

INPAS-SOFT JOURNAL Module

Для разработчиков

Версия 1.0

История изменений документа

Версия	Дата	Перечень изменений
1.0	24.02.2021	Первоначальная версия

Содержание

1. Функциональное описание ПО	4
2. Описание модуля	5
2.1. Journal Module	5
2.1.1. DataBase API	5
2.1.2. IsoMsg API	7
2.1.3. Licending API	10
3. Пример использования функций «INPAS-SOFT JOURNAL Module»	12

1. Функциональное описание ПО

Программное обеспечение «**INPAS-SOFT JOURNAL Module**» состоит из нескольких модулей: «**DataBase API**», «**IsoMsg API**», «**Licencing API**», каждый из которых позволяет выполнять одно из ниже описанных действий:

- Создавать, удалять, искать, изменять, читать и писать записи в журнал транзакций, реализованный в виде файла.
- Формировать и анализировать пакет («**ISO8583**»), описывающий процесс передачи и формат финансовых сообщений (транзакций) системами, обрабатывающими данные банковских карт. Проверять обязательные поля, передаваемые в пакете «**ISO8583**».
- Работать с системой лицензирования, в частности проверять, какие функции разрешены лицензией, срок действия лицензии и тип используемой лицензии, в виде файла или на SAM-модуле.

ПО «**INPAS-SOFT Journal Module**» использует СУБД Unipos Database. Данная СУБД является собственной разработкой ООО «**ИНПАС СОФТ**», не является самостоятельным коммерческим продуктом.

ПО «**INPAS-SOFT JOURNAL Module**» поставляется в составе статической библиотеки «**C**» с расширением «***.a**».

Используемые функции «**INPAS-SOFT JOURNAL Module**», встраиваются в код исполняемого файла при компиляции.

2. Описание модуля

2.1. Journal Module

«Journal Module» был разработан с использованием языка программирования «С» и предназначен для работы с журналом транзакций, пакетами «ISO8583» и функционалом лицензирования. Данный модуль используется в ходе работы основного приложения для централизованной работы с данными проведения транзакций, имеет несколько экспортируемых библиотек свходящими в них функциями, которые описаны ниже.

2.1.1. DataBase API

Предназначен для работы с журналом истории транзакций. Это позволяет создавать, удалять, искать, читать и писать записи. Журнал транзакций реализован в виде файлов.

- **DataBase_Open** – эта функция открывает базу данных транзакции или создает ее, если она не существует:
 - Название – «**long DataBase_Open(const char * name)**»;
 - Возвращает:
 - handleDB – управление базой данных;
 - ERR_DATABASE_OPEN – ошибка открытия базы данных.
 - Параметры:
 - name [in] – название базы данных. Дополнительно к этому создаются временные файлы (с символами «n» и «b») и индекс файл (с символом «i»).
- **DataBase_SetParameters** – эта функция устанавливает параметры базы данных:
 - Название – «**long DataBase_SetParameters(long handleDB, const DataBaseParameters * parameters)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - parameters [in] – параметры базы данных.
- **DataBase_Check** – эта функция проверяет базу данных на целостность:
 - Название – «**long DataBase_Check(long handleDB)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта;
 - ERR_DATABASE_DATA – ошибка структуры базы данных.
 - Параметры:
 - handleDB [in] – управление базой данных.
- **DataBase_Close** – эта функция закрывает базу данных:
 - Название – «**void DataBase_Close(long handleDB)**»;
 - Параметры:
 - handleDB [in] – управление базой данных.
- **DataBase_SetSavedFields** – эта функция устанавливает список поддерживаемых параметров:
 - Название – «**void DataBase_SetSavedFields(long handleDB, t_key * savedFields, word fieldCount)**»;
 - Параметры:

