СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Прикладной интерфейс

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: info@linter.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие

Назначение документа

В документе приведено описание функций библиотеки прикладного интерфейса `LinAPI` и ее использование для разработки клиентских приложений.

Документ предназначен для СУБД ЛИНТЕР СТАНДАРТ 6.0 сборка 20.2, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для прикладных программистов, разрабатывающих приложения на языке программирования C/C++.

Дополнительные документы

- Архитектура СУБД
- Сетевые средства
- Интерфейс нижнего уровня

Основные сведения о LinAPI

Библиотека представляет собой набор функций для работы с СУБД ЛИНТЕР из программ, написанных на языке программирования C/C++.

Библиотека доступна в виде `lapi325.dll` или `lapi64.dll` (в зависимости от разрядности ОС) для Windows-систем или объектной библиотеки для UNIX-систем `linapi.so` и `linapimt.so`.

Для 32-х и 64-х разрядных ОС Windows `lapi325.lib` и `lapi64.lib` для Microsoft C; для 32-х разрядных ОС Windows `lapi325.lib` для Intel C и Watcom C. Для Linux и UNIX: `linapi.a`, `linapi.la`, `linapimt.a` и `linapimt.la`.

Структура LinAPI

В LinAPI имеются три типа объектов, позволяющих работать с базой данных (БД):

1) соединения (Connections);
2) курсоры (Cursors);
3) операторы (Statements).

Соединение (Connection)

Функция LINTER_Connect устанавливает соединение с ЛИНТЕР-сервером. При создании соединения открывается один канал связи с ЛИНТЕР-сервером. Создав соединение, приложение получает доступ к БД, с которой в данный момент работает ЛИНТЕР-сервер.

Одно приложение может установить несколько соединений как с одним, так и с разными ЛИНТЕР-серверами.

Курсор (Cursor)

Функция LINTER_OpenCursor открывает курсор для указанного соединения. Курсор – это средство выполнения запросов и/или операторов. При открытии курсора открывается канал связи с ЛИНТЕР-сервером.

Этот канал является дочерним (подчиненным) каналом по отношению к соединению, для которого открывается курсор.

Таким образом, под курсором понимается некоторая область данных, описывающая состояние выполнения запроса или оператора. С одним курсором могут быть связаны несколько операторов, но курсор будет содержать состояние того запроса или оператора, который был выполнен последним.

Оператор (Statement)

Функция LINTER_CreateStatement создает оператор, который является результатом трансляции SQL-запроса. Операция трансляции проводится по каналу, открытому при создании соединения.

Запрос, поданный на трансляцию, может содержать параметры. Параметры можно рассматривать как переменные, значения которых можно изменять. Чтобы подставить в оператор конкретное значение параметра, необходимо произвести привязку буфера параметра, в который помещается значение, к оператору.

Создавать оператор имеет смысл тогда, когда предполагается его многократное выполнение или использование параметров.

Один и тот же оператор может быть связан с несколькими курсорами, относящимися к соединению, по которому был создан оператор.

Связывание курсора и оператора происходит при привязке буфера параметра запроса и при привязке буфера поля ответа.

Эта связь пассивна и активизируется только при выполнении оператора.

Каждый из описанных объектов имеет набор свойств – характеристик, по которым можно судить о его состоянии или изменять это состояние.

Основные соглашения

В этом разделе даны основные определения и соглашения библиотеки LinAPI.

Соглашения о диагностике ошибок

Все функции LinAPI возвращают следующие коды завершения (см. таблицу 1):

Таблица 1. Коды завершения функций LinAPI

Код завершения	Числовое значение	Описание
LINAPI_SUCCESS	0	В случае удачного завершения
LINAPI_ERROR	-1	В случае внутренней ошибки
LINAPI_INV_ID	-2	В случае передачи функции неверного идентификатора
LINAPI_NO_MEM	-3	В случае недостатка оперативной памяти
LINAPI_BUSY	-4	В случае попытки повторного обращения к какому-либо объекту LinAPI, если предыдущая операция по этому объекту не была завершена
LINAPI_INV_CONTEXT	-5	В случае обнаружения ошибки во внутренних структурах
LINAPI_ID_NUM_EXCEEDED	-6	В случае превышения максимального количества используемых идентификаторов

Эти коды определены в заголовочном файле linapi.h.

Примечание
В случае удачного завершения возвращается 0 (LINAPI_SUCCESS). Если код завершения равен LINAPI_ERROR, то для получения диагностики работы библиотеки необходимо вызвать функцию LINTER_Error.

Синхронное и асинхронное выполнение запросов

В функции, допускающей асинхронную работу, присутствуют два параметра:

- AsyncFunc – адрес функции обработки ответа;
- UserArg – адрес единственного пользовательского аргумента этой функции.

Функция AsyncFunc должна иметь три аргумента:

```
void UserAsyncFunc(
   L_WORD CursorID,     /* идентификатор курсора */
   L_LONG LastError,    /* последняя LinAPI-ошибка */
   void *UserArg)       /* пользовательский аргумент */
```

Именно таким образом вызывается пользовательская функция в LinAPI.

Если оба адреса (`AsyncFunc` и `UserArg`) равны NULL, то запрос выполняется синхронно, т.е. выполнение следующего фрагмента программы не начнется раньше, чем выполнится запрос.

В противном случае программа выполняется далее, не ожидая конца обработки запроса (асинхронно). Как только запрос будет выполнен (с ошибкой или без нее), и приложение узнает об этом, то естественное выполнение программы прерывается, и вызывается указанная функция обработки ответа (с передачей ей `UserArg`). Выполнив функцию AsyncFunc, исполняющая система продолжит работу программы с той точки, где она была прервана.

В аргументе `LastError` возвращается последний код завершения LinAPI-интерфейса (коды завершения LinAPI приведены в приложении 1).

Примечания

1. О реентерабельности функций обработки ответа пользователь должен заботиться сам, т.к. нет гарантий, что при выполнении такой функции (асинхронный случай) не может быть вызвана другая (или та же) функция обработки ответа.
2. С помощью функций `LINTER_ConnectComplete` или `LINTER_CursorComplete` приложение может проверить завершение обработки запроса и, если запрос не обработан, перейти в режим ожидания завершения его обработки. Если адрес `AsyncFunc` (при не нулевом UserArg) равен NULL, то запрос все равно будет выполняться асинхронно, единственным способом определения прихода ответа является использование функции `LINTER_ConnectComplete`/`LINTER_CursorComplete`.
3. Асинхронность доступна в нескольких операционных системах: UNIX, Windows NT, VMS, OS-9, OS-9000.

Этапы прохождения запроса в LinAPI

Запрос в LinAPI может проходить несколько этапов:

1) трансляция запроса (для многократного выполнения это может быть достаточно выгодно). При этом запрос может содержать параметры – неизвестные константы, значения которых будут подставлены в запрос на следующем этапе;
2) привязка параметров, подстановка в запрос конкретных значений;
3) собственно выполнение запроса;
4) привязка полей ответа к программным переменным (в SELECT-запросах);
5) получение ответов с возможным перемещением по выборке (в SELECT-запросах).

Не все эти этапы являются необходимыми. Так, могут отсутствовать этапы трансляции и привязки, если в запросе отсутствуют параметры и его трансляция не планируется. Привязка полей ответа может предшествовать выполнению запроса.

Соединение (Connect)

Соединение – это средство подключения к СУБД. Только через соединение можно открыть курсор. Кроме того, соединение – средство объединения нескольких курсоров в одну транзакцию. Так что команда **COMMIT/ROLLBACK**, поданная по соединению,

относится ко всем курсорам этого соединения. Команда **CLOSE** закрывает все открытые курсоры указанного соединения.

Характеристики соединения приведены в приложении 2.

Характеристика с типом `cDBDesc` содержит информацию о БД:

```
#define linNameLen 66
typedef struct {
  L_LONG VerMajor,     /* версия ЛИНТЕР, для которой построена БД
 */
         VerMinor,
         VerBuild;
  L_LONG SortPoolSize; /* размер пула (в страницах) */
                       /* сортировки системы */
  L_LONG KernelPoolSize;   /* размер пула (в страницах) */
                           /* ядра системы */
  L_LONG FileQueueSize;    /* размер очереди файлов */
  L_LONG UserQueueSize;    /* размер очереди пользователей */
  L_LONG TableQueueSize;   /* размер очереди таблиц */
  L_LONG ColumnQueueSize;  /* размер очереди столбцов */
  L_LONG ChannelQueueSize; /* размер очереди каналов */
  L_LONG SnapTimeout;  /* период времени между операциями Full
 Snap */
  L_LONG KillTimeout;  /* таймаут опроса существования клиента */
  L_WORD NumOfSort;    /* кол-во процессов сортировки */
  L_BYTE Flags;        /* характеристика БД */
  L_BYTE BReserv1;
  L_LONG LReserv2;
  L_WORD SQLUsrCacheSize;  /* размер кэша пользователей БД */
                           /* в SQL-трансляторе */
  L_WORD SQLTabCacheSize   /* размер кэша таблиц в SQL-трансляторе
 */
  L_WORD SQLColCacheSize;  /* размер кэша столбцов */
                           /* в SQL-трансляторе */
  L_WORD SQLPrcCacheSize   /* размер кэша процедур */
                           /* в SQL-трансляторе */
  L_WORD SQLChsCacheSize;  /* размер кэша кодировок */
                           /* в SQL-трансляторе */
  L_WORD MaxRecSize;       /* максимальный размер записи таблицы
 */
  L_CHAR BaseName[18];     /* имя БД */
  L_CHAR SysLog;       /* признак работы с журн. транзакц. */
  L_CHAR Sync;         /* признак синхронизации ввода/вывода */
  L_CHAR Log;          /* признак ведения файла-протокола */
  L_CHAR Os;           /* идентификатор операционной системы
 сервера */
  L_WORD CharSet;      /* кодовая страница */
```

```
} t_DBDesc;
```

Курсор (Cursor)

Курсор – это средство выполнения запросов и/или операторов. По курсору может быть подано (последовательно) несколько запросов. Кроме того, курсор – средство для организации транзакций. Несколько запросов, поданных по курсору, могут быть завершены командой **COMMIT** (фиксация в БД изменений, произведенных этими запросами) или **ROLLBACK** (откат изменений).

Характеристики, которые имеет курсор, также можно найти в приложении 2.

Оператор (Statment)

Оператор – это результат трансляции запроса, его характеристики можно найти в приложении 2.

