

INPAS System Module

Для разработчиков

Версия 1.0

История изменений документа

Версия	Дата	Перечень изменений
1.0	17.07.2020	Первоначальная версия

Содержание

1. Описание архитектуры платежного приложения «Unipos Terminal»	4
1.1. Описание уровней архитектуры платежного приложения «Unipos Terminal»	5
2. Уровень аппаратной абстракции и дополнительные библиотеки.....	7
2.1. Описание модулей	7
2.1.1. Abstract Layer Module	7
3. Приложения	10
3.1. Компонентная модель ПО «Unipos»	10
3.2. Компонентная модель системных библиотек «Unipos»	11
3.3. Список заголовочных файлов	12

1. Описание архитектуры платежного приложения «Unipos Terminal»

Архитектура платежного приложения «Unipos Terminal» является многоуровневой и строится по принципу разделения кода на логические уровни, отвечающие за определенную категорию функционала. Каждый уровень имеет определенный и фиксированный интерфейс, что позволяет изменять конечную реализацию логического уровня без модификации других уровней.

Функции одного уровня, используемые несколькими загруженными приложениями, располагаются в отдельных библиотеках. Это позволяет уменьшить размер исполняемых приложений за счет исключения дублирования кода, а также оптимизировать использование памяти (Рисунок 1. Многоуровневая модель с отдельными библиотеками).

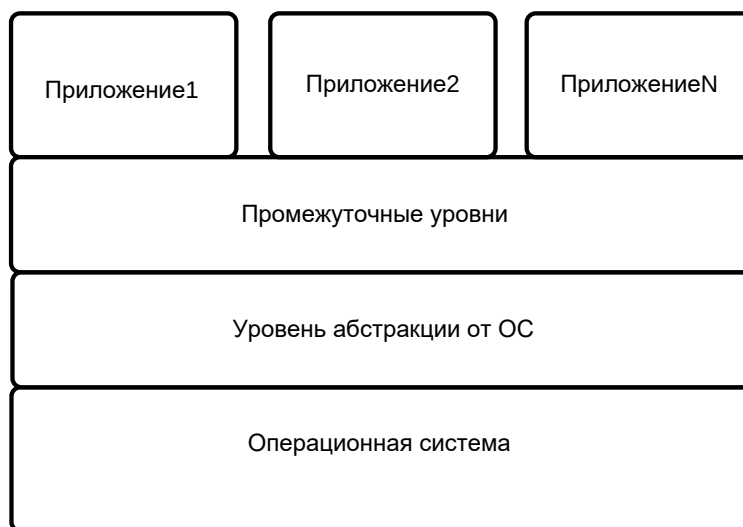


Рисунок 1. Многоуровневая модель с отдельными библиотеками

Если аппаратная платформа не поддерживает работу с разделяемыми библиотеками, то весь функционал приложения находится в одном исполняемом модуле (Рисунок 2. Многоуровневая модель без использования отдельных библиотек).

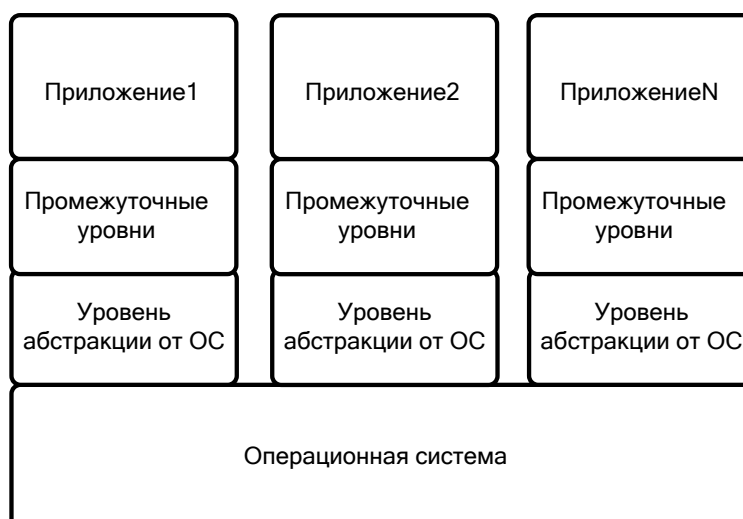


Рисунок 2. Многоуровневая модель без использования отдельных библиотек

1.1. Описание уровней архитектуры платежного приложения «Unipos Terminal»