- savedFields [in] - массив идентификаторов транзакций, сохраненных в базе данных. Эти параметры не сохраняются, если они пусты. Если параметры не определены, то все параметры контекста транзакции сохраняются;
- fieldCount [in] – количество элементов массива «saveFields».
- **DataBase_AddRecord** – эта функция добавляет записи в базу данных:
 - Название – «**long DataBase_AddRecord(long handleDB, const s_storage * transaction)**»;
 - Возвращает:
 - Результат >=0 – идентификатор добавленной записи, в противном случае – ошибка. Идентификатор действителен до удаления любой другой записи, на которую ссылается идентификатор;
 - ERR_DATABASE_ADDRECORD – ошибка записи данных;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - transaction [in] – параметры транзакции.
- **DataBase_UpdateRecord** – эта функция обновляет данные в текущей базе данных:
 - Название – «**long DataBase_UpdateRecord(long handleDB, long recordId, s_storage * transaction)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_UPDATERECORD – ошибка обновления записи;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта;
 - ERR_DATABASE_RECORD_NOT_FOUND – запись не найдена.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - recordId [in] – идентификатор записи;
 - transaction [in] – обновленные данные транзакции.
- **DataBase_ReadRecord** – эта функция читает запись:
 - Название – «**long DataBase_ReadRecord(long handleDB, long recordId, s_storage * record)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - recordId [in] – идентификатор записи;
 - record [out] – возвращенная запись.
- **DataBase_DeleteRecord** – эта функция удаляет запись:
 - Название – «**long DataBase_DeleteRecord(long handleDB, long recordId)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - recordId [in] – идентификатор записи.

- **DataBase_DeleteAllRecords** – эта функция удаляет все записи в базе данных:
 - Название – «**long DataBase_DeleteAllRecords(long handleDB)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных.
- **DataBase_FindRecord** – эта функция выполняет поиск в базе данных:
 - Название – «**long DataBase_FindRecord(long handleDB, s_storage * record, DataBaseFindInfo * findInfo)**»;
 - Возвращает:
 - recordId - идентификатор найденной записи. Идентификатор действителен до удаления любой другой записи, на которую ссылается идентификатор;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - recordId [in] – идентификатор записи;
 - findInfo [in] – критерии поиска и обработки найденных данных. Если этот параметр не определен (равен нулю), возвращается первая запись.
- **DataBase_FindNextRecord** – эта функция выполняет поиск в базе данных по параметрам, которые определены в методе «**DataBase_FindRecord**»:
 - Название – «**long DataBase_FindNextRecord(long handleDB, s_storage * record)**»;
 - Возвращает:
 - recordId - идентификатор найденной записи;
 - ERR_DATABASE_NOT_OPEN – база данных не открыта;
 - ERR_DATABASE_RECORD_NOT_FOUND – запись не найдена.
 - Параметры:
 - handleDB [in] – управление базой данных;
 - record [out] – данные найденной записи.
- **DataBase_GetCount** – эта функция получает номер всех записей в базе данных:
 - Название – «**long DataBase_GetCount(long handleDB)**»;
 - Возвращает количество записей;
 - Параметры:
 - handleDB [in] – управление базой данных.

2.1.2. IsoMsg API

Формирует пакет «**ISO8583**», анализирует пакет «**ISO8583**», проверяет обязательные поля.

- **Iso8583Msg_AssembleMsg** – эта функция формирует сообщение ISO8583 для отправки на хост:
 - Название – «**long Iso8583Msg_AssembleMsg(Iso8583FieldDescriptor * descriptors, s_storage *inData, char * outMessage, dword * inoutMessageSize, const char * mandatoryFields, dword mandatoryFieldsLen, const char * optionalFields, dword optionalFieldsLen)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;