Характеристика с типом `sAnswerDesc` – это описание столбца ответа, которое имеет следующую структуру:

```
#define linNameLen 66
typedef struct {
  L_CHAR Owner[linNameLen];    /* имя пользователя */
  L_CHAR Table[linNameLen];    /* имя таблицы */
  L_CHAR Column[linNameLen];   /* имя столбца/параметра */
  L_WORD Length;               /* максимальная длина */
  L_BYTE Type;                 /* тип */
  L_BYTE Prec;                 /* точность */
  L_BYTE Scale;                /* масштаб */
  L_BYTE NullIndicator;        /* индикатор наличия NULL-значения
 */
  L_LONG RealLength;           /* фактическая длина */
} t_ParamDesc;
```

Запрос, поданный на трансляцию, может содержать (или не содержать) параметры, значения которых потом (на этапе `Bind`) вставляются на соответствующие места запроса.

Характеристика с типом `sParamDesc` – это описание параметра запроса (оператора). Данная характеристика принимается также в структуру `t_ParamDesc`, однако при этом не заполняются поля `Owner`, `Table` и `RealLength`. В поле Column возвращается имя параметра либо пустая строка, если имя не было задано.

Если значение поля `RealLength` равно `-1`, соответствующий параметр содержит null-значение.

Характеристика с типом `cProcArgDesc` или `ARGPROC_OUT` – это описание возвращаемого значения и выходных параметров после выполнения хранимой процедуры.

```
typedef struct {
#if _VER_MAX >= 500
```



```
    L_WORD Length,  /* длина значения */
    L_BYTE Ntype,   /* тип значения */
    Prec,   /* точность */
    Scale,  /* масштаб */
    Reserv; /* зарезервировано */
    #if _VER_MAX >= 600
       WORD charset;   /* кодовая страница */
    #endif
#endif
} P_TYPE;

typedef struct {
    L_BYTE Flags,   /* флаги значения */
    L_BYTE Reserv,  /* зарезервировано */
    L_WORD Value,   /* относительный адрес значения в буфере
 ответа */
    P_TYPE Type,    /* описание типа значения */
} t_ProcArgDesc;    /* флаги значения */

typedef struct {
    L_BYTE Flags,   /* флаги значения ARGPROC_OUT */
    L_BYTE Reserv,  /* зарезервировано */
#if _VER_MAX >= 550
    L_WORD Expr,    /* идентификатор выражения */
                    /* (для внутреннего применения) */
#endif
    L_WORD Value,   /* относительный адрес значения в буфере
 ответа */
    P_TYPE Type,    /* описание типа значения */
#if _VER_MAX >= 550
    WORD Reserv2;
#endif
} ARGPROC_OUT;  /* флаги значения */
```

Флаги значения:

```
#define fNULL 0x02
#define fCursor 0x04
#define fName 0x08
```

Объекты БД

LinAPI позволяет получать справки по разнообразным объектам БД (описания таблиц, столбцов, представлений и пр.).

Типы данных (LinAPI)

Типы данных, используемые в библиотеке представлены в таблице 2 :

Таблица 2. Типы данных LinAPI

Идентификатор типа	Числовое значение	Описание
`tChar`	1	Символьная строка фиксированной длины (может содержать двоичный нуль)
`tByte`	6	Буфер байтов фиксированной длины
`tString`	15	Символьная строка, заканчивающаяся двоичным нулем
`tSmallInt`	8	Короткое целое число
`tInteger`, `tInt`	2	Целое число
`tReal`	3	Действительное число
`tDouble`	8	Действительное число двойной точности
`tNumeric`, `tDecimal`, `tDec`	5	Действительное число, соответствует типу Decimal в СУБД ЛИНТЕР
`tDate`, `tTimeStamp`	4	Дата и время, тип, соответствующий типу DATE в СУБД ЛИНТЕР
`tBlob`	7	BLOB-значение
`tBigInt`	10	Длинное целое число
`tVarChar`	11	Символьная строка переменной длины (может содержать двоичный нуль)
`tVarByte`	12	Буфер байтов переменой длины
`tBoolean`	13	Логическое значение
`tNChar`	16	Символьная UNICODE-строка фиксированной длины
`tNVarChar`	17	Символьная UNICODE-строка переменной длины
`tExtFile`	18	Внешний файл

Стандартная структура приложения

На рисунке показана стандартная структура приложения, написанного с использованием LinAPI.

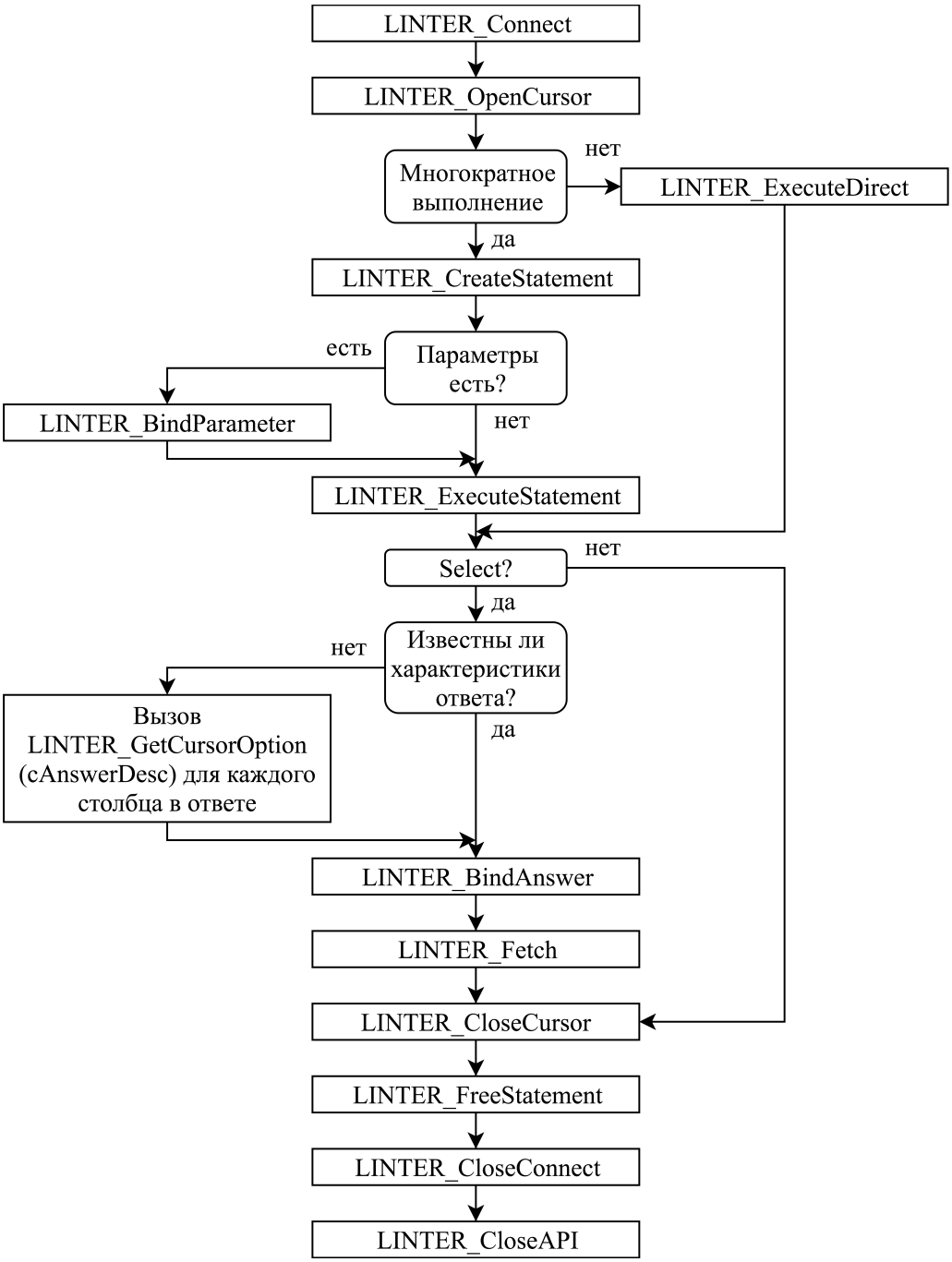

Рисунок. Стандартная структура LinAPI-приложения

Рассмотрим эту структуру.

Для выполнения какого-либо запроса необходимо создать соединение и открыть курсор. Если запрос будет выполняться несколько раз или имеет параметры, то необходимо создать оператор. В случае наличия параметров производится их привязка.

Затем выполняется оператор. Если запрос был SELECT-запросом, то для получения ответа привязываются буфера полей ответа, в которые будет выводиться ответ при перемещении по выборке.

Оператор выполняется необходимое количество раз.

Если запрос не имеет параметров либо должен выполняться только один раз, то создавать оператор не имеет смысла. Запрос выполняется непосредственно функцией `LINTER_ExecuteDirect`. Если это SELECT-запрос, то для получения ответа можно произвести действия, аналогичные действиям при работе с оператором.

Перед завершением работы приложения необходимо закрыть все курсоры, соединения, освободить операторы и функцией `LINTER_CloseAPI` освободить все задействованные ресурсы.

Следует отметить, что приведенная последовательность действий не является единственно правильной. Так, например, регистрация буферов полей ответа может предшествовать выполнению оператора, и буфера для получения ответа регистрировать не обязательно, ответ можно получить еще двумя способами, в зависимости от специфики приложения.

Описание функций

Работа с соединениями, курсорами, операторами

Создание соединения – LINTER_ConnectCSEx/ LINTER_ConnectCS/ LINTER_Connect

Прототипы функций

```
L_LONG LINTER_ConnectCSEx(
  L_CHAR *UserName,       /* имя пользователя */
  L_SWORD NameLen,        /* длина имени пользователя */
  L_CHAR *PassWord,       /* пароль пользователя */
  L_SWORD PassLen,        /* длина пароля пользователя */
  L_CHAR *ServerName,     /* имя ЛИНТЕР-сервера */
  L_CHAR *CharSet,        /* имя кодовой страницы */
  L_LONG  Mode,           /* режим работы соединения  */
  L_WORD *ConnectionID,   /* идентификатор соединения */
  L_LONG *plApiError,     /* код завершения LinAPI */
  L_LONG *plLinError,     /* код завершения СУБД ЛИНТЕР */
  L_LONG *plSysError);    /* код завершения ОС */
```

Устаревшие варианты:

```
L_LONG LINTER_ConnectCS(
  L_CHAR *UserName,       /* имя пользователя */
  L_SWORD NameLen,        /* длина имени пользователя */
  L_CHAR *PassWord,       /* пароль пользователя */
  L_SWORD PassLen,        /* длина пароля пользователя */
  L_CHAR *ServerName,     /* имя ЛИНТЕР-сервера */
  L_CHAR *CharSet,        /* имя кодовой страницы */
  L_LONG  Mode,           /* режим работы соединения  */
  L_WORD *ConnectionID);  /* идентификатор соединения */

  L_LONG LINTER_Connect(
  L_CHAR *UserName,       /* имя пользователя */
  L_SWORD NameLen,        /* длина имени пользователя */
  L_CHAR *PassWord,       /* пароль пользователя */
  L_SWORD PassLen,        /* длина пароля пользователя */
  L_CHAR *ServerName,     /* имя ЛИНТЕР-сервера */
  L_LONG  Mode,           /* режим работы соединения */
  L_WORD *ConnectionID);  /* идентификатор соединения */
```

Входные параметры

Параметр	Описание
UserName	Имя пользователя
NameLen	Длина имени пользователя

Параметр	Описание
Password	Пароль пользователя (может иметь значение NULL)
PassLen	Длина пароля пользователя
ServerName	Имя ЛИНТЕР-сервера (может иметь значение NULL для использования сервера по умолчанию)
CharSet	Имя кодовой страницы
Mode	Режим работы соединения

Режимы соединения (определены в заголовочном файле `linapi.h`):

- mAutocommit – режим транзакций AUTOCOMMIT;
- mOptimistic – режим транзакций OPTIMISTIC;

Примечание
Режим OPTIMISTIC устарел (использовать не рекомендуется).