Самый верхний уровень – это требуемая система (уровень «**Приложения**»), а самый нижний уровень – это доступные ресурсы: аппаратура, операционные системы, библиотеки (уровень «**Операционная система**»).

Обязательным уровнем является «**Уровень абстракции от операционной системы**», он обеспечивает переносимость системы на различные аппаратные платформы. Позволяет вышестоящим уровням абстрагироваться от конечного оборудования.

Интерфейс взаимодействия с устройствами через общий интерфейс включает в себя работу с:

- системой ввода/вывода;
- считывателями карт (магнитных, чиповых, бесконтактных);
- устройствами предоставления криптографического функционала;
- устройствами печати;
- коммуникационными устройствами;
- памятью;
- файловой системой.

Промежуточные уровни вводятся разработчиками системы. Каждый уровень базируется на всех следующих за ним уровнях. То есть уровень «**Приложения**» использует в своей работе уровни «**Промежуточные уровни**» и «**Уровень абстракции от операционной системы**». Использование уровня «**Операционная система**» допустимо только «**Уровнем абстракции от ОС**».

1. Уровень абстракции от железа («**Hardware Abstract Layer**»):
 - a. **HAL** - это модуль абстракции, между физическим оборудованием POS-терминала и программным обеспечением, которое работает на этом терминале. Он реализует общий интерфейс для связи с оборудованием на устройствах «**VeriFone**» и «**PAX**»;
 - b. **Communication** – модуль взаимодействия терминала с внешними устройствами по стандартным каналам связи. Модуль обеспечивает единый интерфейс работы с устройствами связи и транспортными протоколами для управляющего приложения;
 - c. **VSS** – модуль реализующий взаимодействие со специализированными скриптами безопасности «**VeriShield Security Scripts**» и предназначен для поддержки различных схем управления ключами, выполнения автономных проверок PIN и других функций по работе с хранениями ключей безопасности в аппаратном пинпаде;
 - d. **Crypto** - модуль, реализующий криптографические функции, такие как шифрование, дешифрование, расчет MAC, загрузка ключей в крипто-модуль (VSS/SAM), получение информации о загруженных ключах, преобразование форматов ключей (из «**ATM**» в «**GISKE**») и др.
2. Вспомогательные библиотеки («**Utils Layer**»):
 - a. **Utils** – вспомогательный модуль, содержащий имплементацию функций по работе со строками, файловыми хранилищами, алгоритмическим преобразованиями и др.;
 - b. **TACL** – модуль упаковки данных в конкретный формат хостовых сообщений;
 - c. **OpenSSL** – модуль, реализующий функции по работе с криптографической библиотекой с открытым исходным кодом «**OpenSSL**», реализующая поддержку SSL/TLS;
 - d. **IG** – графическая библиотека, предназначена для реализации графического интерфейса пользователя;