- ERR_ – ошибки.
- Параметры:
 - thisMsg [in][out] - пакет ISO, созданный функцией «Iso8583Msg_Init»;
 - inData [in] – набор данных, поля для формирования запроса;
 - outMessage [in] – указатель на определенный буфер;
 - inoutMessageSize [in] – размер буфера;
 - mandatoryFields [in] – список обязательных полей. Например, «2,11,12,24,41»;
 - mandatoryFieldsLen [in] – длина строки обязательных полей;
 - optionalFields [in] – список необязательных полей. Например, «48»;
 - optionalFieldsLen [in] – длина строки необязательных полей.
- Iso8583Msg_AssembleMsgEx – эта функция формирует сообщение ISO8583 для отправки на хост:
 - Название – «long Iso8583Msg_AssembleMsgEx(ISO8583FieldDescriptor * descriptors, s_storage * storage, char * buffer, long bufferSize, const char * mandatoryFields, const char * optionalFields, bool singleMac, mac_method mac, byte * macIDs)»;
 - Возвращает количество упакованных данных в случае успешного выполнения, в противном случае – ошибки;
 - Параметры:
 - descriptors [in] – указатель на поле спецификатора формата;
 - storage [in] – набор данных для формирования запроса;
 - buffer [in] – указатель на определенный буфер;
 - bufferSize [in] – размер буфера;
 - mandatoryFields [in] – список обязательных полей. Например, «2,11,12,24,41»;
 - optionalFields [in] – список необязательных полей. Например, «48»;
 - singleMac [in]:
 - true – использовать MAC указанный в полях 64 или 128;
 - false – в случае наличия полей 65..127, включать MAC в поля 64 и 128.
 - mac [in] – указатель для вычисления функции MAC;
 - macIDs [in] – указатель на массив полей идентификаторов, используемых при вычислении MAC, байты «macIDs[] = { 2, 3, 4, 11, 255 }»:
 - для включения типа сообщения в расчет MAC использовать идентификатор = 0;
 - для включения битовой маски в расчет MAC использовать идентификатор = 1,
 - массив идентификаторов полей должен заканчиваться на = 255,
 - если «macIDs» не определены (macIDs = 0) тогда все поля используются в определении MAC.
- Iso8583Msg_ProcessMsg – эта функция анализирует сообщение ISO8583:
 - Название – «long Iso8583Msg_ProcessMsg(ISO8583FieldDescriptor * descriptors, s_storage * outData, char * const inMessage, dword

inMessageSize, const char * mandatoryFields, dword mandatoryFieldsLen, const char * optionalFields, dword optionalFieldsLen)»;

- Возвращает указатель на строку с типом сообщения;
- Параметры:
 - thisMsg [in][out] – сообщение ISO8583 созданное функцией «**Iso8583Msg_Init**»;
 - outData [out] – набор данных для полей анализируемого сообщения;
 - inMessage [in] – сообщение ISO8583 для анализа;
 - inMessageSize [in] – длина сообщения ISO8583;
 - mandatoryFields [in] – список обязательных полей. Например, «**2,11,12,24,41**»;
 - mandatoryFieldsLen [in] – длина строки обязательных полей;
 - optionalFields [in] – список необязательных полей. Например, «**48**»;
 - optionalFieldsLen [in] – длина строки необязательных полей.
- **Iso8583Msg_GetMti** – эта функция получает тип сообщения (Message Type Identifier):
 - Название – «**const char * Iso8583Msg_GetMti(ISO8583FieldDescriptor * descriptors, char * const inMessage, dword inMessageSize)**»;
 - Возвращает указатель на строку с MTI;
 - Параметры:
 - descriptors [in] – указатель на поле спецификатора формата;
 - inMessage [in] – сообщение ISO8583 для анализа;
 - inMessageSize [in] – длина сообщения ISO8583.
- **Iso8583Msg_CheckMandatoryFields** – эта функция проверяет обязательные поля:
 - Название – «**long Iso8583Msg_CheckMandatoryFields(s_storage * storage, const char * mandatoryFields)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_ – ошибки.
 - Параметры:
 - storage [in] – набор данных (полей);
 - mandatoryFields [in] – список обязательных полей. Например, «**2,11,12,24,41**».
- **Iso8583Msg_CheckEcho** – эта функция проверяет поля echo:
 - Название – «**long Iso8583Msg_CheckEcho(ISO8583FieldDescriptor * descriptors, s_storage * storageIn, s_storage * storageOut, const char * echoFields)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_ – ошибки.
 - Параметры:
 - descriptors [in] – указатель на поле спецификатора формата;
 - storageIn [in] – набор данных полей запроса;
 - storageOut [in] – набор данных полей ответа;
 - mandatoryFields [in] – список полей echo. Например, «**2,11,12,24,41**».