- mExclusive – режим транзакций PESSIMISTIC;
- mAnsi – канал работает в кодировке ANSI;
- mKoi8 – канал работает в кодировке KOI8.

Выходные параметры

Параметр	Описание
ConnectID	Идентификатор созданного соединения
Mode	Установленный режим работы соединения
plApiError	Код завершения LinAPI-интерфейса
plLinError	Код завершения СУБД ЛИНТЕР
plSysError	Код завершения ОС

Описание

Функции устанавливают соединение с ЛИНТЕР-сервером с именем `ServerName` для пользователя `UserName` с паролем `PassWord`. Одновременно может быть открыто несколько соединений.

В поле `UserName` возможно указание имени пользователя и, одновременно, пароля через косую черту (**USERNAME/PASSWORD**). Соединение будет иметь режим обработки транзакций равный **MODE**.

Если имена пользователей и их пароли не обрамлены двойными кавычками, то их значения приводятся к верхнему регистру, иначе передаются ядру СУБД ЛИНТЕР как есть, например,

пользователь: "\"Вася\""

пароль: \"Пупкин\"

Режимы обработки транзакций можно задавать одновременно с указанием режима кодировки записей выборки данных.

Функция LINTER_ConnectCSEx возвращают коды завершения в выходных параметрах, а для получения кодов завершения функций LINTER_Connect и LINTER_ConnectCS необходимо после их выполнения вызывать дополнительно функцию LINTER_GetError.

Пример

```
long lError;
short nConnID;
…
If (lError =LINTER_Connect("system", 0, "manager", 0, NULL,
 mOptimistic, &nConnID))
 processing_error(lError, nConnID, 0, 0, "LINTER_Connect");
```

Создание соединения с помощью конфигурации соединения – LINTER_ConnectEnvEx/ LINTER_ConnectEnv

Прототипы функций

```
L_LONG LINTER_ConnectEnvEx(
 L_CHAR *sCfgName,       /* имя конфигурации соединения */
 L_SWORD swCfgNameLen,   /* длина имени конфигурации соединения */
 L_LONG *plMode,         /* режим работы создаваемого соединения */
 L_WORD pwConnectionId, /* идентификатор созданного соединения */
 L_LONG *plApiError,     /* код завершения LinAPI */
 L_LONG *plLinError,     /* код завершения СУБД ЛИНТЕР */
 L_LONG *plSysError);    /* код завершения ОС */
```

Устаревший вариант:

```
L_LONG LINTER_ConnectEnv(
 L_CHAR *sCfgName,       /* имя конфигурации соединения */
 L_SWORD swCfgNameLen,   /* длина имени конфигурации соединения */
 L_LONG *plMode,         /* режим работы создаваемого соединения */
 L_WORD pwConnectionId);/* идентификатор созданного соединения */
```

Входные параметры

Параметр	Описание
`sCfgName`	Имя конфигурации соединения. Если аргумент не задан, по умолчанию используется имя LINTER
`swCfgNameLen`	Длина имени конфигурации соединения (максимально допустимая длина 58 символов). Если строка sCfgName ограничена символом '\0', то можно указать нулевое значение длины
`plMode`	Режим работы создаваемого соединения
`pwConnectionId`	Идентификатор соединения

Режимы работы создаваемого соединения (определены в заголовочном файле `linapi.h`):

- mAutocommit – режим транзакций AUTOCOMMIT;
- mOptimistic – режим транзакций OPTIMISTIC;

Примечание
Режим OPTIMISTIC устарел (использовать не рекомендуется).

Выходные параметры

Параметр	Описание
`pwConnectionId`	Идентификатор созданного соединения
`plMode`	Установленный режим соединения
`plApiError`	Код завершения LinAPI-интерфейса
`plLinError`	Код завершения СУБД ЛИНТЕР
`plSysError`	Код завершения ОС

Описание

Функция предназначена для создания соединения с ЛИНТЕР-сервером на основе определяемой клиентским приложением конфигурации соединения.

Клиентское приложение может содержать любое количество конфигураций соединения и использовать их в соответствии со своей логикой работы (например, в случае отказа в доступе к локальному ЛИНТЕР-серверу попытаться установить соединение с удаленным ЛИНТЕР-сервером).

Для идентификации используемой при создании соединения конфигурации соединения используется имя конфигурации, которое задается в аргументе `sCfgName` данной функции.

Для каждой конфигурации соединения в ОС должны быть созданы две переменных окружения, определяющие следующие значения:

- вид создаваемого соединения (локальное или сетевое);
- спецификацию файла с параметрами создаваемого соединения

Формат переменной окружения, задающей вид создаваемого соединения:

CONNECT<подчёркивание><имя конфигурации соединения>=<значение переменной CONNECT>

<имя конфигурации соединения>::= <значение параметра sCfgName>

<значение переменной CONNECT>::= <локальное соединение> | <сетевое соединение>

<локальное соединение>::=**1**:<значение LINTER_MBX>

<сетевое соединение>::=**2**:<имя ЛИНТЕР-сервера>

Примечание
Если переменная окружения CONNECT_" + sCfgName задает локальное соединение, то функция должна использоваться только в том случае, если у клиентского приложения нет других локальных соединений.

<значение LINTER_MBX>:= идентификатор межзадачного обмена (см. документ «Сетевые средства», раздел «Управление сетевым доступом»)

<имя ЛИНТЕР-сервера>::= имя узла локальной сети из файла сетевой конфигурации nodetab (см. документ «Сетевые средства», раздел «Файл сетевой конфигурации»)

Примеры

```
CONNECT_DEMO=1:30000
CONNECT_NETDEMO=2:NODE1
```

Формат переменной окружения, задающей параметры создаваемого соединения:

CFG<подчёркивание><имя конфигурации соединения>=<значение переменной CFG>

<значение переменной CFG>::=<спецификация файла>

<спецификация файла>::= путь и имя текстового файла с параметрами соединения

Формат текстового файла с параметрами соединения. Текстовый файл состоит из строк следующего формата:

<строка>::= <имя параметра>=<значение параметра>

<имя параметра>::= **USER** | **PASSWORD** | **CODEPAGE** | **CONCURRENCY**

Имена параметров регистронезависимы.

<значение параметра>:

1) USER – имя пользователя БД (строка длиной до 66 символов)
2) PASSWORD – пароль пользователя БД (строка длиной до 18 символов)
3) CODEPAGE – имя кодировки соединения (символьная строка)
4) CONCURRENCY – режим транзакций соединения (одно или несколько значений (с учётом регистра!)):
   - mAutocommit;
   - mOptimistic;

Примечание
Режим mOptimistic устарел (использовать не рекомендуется).

   - mExclusive.

Если значения в файле не заданы, то для параметров USER, PASSWORD, CODEPAGE будет устанавливаться пустая строка, а для параметра CONCURRENCY – значение `0` (соответствует mAutocommit).

Если один и тот же параметр задан несколько раз, используется последнее значение параметра.

Разделитель между значениями может быть произвольным строковым символом – пробелом, запятой и т.д.

Примеры

```
Переменные среды окружения (предполагается, что БД запущена
 локально с LINTER_MBX=20561):
CONNECT_LINTER=1:20561
```

```
CFG_LINTER=config.lin

Файл config.lin:
USER=SYSTEM
PASSWORD=MANAGER8
CODEPAGE=CP1251
CONCURRENCY=mExclusive
```

После обработки конфигурации соединения происходит вызов функции LINTER_ConnectCSEx с извлеченными параметрами из конфигурации соединения.

Функция LINTER_ConnectEnvEx возвращает коды завершения в выходных параметрах, а для получения кодов завершения функции LINTER_ConnectEnv необходимо после её выполнения вызывать дополнительно функцию LINTER_GetError.

Пример

1) Создание и настройка конфигурации локального соединения с именем LOCAL_DB (с LINTER_MBX=3000):

- создание файла параметров соединения param_local.lin:

```
USER=USER1
PASSWORD=12345
CODEPAGE=CP1251
CONCURRENCY=mAutocommit
```

- создание и установка переменных окружения:

```
SET CONNECT_LOCAL_DB = 1:3000
SET CFG_LOCAL_DB=param_local.lin (файл будет взят из каталога
 запуска клиентского приложения)
```

2) Создание и настройка конфигурации удаленного соединения с именем REMOTE_DB (имя сервера NODE1):

- создание файла параметров соединения param_remote.lin:

```
USER=USER1
PASSWORD=12345
CODEPAGE=CP1251
CONCURRENCY=mAutocommit
```

- создание и установка переменных окружения:

```
SET CONNECT_REMOTE_DB  =2:NODE1
SET CFG_REMOTE_DB=c:\linter\param_remote.lin
```

3) Вызов функции

```
long lError, lTransactionMode;
short nConnID;
...
If (lError =LINTER_ConnectEnv("LOCAL_DB", 0, &lTransactionMode,
 &nConnID))
```



```
 Processing_error(lError, nConnID, 0, 0, "LINTER_local_Connect");
...
If (lError =LINTER_ConnectEnv("REMOTE_DB", 0, &lTransactionMode,
 &nConnID))
 Processing_error(lError, nConnID, 0, 0, "LINTER_remote_Connect");
```

Создание асинхронного соединения – LINTER_AsyncConnectCSEx/ LINTER_AsyncConnectCS