- e. **Memory Manager** – модуль распределения и управления доступной памятью (heap);
 - f. **DLA** – модуль, реализующий взаимодействие с системой удаленной загрузки мастер-ключей «RKL».
3. Сервисный уровень («Service Layer»):
- a. **Loyalty** – модуль, реализующий функциональность систем лояльности;
 - b. **ESLIP** – модуль взаимодействия с сервером хранения электронных чеков;
 - c. **HOSTS** – модуль взаимодействия с банковским хостом, реализующий протокол обмена данными;
 - d. **Monitoring** – модуль взаимодействия с сервером мониторинга, реализующий протокол обмена и регистрации событий на терминале;
 - e. **Plugin Manager** – интерфейс для взаимодействия терминального ПО с плагинами (динамические библиотеки реализующие вспомогательную функциональность).
4. Уровень приложения:
- a. **EMV** – платформо-независимый модуль, реализующий логику обслуживания EMV-карт;
 - b. **Contactless** – модуль обслуживания бесконтактных карт для взаимодействия с NFC-ридером по протоколу вендора;
 - c. **SmartSale** – модуль реализующий логику взаимодействия терминала и кассы в интегрированных решениях по протоколу «SA»;
 - d. **DLS** – модуль реализующий функциональность системы лояльности DLS;
 - e. **Host Client** – интерфейс между основным платежным приложением и хостовой библиотекой;
 - f. **TransactionExec** – реализация основной логики выполнения транзакций на терминале;
 - g. **Params** – модуль взаимодействия с системой параметризации.

2. Уровень аппаратной абстракции и дополнительные библиотеки

Программное обеспечение POS-терминала состоит из трех уровней:

1. Уровень программного обеспечения;
2. Уровень аппаратной абстракции и дополнительные библиотеки;
3. Уровень операционной системы.

Этот пункт описывает только второй уровень. Общее расположение и общее описание программного обеспечения POS-терминала показано на рисунке ниже (Рисунок 3. Общее расположение и общее описание программного обеспечения POS-терминала).

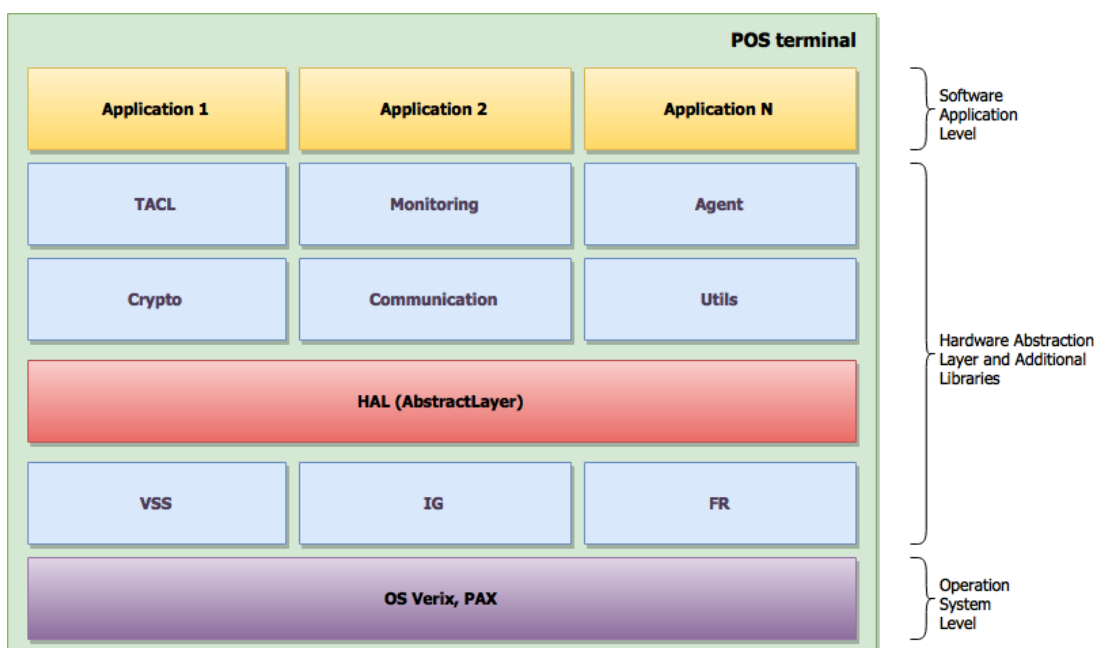


Рисунок 3. Общее расположение и общее описание программного обеспечения POS-терминала

2.1. Описание модулей

2.1.1. Abstract Layer Module

Abstract Layer - это уровень абстракции, реализованный программно, между физическим оборудованием POS-терминала и программным обеспечением, которое работает на этом терминале. Он реализует общий интерфейс для связи с оборудованием на устройствах «VeriFone» и «PAX».