2.1.3. Licencing API

Используется для работы с системой лицензирования. Лицензия может быть двух типов: лицензия в виде файла и лицензия на SAM модуль. Используя функции этого модуля, мы можем проверить, какие функции разрешены лицензией, срок действия лицензии, тип используемой лицензии.

- **Licensing_Init** – эта функция выполняет инициализацию функции для поддержки лицензии:
 - Название – «**long Licensing_Init(const char * productPublicKey , dword productPublicKeyExponent, long productID)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_ – ошибки.
 - Параметры:
 - productPublicKey – открытый ключ продукта;
 - productPublicKeyExponent – экспонента открытого ключа продукта;
 - productID – идентификатор продукта.
- **Licensing_VerifyAvailableFeature** – эта функция допустимость использования функциональности:
 - Название – «**long Licensing_VerifyAvailableFeature(LicenseFeaturesTypes typeFeature, LicenseTerminalType terminalType)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_ – ошибки.
 - Параметры:
 - typeFeature – тип функциональности.
- **Licensing_Close** – эта функция завершает работу с функциональностью поддержки лицензии:
 - Название – «**long Licensing_Close(void)**»;
 - Возвращает:
 - SUCCESS – успешное выполнение;
 - ERR_ – ошибки.
- **Licensing_IsDemoMode** – эта функция возвращает признак работы терминала в демонстративном режиме:
 - Название – «**bool Licensing_IsDemoMode(void)**»;
 - Возвращает:
 - True – демонстративный режим.
- **Licensing_GetSamID** – эта функция получает серийный номер SAM модуля:
 - Название – «**long Licensing_GetSamID(byte * samId, long samIdSize)**»;
 - Возвращает:
 - [long] – результат выполнения функции.
 - Параметры:
 - samId [in] – буфер для серийного номера;
 - samIdSize [in] – размер буфера.
- **GetLicenseInfo** – эта функция получает информацию о лицензии: идентификатор продукта (Product ID), идентификатор SAM (CPLC), идентификатор клиента (ClientID), список функций продукта и даты их истечения:
 - Название – «**GetLicenseInfo(char** LicenseInfo)**»;

- Возвращает:
 - [long] – результат выполнения функции, размер заполненных данных.
- Параметры:
 - LicenseInfo [in] –буфер для информации.
- **GetLicenseInfoFeatures** – эта функция заполняет строку следующей информацией о лицензии: список функциональных возможностей продукта и даты их истечения:
 - Название – «**long GetLicenseInfoFeatures(char** LicenseInfoFeatures)**»;
 - Возвращает:
 - [long] – результат выполнения функции, размер заполненных данных.
 - Параметры:
 - LicenseInfo [in] –буфер для информации.
- **GetLicenseType** –эта функция получает информацию о типе лицензии:
 - Название – «**byte GetLicenseType(void)**»;
 - Возвращает:
 - [byte] – результат выполнения функции, тип используемой лицензии.