Прототипы функций

```
L_LONG LINTER_AsyncConnectCSEx(
  L_CHAR *UserName,             /* имя пользователя */
  L_SWORD NameLen,              /* длина имени пользователя */
  L_CHAR *PassWord,             /* пароль пользователя */
  L_SWORD PassLen,              /* длина пароля пользователя */
  L_CHAR *ServerName,           /* имя ЛИНТЕР-сервера */
  L_CHAR *CharSet,              /* имя кодовой страницы */
  L_LONG  Mode,                 /* режим обработки транзакций */
  L_WORD *ConnectionID,         /* идентификатор соединения */
  L_LONG *plApiError,           /* код завершения LinAPI */
  L_LONG *plLinError,           /* код завершения СУБД ЛИНТЕР */
  L_LONG *plSysError,           /* код завершения ОС */
  void *AstFunction,            /* функция обработки ответа */
  void *UserArg);               /* пользовательский аргумент */
```

Устаревшие варианты:

```
L_LONG LINTER_AsyncConnectCS(
  L_CHAR *UserName,             /* имя пользователя */
  L_SWORD NameLen,              /* длина имени пользователя */
  L_CHAR *PassWord,             /* пароль пользователя */
  L_SWORD PassLen,              /* длина пароля пользователя */
  L_CHAR *ServerName,           /* имя ЛИНТЕР-сервера */
  L_CHAR *CharSet,              /* имя кодовой страницы */
  L_LONG  Mode,                 /* режим обработки транзакций */
  L_WORD *ConnectionID,         /* идентификатор соединения */
  void *AstFunction,            /* функция обработки ответа */
  void *UserArg);               /* пользовательский аргумент */


  L_LONG LINTER_AsyncConnect(
  L_CHAR *UserName,             /* имя пользователя */
  L_SWORD NameLen,              /* длина имени пользователя */
  L_CHAR *PassWord,             /* пароль пользователя */
  L_SWORD PassLen,              /* длина пароля пользователя */
  L_CHAR *ServerName,           /* имя ЛИНТЕР-сервера */
  L_LONG  Mode,                 /* режим обработки транзакций */
```

```
  L_WORD *ConnectionID,         /* идентификатор соединения */
  void *AstFunction,            /* функция обработки ответа */
  void *UserArg);               /* пользовательский аргумент */
```

Входные параметры

Аналогичны входным параметрам функций LINTER_Connect, LINTER_ConnectCS.

Выходные параметры

Аналогичны выходным параметрам функций LINTER_Connect, LINTER_ConnectCS.

Описание

Функции устанавливают соединение с СУБД в асинхронном режиме. В остальном они аналогичны функциям LINTER_Connect, LINTER_ConnectCS.

Функция LINTER_AsynConnectEx возвращает коды завершения в выходных параметрах, а для получения кодов завершения функции LINTER_ConnectEnv необходимо после её выполнения вызывать дополнительно функцию LINTER_GetError.

Создание асинхронного соединения с помощью конфигурации соединения – LINTER_AsyncConnectEnvEx/ LINTER_AsyncConnectEnv

Прототипы функций

```
L_LONG LINTER_AsyncConnectEnvEx(
 L_CHAR *sCfgName,       /* имя конфигурации соединения */
 L_SWORD swCfgNameLen,   /* длина имени конфигурации соединения */
 L_LONG *plMode,         /* режим работы создаваемого соединения */
 L_WORD *pwConnectionId,/* идентификатор созданного соединения */
 L_LONG *plApiError,     /* код завершения LinAPI */
 L_LONG *plLinError,     /* код завершения СУБД ЛИНТЕР */
 L_LONG *plSysError,     /* код завершения ОС */
 void *AsyncFunc,        /* функция обработки ответа */
 void *UserArg);         /* пользовательский аргумент */
```

Устаревший вариант:

```
L_LONG LINTER_AsyncConnectEnv(
 L_CHAR *sCfgName,       /* имя конфигурации соединения */
 L_SWORD swCfgNameLen,   /* длина имени конфигурации соединения */
 L_LONG *plMode,         /* режим работы создаваемого соединения */
 L_WORD *pwConnectionId,/* идентификатор созданного соединения */
 void *AsyncFunc,        /* функция обработки ответа */
 void *UserArg);         /* пользовательский аргумент */
```

Входные параметры

Аналогичны входным параметрам функций LINTER_ConnectEnvEx и LINTER_ConnectEnv.

Выходные параметры

Аналогичны выходным параметрам функций LINTER_ConnectEnvEx и LINTER_ConnectEnv.

Описание

Функции устанавливают соединение с СУБД ЛИНТЕР в асинхронном режиме. В остальном они аналогичны функциям LINTER_ConnectEnvEx и LINTER_ConnectEnv.

Возвращаемое значение переменной plMode выставляется сразу при выходе из функции LINTER_AsyncConnectEnv, а не при вызове функции обработки ответа, задаваемой в параметре AsyncFunc.

Функция LINTER_AsyncConnectEnvEx возвращает коды завершения в выходных параметрах, а для получения кодов завершения функции LINTER_AsyncConnectEnv необходимо после её выполнения вызывать дополнительно функцию LINTER_GetError.

Получение информации о БД – LINTER_ServerInfo

Прототип функции

```
L_LONG LINTER_ServerInfo(
  L_CHAR *ServerName,         /* имя сервера БД */
  L_SWORD ServerNameLength,   /* длина имени сервера БД */
  L_SWORD InfoType,           /* тип запрашиваемой информации */
  void * Buffer,              /* выходной буфер */
  L_WORD BufferLength,        /* размер выходного буфера */
  L_WORD * OutLengthPtr,      /* реальное количество байтов */
                              /* выходного буфера */
  L_LONG * ApiCode,           /* код завершения LinAPI */
  L_LONG * LinCode,           /* код завершения СУБД ЛИНТЕР */
  L_LONG * SysCode);          /* код завершения ОС */
```

Входные параметры

Параметр	Описание
`ServerName`	Имя сервера ЛИНТЕР. Если ServerName равен NULL, то запрос будет передан к локальному серверу
`ServerNameLength`	Длина имени сервера. Если длина меньше или равна нулю, то ServerName должен заканчиваться двоичным нулем
`InfoType`	Тип запрашиваемой информации
`BufferLength`	Размер выходного буфера Buffer

Выходные параметры

Параметр	Описание
`Buffer`	Выходной буфер
`OutLengthPtr`	Реальное количество байт, записанных в буфер `Buffer`, или требуемый размер буфера (если произошла ошибка из-за недостаточно размера буфера)

Параметр	Описание
ApiCode	Код завершения LinAPI, если функция вернула LINAPI_ERROR
LinCode	Код завершения СУБД, если функция вернула LINAPI_ERROR
SysCode	Код завершения ОС, если функция вернула LINAPI_ERROR

Описание

Помещает в буфер Buffer информацию о параметрах запрашиваемой БД в виде структуры t_DBDesc. Описание t_DBDesc приведено в пункте Соединение (Connect).

Примечание
В текущей реализации допустимо получение только одного типа запрашиваемой информации по идентификатору cDBDesc, который необходимо использовать в качестве значения для параметра InfoType.

Закрытие соединения – LINTER_CloseConnect

Прототип функции

```
L_LONG LINTER_CloseConnect(
  L_WORD ConnectID);          /* идентификатор соединения */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения

Описание

Закрывает соединение с идентификатором ConnectID и все курсоры/операторы этого соединения.

Пример

```
long lError;
short nConnID;
if (lError =LINTER_CloseConnect(nConnID))
  processing_error(lError, nConnID, 0, 0, "LINTER_CloseConnect");
```

Освобождение соединения – LINTER_FreeConnect

Прототип функции

```
L_LONG LINTER_FreeConnect(
  L_WORD ConnectID,          /* идентификатор соединения */
  L_LONG *plLinterError);    /* код завершения СУБД ЛИНТЕР */
```

Входные параметры

Параметр	Описание
ConnectionID	Идентификатор соединения

Описание

В случае разрыва соединения с сервером (независимо от причины) освобождает память, занимаемую структурами соединения.

Выходные параметры

Параметр	Описание
plLinterError	Код завершения СУБД ЛИНТЕР

Открытие курсора – LINTER_OpenCursor

Прототип функции

```
L_LONG LINTER_OpenCursor(
  L_WORD ConnectID,       /* идентификатор соединения */
  L_WORD *CursorID,       /* идентификатор курсора */
  L_CHAR *CursorName,     /* имя курсора */
  L_SWORD NameLen,        /* длина имени курсора */
  L_LONG Mode);           /* режим обработки транзакций */
```

Входные параметры

Параметр	Описание
ConnectID	Номер соединения, по которому открывается курсор
CursorName	Необязательное имя курсора, удовлетворяющее синтаксису имен в ЛИНТЕР-SQL, применяется для выборки
NameLen	Длина имени курсора; если NameLen=0, то считается, что CursorName должно заканчиваться двоичным нулем
Mode	Режим обработки для курсора (может отличаться от режима обработки транзакций соединения). Описание режимов см. в функции LINTER_ConnectCSEx

Выходные параметры

Параметр	Описание
CursorID	Номер открытого курсора

Описание

Открывает курсор по соединению ConnectID с указанным режимом обработки транзакций, присваивает ему имя CursorName и возвращает идентификатор курсора. Для каждого соединения может быть открыто несколько курсоров.

Примечания

1. Если адрес CursorName равен NULL, то курсор считается неименованным, и для этого курсора нельзя выполнить SQL-запрос UPDATE … CURRENT OF Cursor_Name.
2. При открытии курсора все его характеристики (кроме приоритета) получают нулевое значение. Значение приоритета курсор наследует от соединения, по которому он открывается.

Пример

```
long lError;
short nConnID;
short nCursID;
...
if (lError =LINTER_OpenCursor(nConnID, &nCursID, NULL, 0,
 mOptimistic))
 processing_error(lError, nConnID, 0, 0, "LINTER_OpenCursor");
```

Закрытие курсора – LINTER_CloseCursor

Прототип функции

```
L_LONG LINTER_CloseCursor(
  (L_WORD CursorID);      /* идентификатор курсора */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Описание

Закрывает курсор с идентификатором CursorID.

Если с курсором были созданы связи, т.е. производилась привязка параметров или полей ответа, то при закрытии курсора эти связи уничтожаются.

В случае возникновения любой ошибки, кроме ChannelBusy и ERRSEQCOM, функция успешно завершается с выдачей кода завершения LINAPI_ERROR_ON_CLOSE (см. файл linapi.h). В данном случае курсор удаляется, и получение дополнительной информации об ошибке становится невозможным.