Модуль «**Abstract Layer**» - это динамическая библиотека ссылок, которая предоставляет внешний интерфейс. Этот интерфейс используется основным программным обеспечением POS-терминала для управления батареей, считывателем чип-карт, COM-портом, настройками даты и времени, дисплеем, файловой системой, клавиатурой, считывателем магнитных карт, PIN-кодом и принтером. Модуль «**Abstract Layer**» был разработан с использованием языка программирования «C» и имеет несколько экспортируемых функций, которые описаны ниже.

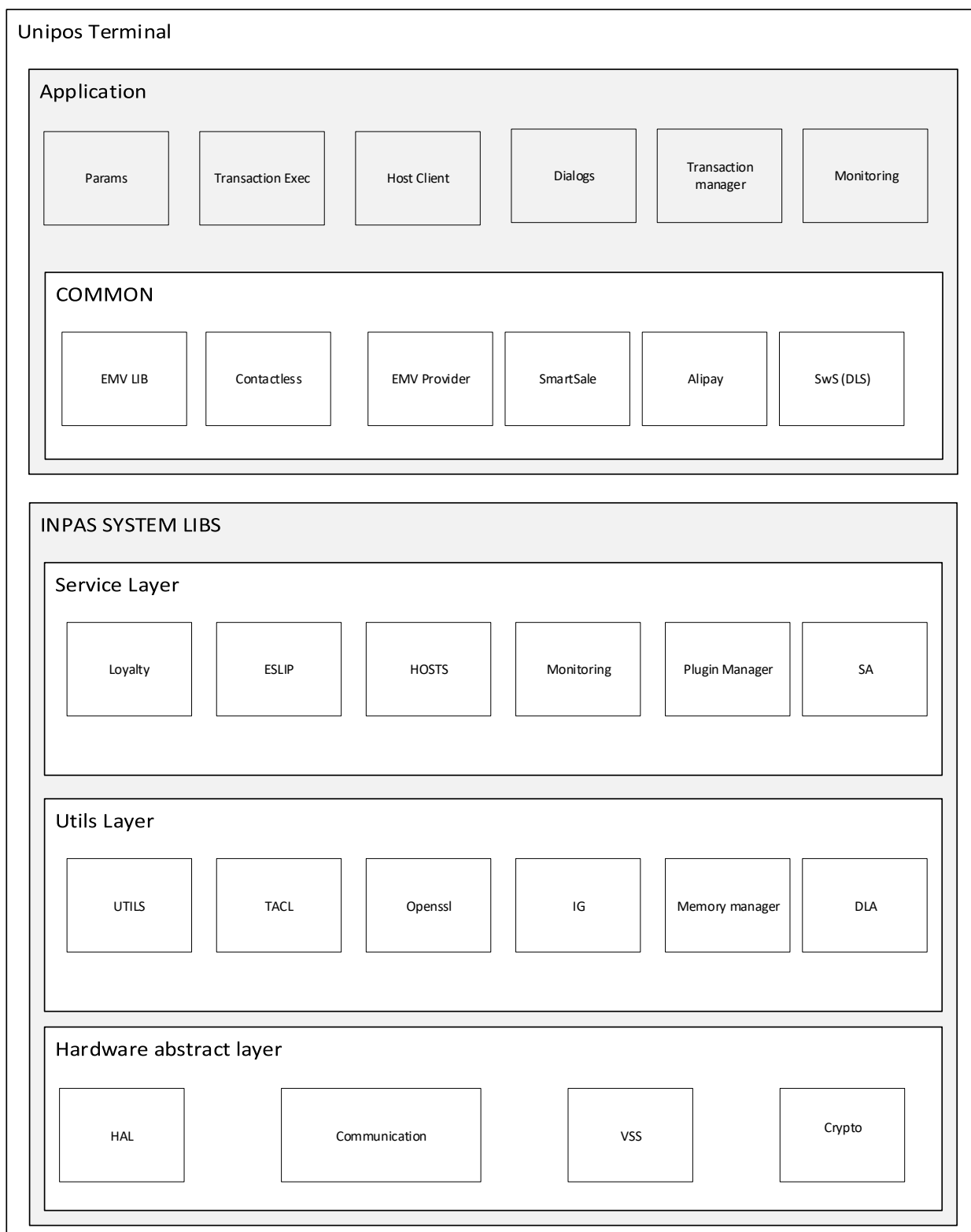
- **System Module API:**
 - **GetApplicationName** – эта функция получает имя запущенного приложения:
 - Название – «`long GetApplicationName(char * name, word nameSize);`»;
 - Возвращает:
 - length – длина имени приложения;
 - ERR_SYSTEMS_INVALID_BUFFER – значение нулевого указателя на блок памяти;
 - ERR_SYSTEMS_GETAPPNAME – ошибка получения имени приложения.