3. Пример использования функций «INPAS-SOFT JOURNAL Module»

В данном пункте приведен пример использования функций:

- **DataBase_Open** - открывает базу данных транзакции или создает ее, если она не существует;
- **DataBase_SetSavedFields** - устанавливает список поддерживаемых параметров;
- **DataBase_Check** - проверяет базу данных на целостность;
- **DataBase_GetCount** - получает номер всех записей в базе данных;
- **DataBase_ReadRecord** - читает запись;
- **DataBase_Close** - закрывает базу данных.

```
t_key ParamPosIdsDataToSave[] =
{
    /** Название виртуального терминала. */
    PosID_VirtualPOSID,
    /** Название учреждения */
    PosID_PossesorName,
    /** Наименование профиля терминала */
    PosID_ProfileName
};

constword ParamPosIdsDataToSaveCount = sizeof(ParamPosIdsDataToSave) / sizeof(t_key);

/**
 */
long PosIdDataExt_UnpackFromFile(ParamPosIdsDataExt * posIdsDataExt, long counts)
{
    long res = SUCCESS;
    long handleDB = -1;
    s_storage * posIdBase = NULL;
    long recordId = 0;

    if(posIdsDataExt == NULL)
        return APP_ERR_ARG;

    memset(posIdsDataExt, 0, sizeof(ParamPosIdsDataExt));
    posIdsDataExt->Count = counts;

    if(counts > 0)
    {
        posIdsDataExt->Items = (ParamPosIdDataExt*)Memory_Malloc(posIdsDataExt->Count *
sizeof(ParamPosIdDataExt));
        if(posIdsDataExt->Items == NULL)
            return APP_ERR_OUT_OF_MEMORY;
    }

    memset(posIdsDataExt->Items, 0, posIdsDataExt->Count * sizeof(ParamPosIdDataExt));

    res = DataBase_Convert(DEF_POSIDDATAEXT_FILE_NAME);
    if(res != SUCCESS)
        return res;

    handleDB = DataBase_Open(DEF_POSIDDATAEXT_FILE_NAME);
    if(handleDB < 0)
        return handleDB;
```

```

DataBase_SetSavedFields(handleDB, ParamPosIdsDataToSave, ParamPosIdsDataToSaveCount);

do
{
    res = DataBase_Check(handleDB);
    if (res != SUCCESS)
        break;

    res = APP_ERR_OUT_OF_MEMORY;
    posIdBase = Storage_Create();
    if (posIdBase == NULL)
        break;

    res = APP_ERR_NO_DATA;
    if (DataBase_GetCount(handleDB) == 0)
        break;

    while (DataBase_ReadRecord(handleDB, recordId, posIdBase) >= 0 && (recordId < counts))
    {
        ParamPosIdDataExt * item = &posIdsDataExt->Items[recordId];

        if (item)
        {
            char * tmpString = NULL;
            dword lenVal = 0;
            int i = 0;

            for (i = 0; i < ParamPosIdsDataToSaveCount; i++)
            {
                t_key key = ParamPosIdsDataToSave[i];
                bool result = Storage_GetString(posIdBase, key, &tmpString, &lenVal);

                if ((result == true) && (tmpString != NULL))
                {
                    char *p = 0;
                    size_t max_len = 0;

                    switch (key)
                    {
                        {
                            case PosID_VirtualPOSID:
                                p = item->Id.VirtualPOSID;
                                max_len = sizeof(item->Id.VirtualPOSID);
                                break;
                            case PosID_PossesorName:
                                p = item->Id.PossesorName;
                                max_len = sizeof(item->Id.PossesorName);
                                break;
                            case PosID_ProfileName:
                                p = item->Id.ProfileName;
                                max_len = sizeof(item->Id.ProfileName);
                                break;
                        }
                    }

                    if (p && (lenVal < max_len))
                    {
                        memcpy(p, tmpString, lenVal);
                    }
                }
            }
        }
    }
}

```

```
        }  
    }  
    res = SUCCESS;  
}  
recordId++;  
}  
  
}while(false);  
  
if(posIdBase != NULL)  
    Storage_Free(posIdBase);  
  
DataBase_Close(handleDB);  
  
return res;  
} /* PosIdDataExt_UnpackFromFile */
```