Пример

```
long lError;
short nCursID;
…
if(lError = LINTER_CloseCursor(nCursID))
  processing_error(lError, 0, nCursID, 0, "LINTER_CloseCursor");
```

Закрытие канала – LINTER_KillerChannel, LINTER_KillChannel

Прототип функции (неканальный вариант)

```
L_LONG LINTER_KillerChannel(
  L_CHAR  * UserName,    /* имя пользователя           */
  L_SWORD   NameLen,     /* длина имени пользователя   */
  L_CHAR  * PassWord,    /* пароль пользователя        */
  L_SWORD   PassLen,     /* длина пароля пользователя  */
```



```
  L_WORD    nChannelID,  /* идентификатор канала       */
  L_LONG  * plApiError,  /* код завершения LinAPI      */
  L_LONG  * plLinError,  /* код завершения СУБД ЛИНТЕР */
  L_LONG  * plSysError); /* код завершения ОС          */
```

Входные параметры

Параметр	Описание
UserName	Имя пользователя
NameLen	Длина имени пользователя
PassWord	Пароль пользователя
PassLen	Длина пароля пользователя
nChannelID	Идентификатор канала, который следует закрыть

Выходные параметры

Параметр	Описание
plApiError	Код ошибки интерфейса LinAPI
plLinError	Код ошибки интерфейса CALL
plSysError	Код системной ошибки

Описание

Принудительное закрытие канала nChannelID без использования существующего соединения.

Пример

```
L_WORD channelId;
  ...
  L_LONG ApiError = 0;
  L_LONG LinError = 0;
  L_LONG SysError = 0;
  int lRet;

  lRet = LINTER_KillerChannel((L_CHAR *)"SYSTEM", (L_SWORD)0,
 (L_CHAR *)"MANAGER8", (L_SWORD)0, (L_WORD)channelId, &ApiError,
 &LinError, &SysError);
```

Прототип функции (канальный вариант)

```
L_LONG LINTER_KillChannel(
  L_WORD ConnectionId,  /* идентификатор соединения */
  L_WORD nChannelID);   /* идентификатор канала     */
```

Входные параметры

Параметр	Описание
ConnectionId	Идентификатор соединения
nChannelID	Идентификатор канала, который следует закрыть

Описание

Принудительное закрытие канала `nChannelID` с использованием уже существующего соединения.

Пример

```
L_WORD connectId;
L_WORD channelId;
int lRet;
...
lRet = LINTER_KillChannel(connectId, channelId);

if (lRet != 0)
{
  L_LONG ApiError = 0;
  L_LONG LinError = 0;
  L_LONG SysError = 0;

  L_CHAR mess[4096];
  L_SWORD messLen = (L_SWORD)sizeof(mess);

  lRet = LINTER_Error(connectId, 0, 0, &ApiError, &LinError,
 &SysError, mess, &messLen);
  ...
}
```

Создание оператора – LINTER_CreateStatement

Прототип функции

```
L_LONG LINTER_CreateStatement(
  L_WORD ConnectID,           /* идентификатор соединения */
  L_CHAR *Query,              /* текст SQL-запроса */
  L_LONG Length,              /* длина запроса */
  L_WORD *StatementID);       /* идентификатор оператора */
```

Входные параметры

Параметр	Описание
`ConnectID`	Идентификатор соединения
`Query`	Текст запроса
`Length`	Длина запроса

Выходные параметры

Параметр	Описание
`StatementID`	Номер оператора

Описание

Транслирует запрос `Query` и создает оператор с идентификатором `StatementID`.

> **Примечание**
> После создания оператора можно получить его характеристики с помощью функции `LINTER_GetStatementOption`.

Пример

```
long lError;
short nConnID;
short nStmtID;
char cQuery[] = "select make, model from auto where
make = 'FORD'";
...
if (lError =LINTER_CreateStatement(nConnID, cQuery, 0, &nStmtID))
 processing_error(lError, nConnID, 0, 0,
 "LINTER_CreateStatement");
```

Освобождение оператора – LINTER_FreeStatement

Прототип функции

```
L_LONG LINTER_FreeStatement(
  L_WORD StatementID);      /* идентификатор оператора */
```

Входные параметры

Параметр	Описание
`StatementID`	Идентификатор оператора

Описание

Удаляет оператор из списка операторов этого соединения.

Пример

```
long lError;
short nStmtID;
if (lError =LINTER_FreeStatement(nStmtID))
 processing_error(lError, 0, 0, nStmtID, "LINTER_CreateStatement")
```

Получение характеристик соединения – LINTER_GetConnectOption

Прототип функции

```
L_LONG LINTER_GetConnectOption(
  L_WORD ConnectID,            /* идентификатор соединения */
  L_SWORD OptionType,          /* тип характеристики */
  void *Buffer,                /* буфер значения характеристики */
```

```
  L_LONG *BufLen);              /* длина Buffer в байтах */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
OptionType	Тип характеристики соединения
BufLen	Размер буфера значения характеристики

Выходные параметры

Параметр	Описание
Buffer	Буфер значения характеристики
BufLen	Реальный размер значения характеристики

Описание

Помещает в буфер Buffer значение характеристики. Почти все эти характеристики имеют тип L_LONG, исключение составляет лишь cDBDesc, имеющая тип t_DBDesc.

Примечание
Если адрес BufLen равен NULL, то LinAPI не будет производить проверку того, достаточен ли буфер для приема характеристики.

Пример

```
long lError;
short nConnID;
t_DBDesc dbDesc;
...
if (lError =LINTER_GetConnectOption(nConnID, cDBDesc, (void
 *)&dbDesc, NULL))
 processing_error(lError, nConnID, 0, 0,
 "LINTER_GetConnectOption");
```

Установка характеристик соединения – LINTER_SetConnectOption

Прототип функции

```
L_LONG LINTER_SetConnectOption(
  L_WORD ConnectID,             /* идентификатор соединения */
  L_SWORD OptionType,           /* тип характеристики */
  void *OptionValue             /* значение характеристики */
  L_LONG *ValueLen);            /* длина характеристики */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения

Параметр	Описание
OptionType	Тип характеристики
OptionValue	Значение характеристики
ValueLen	Длина значения характеристики в байтах

Выходные параметры

Отсутствуют.

Описание

Устанавливает характеристики курсора.

Пример

```
long lError;
short nConnID;
short nPriority =10;
...
if (lError =LINTER_SetConnectOption(nConnID, cPriprity,
 &nPriority, NULL))
 processing_error(lError, nConnID, 0, 0,
 "LINTER_SetConnectOption");
```

Получение характеристик курсора – LINTER_GetCursorOption

Прототип функции

```
L_LONG LINTER_GetCursorOption(
  L_WORD CursorID,            /* идентификатор курсора */
  L_SWORD OptionType,         /* тип характеристики */
  L_SWORD ColumnNumber,       /* порядковый номер столбца */
  void *Buffer,               /* буфер значения характеристики */
  L_LONG *BufLen);            /* длина Buffer в байтах */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
OptionType	Тип характеристики (см. приложение 2)
ColumnNumber	Номер столбца ответа (для cAnswerDesc), порядковый номер выходного параметра хранимой процедуры (для cProcArgDesc)
BufLen	Размер буфера значения характеристики

Выходные параметры

Параметр	Описание
Buffer	Буфер значения характеристики

Параметр	Описание
BufLen	Реальный возвращаемый размер характеристики

Описание

Помещает в Buffer значение требуемой характеристики курсора (описание характеристик приведено в приложении 2). Почти все эти характеристики имеют тип long, исключение составляют лишь cAnswerDesc, имеющая тип t_ParamDesc, и cProcArgDesc, имеющая тип t_ProcArgDesc.

Примечание
Если адрес BufLen равен NULL, то LinAPI не будет производить проверку того, достаточен ли буфер для приема характеристики.

Пример

```
long lError;
short nCursID;
t_ParamDesc pdAnsDesc;
...
if(lError =LINTER_GetCursorOption(nCursID,cAnswerDesc, 1,
(void*)&pdAnsDesc, NULL))
  processing_error(lError, 0, nCursID, 0,
 "LINTER_GetCursorOption");
```

Установка характеристик курсора – LINTER_SetCursorOption

Прототип функции

```
L_LONG LINTER_SetCursorOption(
  L_WORD CursorID,            /* идентификатор курсора */
  L_SWORD OptionType,         /* тип характеристики */
  void *OptionValue,          /* значение характеристики */
  L_LONG *ValueLen);          /* длина характеристики */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
OptionType	Тип характеристики
OptionValue	Значение характеристики курсора
	Примечание В текущей версии можно установить только приоритет курсора
ValueLen	Длина значения характеристики в байтах

Выходные параметры

Отсутствуют.

Описание

Устанавливает характеристики курсора.

Пример

```
long lError;
short nCursID;
short nPriority =10;
...
if(lError =LINTER_SetCursorOption(nCursID, cPriority, (void
 *)&nPriority, NULL))
  processing_error(lError, 0, nCursID, 0,
 "LINTER_SetCursorOption");
```

Получение характеристик оператора – LINTER_GetStatementOption

Прототип функции

```
L_LONG LINTER_GetStatementOption(
  L_WORD StatementID,          /* идентификатор оператора */
  L_SWORD OptionType,          /* тип характеристики */
  L_SWORD ParamNumber,         /* номер параметра */
  void *Buffer,                /* буфер значения характеристики */
  L_LONG *BufLen);             /* длина Buffer в байтах */
```

Входные параметры

Параметр	Описание
StatementID	Идентификатор оператора
OptionType	Тип характеристики
ParamNumber	Номер параметра (для sParamDesc)
BufLen	Размер буфера под значение характеристики

Выходные параметры

Параметр	Описание
Buffer	Буфер значения характеристики
BufLen	Реальный возвращаемый размер характеристики

Описание

Помещает в Buffer значение требуемой характеристики. Почти все эти характеристики имеют тип long, исключение составляют sParamDesc и sAnswerDesc, имеющие тип t_ParamDesc.

