      1. 系统将省电模式转为工作模式的条件：
         1. 定时器事件；（省电模式下定时器都被清除，不会发生）
         2. GREEN KEY被按下；（触发EVT\_KBD事件）
         3. 低电量；
         4. 插入充电器；（触发EVT\_SYSTEM事件）
      2. 系统将省电模式转为工作模式后，wait\_event会得到一个事件：
         1. 因此可以在wait\_event后重置timer
         2. Wait\_event后继续运行主循环，显示当前应用logo等信息
4. 应用启动和注册
   1. 启动：
      1. 主应用的启动：由系统\*GO决定
      2. 子应用的启动：
         1. 主应用在启动成功并创建主应用管道后，遍历GROUP中的.out文件（RAM和FLASH），对寻找到的.out文件都run；
      3. 如果子应用未检测到主应用（如注册失败），则子应用直接退出，不允许运行
   2. 子应用注册：
      1. 子应用注册方法1：
         1. 主应用在启动时创建主应用管道P:MAINAPP
         2. 子应用在启动时创建各自子应用管道P:[APPNAME]
         3. 子应用通过get\_owner得到主应用管道ID，并通过管道传递给主应用相关子应用信息；
      2. 子应用注册异常：
         1. 子应用被运行起来，但是只有注册成功后才正常运行，否则退出（ \_exit）：
         2. 主应用在启动完子应用后，等待接收管道注册信息，等待超时根据启动起来的子应用数目动态调整。
5. 默认应用的设置：
   1. 主应用通过函数设定默认应用
   2. 如果默认应用为空或默认应用不存在，则显示应用列表
   3. 进入默认应用前，等待2秒，此时可以按键进入应用列表或其他菜单
6. 文件的生成、签名规则：
   1. 压缩包的文件命名规则按照之前的方式进行（A\_XXX\_YYY\_Z.ZIP/D\_XXX\_YYY\_Z.ZIP）
   2. 程序文件签名时，需将签名证书按照签名IC卡的ID号作为文件名，并存放到压缩包中与p7s同位置的目录中。如果目录中以前有其他IC卡的签名证书，则删除用当前的替换。
7. 应用的版本管理
   1. 版本信息是在SMK里进行定义；
   2. 在编译时生成应用同名的版本号文件，存放版本信息
   3. 应用在启动后，读取版本信息文件获得版本号，并通过注册的方式传递给主控应用
8. 应用的下载
9. 子应用的管理：
   1. 主应用应该保存：
      1. 子应用信息列表：
         1. 应用名：（from 注册信息）
         2. 应用显示名：（from 注册信息）
         3. 应用版本：（from 注册信息）
         4. 应用管道ID：（from 注册信息）
         5. 子应用是否可见；
         6. 子应用是否生效；
      2. 当前应用
      3. 默认应用

主控程序逻辑：

1. 配置事件驱动表，包括内容：
   1. 定时器事件
   2. Scan事件（只处理功能）
   3. Redraw事件（只处理屏幕）
   4. 按键事件（只对配置好的热键进行响应）
   5. Callfunc事件
   6. 刷卡事件
   7. 插卡事件
   8. 通讯口事件（不要）
2. main启动，初始化（库函数）
   1. 打开设备
   2. 创建管道
   3. 查询子应用
   4. 统计子应用数量和信息
   5. 创建子应用列表数组
3. 设置定时器（库函数）