- Параметры:
 - name [out] – возвращаемое значение имени приложения;
 - nameSize [in] – размер буфера для имени приложения.
- **ApplicationRestart** – эта функция перезагружает запущенное приложение:
 - Название – «**void ApplicationRestart(void)**».
- **GetModelName** – эта функция получает название модели устройства:
 - Название – «**char * GetModelName(void)**»;
 - Возвращает:
 - Название модели устройства: «VeriFone», «PAX» и другие.
- **GetSerialNumber** – эта функция получает серийный номер устройства:
 - Название – «**long GetSerialNumber(char * serialNumber, word serialNumberSize)**»;
 - Возвращает:
 - length – длина серийного номера;
 - ERR_SYSTEMS_INVALID_BUFFER – значение нулевого указателя на блок памяти;
 - ERR_SYSTEMS_GETSERIALNUMBER – ошибка получения серийного номера.
 - Параметры:
 - serialNumber [out] – возвращаемое значение серийного номера;
 - serialNumberSize [in] – размер буфера для серийного номера.
- **Wait** – эта функция задерживает выполнение программы на указанное время. Вызов этой функции является вспомогательным для обеспечения модели события:
 - Название – «**void Wait(dword msec)**»;
 - Параметры:
 - msec [in] – время ожидания в миллисекундах.
- **WaitAnyEvent** – эта функция задерживает выполнение программы на указанное время. Вызов этой функции является вспомогательным для обеспечения модели события. Функция завершается, если произошло какое-либо обработанное событие:
 - Название – «**void WaitAnyEvent(dword msec)**»;
 - Параметры:
 - msec [in] – время ожидания в миллисекундах.
- **System_GetEnv** – эта функция читает системную переменную:
 - Название – «**long System_GetEnv(const char * key, char * buffer, long bufSize)**»;
 - Возвращает результат чтения:
 - >0 – длина переменной;
 - 0 – переменная отсутствует;
 - -1 – ошибка.
 - Параметры:
 - key [in] – имя переменной;
 - buffer [out] – читать переменную;
 - bufSize [in] – размер буфера для чтения переменной.
- **System_PutEnv** – эта функция записывает системную переменную:
 - Название – «**long System_PutEnv(const char * key, const char * val, long len)**»;