Пример

```
long lErr;
short StmtID;
long lParCnt;
if(lErr=LINTER_GetStatementOption(StmtID, sParamCount, 0,(void
 *)&lParCnt, NULL))
 processing_error(lErr, 0, 0, StmtID,
 "LINTER_GetStatementOption");
```

Выполнение оператора – LINTER_ExecuteStatement

Прототип функции

```
L_LONG LINTER_ExecuteStatement(
  L_WORD CursorID,              /* идентификатор курсора */
  L_WORD StatementID,           /* идентификатор оператора */
  L_LONG *ExecCount,            /* число выполнений оператора */
  void *AsyncFunc,              /* функция обработки ответа */
  void *UserArg);               /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
StatementID	Идентификатор оператора
ExecCount	Требуемое число выполнений оператора
AsyncFunc	Функция обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Параметр	Описание
ExecCount	Реальное число выполнения оператора

Описание

Выполняет оператор по указанному курсору заданное количество раз. В случае, когда оператор LINTER_BindParameter использовался для привязки массива параметров, предпринимается попытка выполнения оператора LINTER_ExecuteStatement указанное количество раз с каждым параметром из массива поочередно.

Примечание
При пакетной вставке данных триггеры, настроенные на вставку данных, срабатывать не будут.

Пример

```
long lError;
short nCursID;
```



```
short nStmtID;
...
if (lError =LINTER_ExecuteStatement(nCursID, nStmtID, NULL, NULL,
 NULL))
  processing_error(lError, 0, nCursID, 0,
 "LINTER_ExecuteStatement");
```

Параметры запроса и работа с ними

Привязка параметра запроса – LINTER_BindParameter

Прототип функции

```
L_LONG LINTER_BindParameter(
 L_WORD CursorID,          /* идентификатор курсора */
 L_WORD StatementID,       /* идентификатор оператора */
 L_SWORD ParameterNumber, /* номер параметра */
 L_CHAR *ParameterName,    /* имя параметра */
 L_CHAR *NullIndicator,    /* признак NULL-значения */
 void *ParameterAddress,  /* адрес параметра */
 L_LONG ItemCounter,       /* число элементов массива */
 L_LONG Shift,             /* характ-ка элемента массива пар-ов */
 L_SWORD ParameterType,    /* тип параметра */
 L_LONG *Reserved,         /* не используется */
 L_LONG *ParameterLength);/* длина значения параметра */
```

Входные параметры

Параметр	Описание
`CursorID`	Идентификатор курсора
`StatementID`	Идентификатор оператора
`ParameterNumber`	Номер параметра (используется при ParameterName равном NULL). Нумерация параметров с 1
`ParameterName`	Адрес строки (заканчивается нулевым символом), содержащей имя параметра
`NullIndicator`	Флаг NULL-значения
`ParameterAddress`	Адрес буфера значений/адресов значений параметров
`ItemCounter`	Количество параметров в массиве
`Shift`	Характеристика элемента массива параметров: если значение >= 0, то переменная `Shift` задаёт размер элемента массива параметров, а если значение < 0, то `ParameterAddress` задает массив указателей значений параметров
`ParameterType`	Тип параметра
`ParameterLength`	Адрес длины значения параметра

Выходные параметры

Отсутствуют.

Описание

Привязывает буфер параметра к оператору. При выполнении оператора значение из этого буфера будет подставлено на место соответствующего параметра. Пользователь может (используя `LINTER_CreateStatement`) создавать запросы с параметрами, как именованными, так и неименованными.

Можно производить привязку параметров как массива указателей. Для этого аргумент `Shift` должен быть равен -1.

Примеры

1) Использование именованного параметра:

```
SELECT * FROM auto WHERE make = :MAKE;
```

2) Использование неименованного параметра:

```
SELECT * FROM auto where make = ?;
INSERT INTO auto(PersonID, Make) VALUES (?, ?);
```

Имеется возможность привязать для одного параметра несколько значений. Для этого нужно привязать этому параметру сразу целый массив из `ItemCounter`.

`ParameterLength` также должен быть представлен в виде массива значений типа `long`. При вызове функции `LINTER_ExecuteStatement` оператор будет выполнен для всех значений параметра.

Если параметр именованный, то его привязка может осуществляться как по имени, так и по номеру. Запрос может одновременно содержать и именованные, и неименованные параметры.

Примечание

В LinAPI при работе с СУБД ЛИНТЕР параметры могут быть трёх типов:

1) входными (`pInput`);
2) выходными (`pOutput`);
3) входными и выходными (`pInputOutput`).

Битовые маски `pInput`, `pOutput` и `pInputOutput` определены в заголовочном файле `linapi.h`. Установка необходимого типа параметра осуществляется посредством объединения типа значения параметра с соответствующей битовой маской (`pInput`, `pOutput` или `pInputOutput`) побитовой операцией «ИЛИ».

Примечание

Если параметр имеет строковый тип (`t_String`), то значение `ParameterLength` может быть любым. В этом случае значение параметра может считаться строкой, заканчивающейся двоичным нулем (ASCIZ-строка).

Пример

```
long lErr;
short nCnnID;
short nCrsID;
```



```
short nStmtID;
char cPar[] ='FORD';
long lParL;
char cQuery[]="select make from auto where make = :MAKE";
...
if(lErr=LINTER_CreateStatement(nCnnID, cQuery, 0, &nStmtID))
 processing_error(lErr, nCnnID, 0, 0, "LINTER_CreateStatement");
lParL =sizeof(cPar);
if(lErr=LINTER_BindParameter(nCrsID,nStmtID,0,"MAKE", NULL,
(void*)cPar,1,0,tChar, NULL,&lParL))
 processing_error(lErr, 0, nCrsID, 0, "LINTER_BindParameter");
```

Привязка поля ответа – LINTER_BindAnswer

Прототип функции

```
L_LONG LINTER_BindAnswer(
 L_WORD CursorID,       /* идентификатор курсора */
 L_WORD StatementID,    /* идентификатор оператора */
 L_SWORD ColumnNumber, /* номер столбца (или поля) ответа */
 void *AnswerBuffer,    /* буфер ответов */
 L_CHAR *NullIndicator,/* буфер NULL-индикаторов */
 L_LONG Shift,          /* характ-ка элемента буфера ответов */
 L_SWORD OutType,       /* выходной тип поля ответа */
 L_LONG OutLength,      /* выходная длина поля ответа */
 L_SWORD OutPrec,       /* выходная точность поля (для tDecimal) */
 L_SWORD OutScale,      /* выходной масштаб поля (для tDecimal) */
 L_LONG *RealLength);  /* массив реально принятых длин */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
StatementID	Идентификатор оператора
ColumnNumber	Номер столбца (или поля) ответа
AnswerBuffer	Буфер ответов
NullIndicator	Буфер NULL-индикаторов значений из AnswerBuffer
Shift	Характеристика элемента буфера ответов: если значение больше либо равно нулю, то переменная Shift задаёт размер элемента буфера ответов, а если значение отрицательное, то AnswerBuffer задает буфер указателей значений параметров ответов
OutType	Выходной тип поля ответа
OutLength	Выходная длина поля ответа
OutPrec	Точность (для tDecimal)
OutScale	Масштаб (для tDecimal)
RealLength	Массив для приема реальных длин поля

Выходные параметры

Параметр	Описание
RealLength	Реальная длина ответа. Заполняется после выполнения функции LINTER_Fetch

Описание

Назначает буфер размещения ответов, получаемых с помощью LINTER_Fetch. Назначить этот буфер можно сразу для нескольких ответов.

Например, следующие вызовы:

```
...
typedef char Make_Type[20];
typedef struct {
 Make_Type Make;
 long PersonID;
 } Answ_Type;
Answ_Type Buf[20];
...
LINTER_BindAnswer(3, 7, 1, Buf, NI, sizeof(Answ_Type), tChar,
 sizeof(Make_Type), 0, 0, NULL);
LINTER_BindAnswer(3, 7, 2, (char*)Buf +sizeof(Make_Type), NI,
 sizeof(Answ_Type), tInt, sizeof(long), 0, 0, NULL);
```

или

```
LINTER_BindAnswer(3, 7, 1, Buf.Make, NI, sizeof(Answ_Type), tChar,
 sizeof(Make_Type), 0, 0, NULL);
LINTER_BindAnswer(3, 7, 2, &Buf.PersonID, NI, sizeof(Answ_Type),
 tInt, sizeof(long), 0, 0, NULL);
```

указали LinAPI, что ответ, полученный по 3-ему курсору при выполнении оператора № 7, имеет структуру Answ_Type. Теперь для приема записей может использоваться массив Buf.

Примечание
Привязка поля ответа может быть произведена только после трансляции (случай использования оператора – Statement) или выполнения запроса (посредством функции LINTER_ExecuteDirect). Во втором случае StatementID должен быть равным 0.

Отсоединение буфера столбца ответа – LINTER_UnBindAnswer

Прототип функции

```
L_LONG LINTER_UnBindAnswer(
  L_WORD CursorID,         /* идентификатор курсора */
```



```
  L_WORD StatementID,       /* идентификатор оператора */
  L_SWORD ColumnNumber);    /* номер столбца ответа */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
StatementID	Идентификатор оператора
ColumnNumber	Номер столбца ответа

Выходные параметры

Отсутствуют.

Описание

Отсоединяет (привязанный LINTER_BindAnswer) буфер, не производя над ним никаких действий. При следующем вызове LINTER_Fetch ответ в этот буфер помещаться не будет. Если значение ColumnNumber равно нулю, то будут сняты все буфера.

Пример

```
long lError;
short nCursID;
short nStmtID;
...
if (lError =LINTER_UnBindAnswer(nCursID, nStmtID, 1))
  processing_error(lError, 0, nCursID, 0, "LINTER_UnBindAnswer");
```

Транзакции и блокировки

Фиксация изменений по курсору – LINTER_CommitCursor

Прототип функции

```
L_LONG LINTER_CommitCursor(
  L_WORD CursorID,          /* идентификатор курсора */
  void *AsyncFunc,          /* функция обработки ответа */
  void *UserArg);           /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
AsyncFunc	Адрес функции обработки ответа
UserArg	Аргумент пользователя

Выходные параметры

Отсутствуют.

Описание

Фиксирует в БД результаты транзакции.

Пример

```
long lError;
short nCursID;
...
if (lError =LINTER_CommitCursor(nCursID, NULL, NULL))
  Processing_error(lError, 0, nCursID, 0, "LINTER_CommitCursor");
```

Фиксация изменений по соединению – LINTER_Commit

Прототип функции

```
L_LONG LINTER_Commit(
  L_WORD ConnectID,  /* идентификатор соединения */
  void *AsyncFunc,   /* функция обработки ответа */
  void *UserArg);    /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
AsyncFunc	Адрес функции обработки ответа
UserArg	Аргумент пользователя

Выходные параметры

Отсутствуют.

Описание

Фиксирует в БД результаты транзакций всех курсоров соединения ConnectID.

Пример

```
long lError;
short nConnID;
...
if (lError =LINTER_Commit(nConnID, NULL, NULL))
 processing_error(lError, nConnID, 0, 0, "LINTER_Commit");
```

Откат изменений по курсору – LINTER_RollBackCursor

Прототип функции

```
L_LONG LINTER_RollBackCursor(
```



```
  L_WORD CursorID,          /* идентификатор соединения */
  void *AsyncFunc,          /* функция обработки ответа */
  void *UserArg);           /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
AsyncFunc	Адрес функции обработки ответа
UserArg	Аргумент пользователя

Выходные параметры

Отсутствуют.

Описание

Откатывает транзакцию по курсору.

Пример

```
long lError;
short nCursID;
...
if (lError =LINTER_RollBackCursor(nCursID, NULL, NULL))
  Processing_error(lError, 0, nCursID, 0,
 "LINTER_RollBackCursor");
```

Откат изменений по соединению – LINTER_RollBack

Прототип функции

```
L_LONG LINTER_RollBack(
  L_WORD ConnectID,     /* идентификатор соединения */
  void *AsyncFunc,      /* функция обработки ответа */
  void *UserArg);       /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
AsyncFunc	Адрес функции обработки ответа
UserArg	Аргумент пользователя

Выходные параметры

Отсутствуют.

Описание

Откатывает транзакции всех курсоров соединения.

Пример

```
long lError;
short nConnID;
...
if (lError =LINTER_RollBack(nConnID, NULL, NULL))
 Processing_error(lError, nConnID, 0, 0, "LINTER_RollBack");
```

Блокировка строки – LINTER_LockRow

Прототип функции

```
L_LONG LINTER_LockRow(
  L_WORD CursorID);   /* идентификатор курсора */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Выходные параметры

Отсутствуют.

Описание

Блокирует текущую строку курсора.

Пример

```
long lError;
short nCursID;
...
if (lError =LINTER_LockRow(nCursID))
  processing_error(lError, 0, nCursID, 0, "LINTER_LockRow");
```

Разблокировка строки – LINTER_UnlockRow

Прототип функции

```
L_LONG LINTER_UnlockRow(
  L_WORD CursorID);     /* идентификатор курсора */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Выходные параметры

Отсутствуют.

Описание

Выполняет разблокирование текущей строки курсора.

Пример

```
long lError;
short nCursID;
...
if (lError =LINTER_UnlockRow(nCursID))
  processing_error(lError, 0, nCursID, 0, "LINTER_UnlockRow");
```

Выполнение SQL-запроса по курсору – LINTER_ExecuteDirect

Прототип функции

```
L_LONG LINTER_ExecuteDirect(
  L_WORD CursorID,          /* идентификатор курсора */
  L_CHAR *Query,            /* указатель на текст SQL-запроса */
  L_LONG QueryLength,       /* длина запроса */
  void *AsyncFunc,          /* адрес функции обработки ответа */
  void *UserArg);           /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
Query	Указатель на текст запроса
QueryLength	Длина запроса; если QueryLength > 0, то это длина запроса, в противном случае текст запроса должен заканчиваться нулевым символом
AsyncFunc	Адрес функции обработки ответа
UserArg	Адрес пользовательского аргумента функции AsyncFunc

Выходные параметры

Отсутствуют.

Описание

Выполняет запрос Query (используя курсор CursorID).

После выполнения запроса могут быть вызваны функции: LINTER_GetCursorOption, LINTER_BindAnswer, LINTER_Fetch, LINTER_GetRowBuffer.

Пример

```
long lError;
short nCursID;
```

```
char cQuery[] = "select make, model from auto";
...
if (lError =LINTER_ExecuteDirect(nCursID, cQuery, 0, NULL, NULL))
  processing_error(lError, 0, nCursID, 0, "LINTER_ExecuteDirect");
```

Выполнение SQL-запроса по соединению – LINTER_ExecControlQuery

Прототип функции

```
L_LONG LINTER_ExecControlQuery(
  L_WORD Connection,        /* идентификатор соединения */
  L_CHAR *Query,            /* указатель на текст SQL-запроса */
  L_LONG Length,            /* длина запроса */
  void *AstFunction,        /* адрес функции обработки ответа */
  void *UserArg);           /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
`Connection`	Идентификатор соединения
`Query`	Указатель на текст SQL-запроса
`Length`	Длина запроса; если `Length` > 0, то это длина запроса, в противном случае текст запроса должен заканчиваться нулевым символом
`AstFunction`	Адрес функции обработки ответа
`UserArg`	Адрес пользовательского аргумента функции `AstFunction`

Выходные параметры

Отсутствуют.

Описание

Выполняет запрос `Query` по указанному соединению. Ответ на запрос не обрабатывается (за исключением возврата кода завершения в случае внутренней ошибки интерфейса). Функция предназначена для более эффективного выполнения запросов типа установки/смены уровней, уровней изоляции, для работы с контрольными точками транзакций и командами commit/rollback.

Допускается асинхронное выполнение функции.

Отмена операции – LINTER_Cancel

Прототип функции

```
L_LONG LINTER_Cancel(
  L_WORD ConnectID,  /* идентификатор соединения */
  L_WORD CursorID,   /* идентификатор курсора */
  L_LONG Option,     /* параметры операции отмены */
```



```
  L_LONG Reserved);  /* зарезервировано */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
CursorID	Идентификатор курсора LinAPI либо номер канала СУБД ЛИНТЕР
Option	Параметры операции отмены
Reserved	Зарезервировано для дальнейшего использования

Параметр Option представляет битовую маску и может содержать следующие значения, объединяемые побитовым оператором "ИЛИ":

C_CANCEL	0x00000001	Отмена операции, выполняемой по курсору
C_CANCEL_AST	0x00000002	Отмена асинхронной операции, выполняемой по курсору с принудительным вызовом пользовательской функции-обработчика
C_CANCEL_AST_CLOSE_CH	0x00000004	Принудительное закрытие курсора после отмены асинхронной операции; применяется совместно с C_CANCEL_AST

Выходные параметры

Отсутствуют.

Описание

Функция отменяет выполнение SQL-запроса или любой другой операции. Возможна отмена операции, выполняемой по любому каналу СУБД ЛИНТЕР.

Параметр Option может быть равен 0. При этом второй параметр функции CursorID должен содержать номер канала СУБД ЛИНТЕР.

Если установлен бит C_CANCEL, остальные биты игнорируются.

Если установлен бит C_CANCEL_AST, происходит немедленный вызов пользовательской функции-обработчика без ожидания ответа. Пользовательской функции будет передан код eOperationCanceled. Курсор переводится в свободное состояние. Однако дальнейшее использование курсора возможно только после синхронизации с помощью функции LINTER_CursorComplete (предварительно должна быть установлена характеристика курсора cWaitComplete). Вызов любых других функций LinAPI для этого курсора до синхронизации запрещён!

C_CANCEL_AST_CLOSE_CH применяется только совместно с C_CANCEL_AST. Инициирует неявное принудительное закрытие курсора. Дальнейшее использование курсора невозможно.

Примечание
Для отмены синхронных операций параметр Option следует устанавливать в 0 и выполнять функцию LINTER_Cancel для канала СУБД ЛИНТЕР (характеристика курсора cChannelID).

Код eOperationCanceled пользовательской функции может быть передан не всегда. При отмене (в случае Option равном `0` или C_CANCEL) некоторых операций, например, LINTER_Fetch, возможны другие коды завершения. При этом обработчику будет передан код eLINTERError.

Перемещение по выборке – LINTER_Fetch

Прототип функции

```
L_LONG LINTER_Fetch(
  L_WORD CursorID,   /* идентификатор курсора */
  L_SWORD Direction, /* направление перемещения */
  L_LONG Position,   /* номер строки в выборке */
  L_LONG RowCounter, /* число требуемых ответов */
  L_LONG *RowAnswer, /* число полученных ответов */
  void *AsyncFunc,   /* функция обработки ответа */
  void *UserArg);    /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
`CursorID`	Идентификатор курсора
`Direction`	Направление перемещения
`Position`	Номер строки
`RowCounter`	Требуемое количество строк в выборке (неположительный `RowCounter` преобразуется к 1)
`AsyncFunc`	Адрес функции обработки ответа
`UserArg`	Аргумент пользователя

Выходные параметры

Параметр	Описание
`RowAnswer`	Реальное количество строк, занесенных в буфер ответа

Описание

Помещает в буфер ответа указанное количество записей ответа, начиная с той, которая определена направлением `Direction` (для направлений `toAbsNumber` и `toRelNumber` учитывается еще и номер `Position`).

Примечание
Текущей становится строка, находящаяся в буфере последней.

Направления, возможные для описываемой функции, приведены в таблице 3.

Таблица 3. Идентификаторы направлений для LINTER_Fetch

Обозначение	Направление	Номер текущей строки
toNext	На следующую запись	CurrNumber+RowAnswer
toPrevious	На предыдущую запись	CurrNumber-1+RowAnswer-1
toFirst	На первую запись	RowAnswer
toLast	На последнюю запись	LastNumber
toAbsNumber	На запись с указанным абсолютным номером в ответе (по абсолютному номеру)	Position+RowAnswer-1
toRelNumber	На запись с указанным номером относительно текущей записи (по относительному номеру)	CurrNumber+Position +RowAnswer-1
toFromEnd	С конца	LastNumber

Пример

```
long lError
short nCursID;
...
if (lError =LINTER_Fetch(nCursID, toNext, 0, 20, NULL, NULL,
 NULL))
  processing_error(lError, 0, nCursID, 0, "LINTER_Fetch");
```

Получение строки ответа в буфер – LINTER_GetRowBuffer

Прототип функции

```
L_LONG LINTER_GetRowBuffer(
  L_WORD CursorID,          /* идентификатор курсора */
  void *Buffer,             /* буфер для приема записи */
  L_LONG *BufLen);          /* длина Buffer в байтах */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
Buffer	Адрес буфера записи ответа
BufLen	Адрес длины буфера в байтах

Выходные параметры

Параметр	Описание
BufLen	Адрес реального размера записи

Описание

Помещает в `Buffer` запись ответа без преобразования. Эту функцию целесообразно использовать тогда, когда известна структура ответа (см. документ «Интерфейс нижнего уровня»).

Пример

```
long lError;
short nCursID;
long lAnsLen;
void *vBuffer;
...
if(lError =LINTER_GetCursorOption(nCursID, cAnswerSize, 0, (void*)
 &lAnsLen, NULL))
  processing_error(lError, 0, nCursID, 0,
 "LINTER_GetCursorOption");
vBuffer = calloc(lAnsLen, sizeof(char));
...
if(lError =LINTER_GetRowBuffer(nCursID, (void *) vBuffer,
 &lAnsLen))
  processing_error(lError, 0, nCursID, 0, "LINTER_GetRowBuffer");
```

Получение столбца ответа в буфер – LINTER_GetData

Прототип функции

```
L_LONG LINTER_GetData(
 L_WORD CursorID,      /* идентификатор курсора */
 L_SWORD ColumnNumber,/* номер столбца в ответе */
 void *OutBuffer,      /* буфер для приема поля */
 L_LONG OutBufLen,     /* длина OutBuffer в байтах */
 L_SWORD OutType,      /* выходной тип поля ответа */
 L_SWORD OutPrec,      /* выходная точность поля (для tDecimal) */
 L_SWORD OutScale,     /* выходной масштаб поля (для tDecimal) */
 L_LONG *RealLength); /* реальная длина поля */
```

Входные параметры

Параметр	Описание
`CursorID`	Идентификатор курсора
`ColumnNumber`	Номер столбца ответа
`OutBuffer`	Буфер для приема поля
`OutBufLen`	Длина буфера `OutBuffer`
`OutType`	Требуемый тип ответа
`OutPrec`	Точность (для tDecimal)
`OutScale`	Масштаб (для tDecimal)

Выходные параметры

Параметр	Описание
OutBuffer	Выходной буфер ответа
RealLength	Реальная длина ответа

Описание

Помещает в OutBuffer значение столбца ответа, а в RealLength действительный размер ответа. Для получения значений переменной длины размер буфера ответа должен быть увеличен на 2 байта (для указания реальной длины значения).