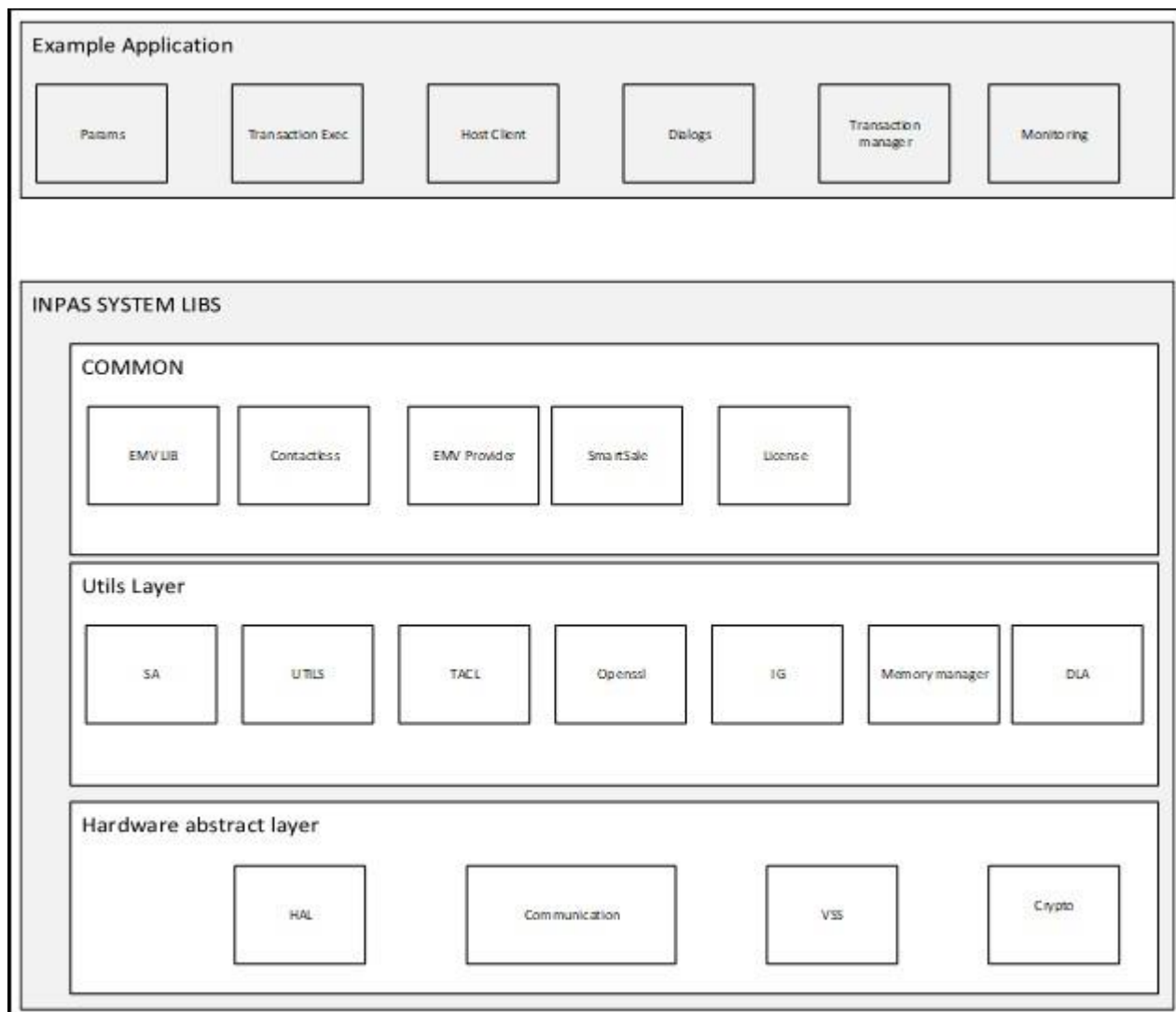
- Возвращает результат записи:
 - ≥ 0 – длина переменной;
 - -1 – ошибка.
- Параметры:
 - key [in] – имя переменной;
 - val [in] – значение переменной;
 - len [in] – длина переменной.
- **GetSizeWritableMemory** – эта функция возвращает размер свободной и используемой перезаписываемой памяти:
 - Название – «**long GetSizeWritableMemory(long * available, long * inUse)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение или ошибка в любом другом случае.
 - Параметры:
 - available [out] – доступная память в байтах (разделяемая память, доступно + используется);
 - inUse [out] – используемая память в байтах.
- **GetSizeWritableFlashMemory** – эта функция возвращает размер свободной и используемой перезаписываемой флэш-памяти:
 - Название – «**long GetSizeWritableFlashMemory(long * available, long * inUse)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение или ошибка в любом другом случае.
 - Параметры:
 - available [out] – доступная память в байтах (разделяемая память, доступно + используется);
 - inUse [out] – используемая память в байтах.
- **GetRandomBlock** – эта функция получает случайному последовательность из 8 байтов:
 - Название – «**long GetRandomBlock(unsigned char rnd[8])**»;
 - Возвращает:
 - SUCCESS – успешное выполнение или ошибка в любом другом случае;
 - ERR_SYSTEMS_INVALID_BUFFER – значение нулевого указателя на блок памяти.
 - Параметры:
 - rnd [out] – буфер для сохранения сформированной случайной последовательности.
- **IsApplicationExist** – эта функция проверяет, присутствует ли приложение с указанным именем в терминале:
 - Название – «**bool IsApplicationExist (const char * appname)**»;
 - Возвращает:
 - True – приложение присутствует.
 - Параметры:
 - appname [in] – имя приложения.

3. Приложения

3.1. Компонентная модель ПО «Unipos»



3.2. Компонентная модель системных библиотек «Unipos»



3.3. Список заголовочных файлов

AdditionalVariables.h	halS200ClssCommands.h
appManager.h	halS200Common.h
appManager2.h	halS200IO.h
coCommon.h	halS200MulappCommands.h
coCommonErrors.h	halS200PedCommands.h
coMacro.h	halS200ScrCommands.h
comCommunication.h	halSHA256.h
comCommunicationParameters.h	halStdlib.h
coTypes.h	halSystem.h
cryCrypto.h	halTerminal.h
dla.h	halTimer.h
DString.h	halTrace.h
halAbstractLayer.h	igBase.h
halBarcode.h	igUI.h
halBattery.h	kkt.h
halBeeper.h	MdbPort.h
halBmp.h	memMemory.h
halChipCardReader.h	PublicEMVKeys.h
halCipher.h	saSess.h
halComPort.h	srvService.h
halContactlessDevice.h	StorageEx.h
halConverter.h	tacl.h
halCrcCalc.h	taclConverter.h
halCryptoAlg.h	taclDataBase.h
halDateTime.h	taclExpressinParser.h
halDesCalc.h	taclEzXml.h
halDir.h	taclIsoMsg.h
halDisplay.h	taclLanguage.h
halDisplayBuf.h	taclLicensing.h
halEmuPrinter.h	taclPrintTemplate.h
halEmuProxy.h	taclQrEncode.h
halExtern.h	taclSet.h
halExternalVssPinPad.h	taclSpdhHeader.h
halFile.h	taclSpdhMsg.h
halFlashDrive.h	taclSpdhTrns.h
halHash.h	taclSpdhTrnsManager.h
halInt64.h	taclSpdhTypes.h
halKeyboard.h	taclXMLParserInterface.h
halLang.h	utiBerTlv.h
halMagCardReader.h	utiBorder.h
halMD5.h	utiCaption.h
halMemory.h	utiCodePage.h
halPinPad.h	utiCodePageManager.h
halPipe.h	utiDialogs.h
halPower.h	utiEdit.h
halPrinter.h	utiFooter.h
halRemoteDevices.h	utilImage.h
halS200BaseCommands.h	utilNi.h
halS200CL_common.h	utiLabel.h

utiList.h
 utiListBox.h
 utiMonitoring.h
 utiResource.h
 utiStorage.h
 utiText.h
 utiTLVParser.h
 utiTraceEx.h
 utiUtils.h
 VmacEvtID.h

vpl.h
 vss.h
 vssOptCode.h
 vssSAPIS.h
 wrt.h
 xlsconst.h
 zlib_iface.h
 EmvKernel.h
 EmvKernelDef.h