Пример

```
long lError;
short nCursID;
t_ParamDesc pdAnsDesc;
void *vBuff;
...
if(lErr=LINTER_GetCursorOption(nCrsID, cAnswerDesc, 1,
 (void*)&pdAnsDesc, NULL))
 processing_error(lErr, 0, nCrsID, 0, "LINTER_GetCursorOption");
vBuff = calloc(pdAnsDesc.Length, sizeof(char));
...
if(lErr=LINTER_GetData(nCrsID, 1, vBuff, pdAnsDesc.Length,
 pdAnsDesc.Type, 0, 0, NULL))
 processing_error(lErr, 0, nCrsID, 0, "LINTER_GetData");
```

Работа с BLOB-значениями

Общее замечание о функциях

Ранние версии СУБД ЛИНТЕР поддерживали только один BLOB-столбец в таблице, поэтому функции для работы с BLOB-значениями не требовали указания столбца, к которому должна быть применена операция. Со временем СУБД ЛИНТЕР стала поддерживать несколько BLOB-столбцов в таблице, соответственно, был разработан новый набор функций для обработки BLOB-значений, который покрывал, среди прочего, функциональность устаревших функций. В данном разделе описаны как старые, так и новые функции для работы с BLOB-значениями. Соответствие между старыми и новыми BLOB-функциями приведено в таблице 4.

Таблица 4. Соответствие между новыми и старыми BLOB-функциями

Устаревшие функции	Эквивалентные функции
LINTER_ClearBlob	LINTER_PurgeBlob
LINTER_GetBlobType	LINTER_GetCursorOption с параметром cBlobType
LINTER_GetBlobLength	LINTER_GetCursorOption с параметром cBlobLength
LINTER_AppendBlob	LINTER_AddBlob2

Устаревшие функции	Эквивалентные функции
LINTER_GetBlob	LINTER_FetchBlob
LINTER_SetBlobType (устаревшая в принципе)	LINTER_AddBlob2

BLOB-функции применяются к текущей записи из предшествующего запроса выборки (SELECT-запроса) или к текущей добавленной записи (INSERT-запрос), поэтому в этих функциях параметр «номер BLOB-столбца» относится не к исходной таблице, а к номеру столбца в текущей записи предшествовавшего SELECT-запроса или к текущей добавленной записи INSERT-запроса.

Расширение BLOB-значения – LINTER_AppendBlob

Прототип функции

```
L_LONG LINTER_AppendBlob(
  L_WORD CursorID,       /* идентификатор курсора */
  L_WORD Blob_Type,      /* тип BLOB-значения */
  void *Buffer,          /* буфер с добавляемой порцией */
  L_LONG BufLen,         /* длина добавляемой порции в байтах */
  void *AsyncFunc,       /* функция обработки ответа */
  void *UserArg);        /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
Blob_Type	Тип BLOB-значения
Buffer	Адрес буфера BLOB-значения
BufLen	Длина BLOB-значения
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Отсутствуют.

Описание

Добавляет порцию значения в первое BLOB-поле текущей записи.

Пример

```
long lError;
short nCursID;
long lBufLen;
void *vBuffer;
...
if(lError =LINTER_AppendBlob(nCursID, 0, vBuffer, lBufLen, NULL,
 NULL))
```



```
  processing_error(lError, 0, nCursID, 0, "LINTER_AppendBlob");
```

Очистка BLOB-значения – LINTER_ClearBlob

Прототип функции

```
L_LONG LINTER_ClearBlob(
  L_WORD CursorID,      /* идентификатор курсора */
  void *AsyncFunc,      /* функция обработки ответа */
  void *UserArg);       /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Отсутствуют.

Описание

Очищает первое BLOB-поле текущей записи.

Пример

```
long lError;
short nCursID;
...
if (lError =LINTER_ClearBlob(nCursID, NULL, NULL))
  processing_error(lError, 0, nCursID, 0, "LINTER_ClearBlob");
```

Получение порции BLOB-значения – LINTER_GetBlob

Прототип функции

```
L_LONG LINTER_GetBlob(
  L_WORD CursorID,    /* идентификатор курсора */
  L_LONG OffSet,      /* смещение порции в BLOB-значении */
  L_LONG *Size,       /* размер требуемой порции */
  void *Buffer,       /* буфер для приема порции */
  void *AsyncFunc,    /* функция обработки ответа */
  void *UserArg);     /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Параметр	Описание
OffSet	Смещение требуемой порции в BLOB-значении
Size	Адрес размера (в байтах) требуемой порции
Buffer	Адрес буфера для приема порции BLOB-значения
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Параметр	Описание
Buffer	Возвращаемое значение

Описание

Читает из БД первое BLOB-значение текущей записи, начиная с позиции OffSet (номер первого байта значения), и помещает ответ длиной Size в буфер Buffer.

Типизация BLOB-значения – LINTER_GetBlobType/ SetBlobType

Прототипы функций

```
L_LONG LINTER_GetBlobType(
  L_WORD CursorID,         /* идентификатор курсора */
  L_WORD *Blob_Type);      /* тип BLOB-значения */

L_LONG LINTER_SetBlobType(
  L_WORD CursorID,         /* идентификатор курсора */
  L_WORD Blob_Type);       /* тип BLOB-значения */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
Blob_Type	Тип BLOB-значения (для LINTER_SetBlobType)

Выходные параметры

Параметр	Описание
Blob_Type	Тип BLOB-значения

Описание

Функция LINTER_GetBlobType (LINTER_SetBlobType) возвращает (изменяет) тип первого BLOB-значения текущей записи.

Пример

```
long lError;
```



```
short nCursID;
short nType;
...
if(lError =LINTER_GetBlobType(nCursID, &nType))
  processing_error(lError, 0, nCursID, 0, "LINTER_GetBlobType");
```

Получение размера BLOB-значения – LINTER_GetBlobLength

Прототип функции

```
L_LONG LINTER_GetBlobLength(
  L_WORD CursorID,          /* идентификатор курсора */
  L_LONG *Blob_Size);       /* длина BLOB-значения */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Выходные параметры

Параметр	Описание
Blob_Size	Длина в байтах BLOB-значения

Описание

Возвращает длину первого BLOB-значения текущей записи.

Пример

```
...
long lError;
short nCursID;
long lBlobLength;
...
if(lError =LINTER_GetBlobLength(nCursID, &lBlobLength))
  processing_error(lError, 0, nCursID, 0, "LINTER_GetBlobLength");
```

Очистка BLOB-значения заданного столбца – LINTER_PurgeBlob

Прототип функции

```
L_lONG LINTER_PurgeBlob(
  L_WORD CursorID,      /* идентификатор курсора */
  L_SWORD ColumnNumber, /* номер столбца */
  void *AsyncFunc,      /* функция обработки ответа */
  void *UserArg);       /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
ColumnNumber	Номер столбца таблицы
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Отсутствуют.

Описание

Очищает BLOB-поле текущей записи заданного столбца из предшествующего SELECT/INSERT-запроса.

Расширение BLOB-значения заданного столбца – LINTER_AddBlob

Прототип функции

```
L_LONG LINTER_AddBlob(
  L_WORD CursorID,       /* идентификатор курсора */
  L_SWORD ColumnNumber, /* номер столбца */
  L_WORD Blob_Type,      /* тип BLOB-значения */
  void *Buffer,          /* буфер с добавляемой порцией */
  L_LONG BufLen,         /* длина добавляемой порции в байтах */
  void *AsyncFunc,       /* функция обработки ответа */
  void *UserArg);        /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
ColumnNumber	Номер столбца в таблице
Blob_Type	Тип BLOB-значения
Buffer	Адрес буфера BLOB-значения
BufLen	Длина BLOB-значения
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Отсутствуют.

Описание

Добавляет порцию значения в BLOB-поле текущей записи заданного столбца из предшествующего SELECT/INSERT-запроса.

Длинное расширение BLOB-значения заданного столбца – LINTER_AddBlob2

Прототип функции

```
L_LONG LINTER_AddBlob2(
  L_WORD CursorID,        /* идентификатор курсора */
  L_SWORD ColumnNumber,   /* номер столбца */
  L_LONG Blob_Type,       /* тип BLOB-значения */
  void *Buffer,           /* буфер с добавляемой порцией */
  L_LONG BufLen,          /* длина добавляемой порции в байтах */
  void *Adr,              /* должен быть NULL */
  void *AsyncFunc,        /* функция обработки ответа */
  void *UserArg);         /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
ColumnNumber	Номер столбца в таблице
Blob_Type	Тип BLOB-значения
Buffer	Адрес буфера BLOB-значения
BufLen	Длина BLOB-значения
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Отсутствуют.

Описание

Добавляет порцию значения в BLOB-поле текущей записи заданного столбца из предшествующего SELECT/INSERT-запроса.

Получение порции BLOB-значения заданного столбца – LINTER_FetchBlob

Прототип функции

```
L_LONG LINTER_FetchBlob(
 L_WORD CursorID,       /* идентификатор курсора */
 L_SWORD ColumnNumber,  /* номер столбца */
 L_LONG OffSet,         /* номер первого байта извлекаемой порции
 BLOB-значения */
 L_LONG *Size,          /* размер требуемой порции */
 void *Buffer,          /* буфер для приема порции */
 void *AsyncFunc,       /* функция обработки ответа */
```

```
 void *UserArg);        /* пользовательский аргумент */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора
ColumnNumber	Номер столбца таблицы
OffSet	Номер первого байта извлекаемой порции BLOB-значения
Size	Адрес размера (в байтах) требуемой порции
Buffer	Адрес буфера для приема порции BLOB-значения
AsyncFunc	Адрес функции обработки ответа
UserArg	Пользовательский аргумент

Выходные параметры

Параметр	Описание
Buffer	Возвращаемое значение

Описание

Читает из БД BLOB-значение заданного столбца ColumnNumber из предшествующего SELECT-запроса, начиная с позиции OffSet (номер первого байта значения), и помещает ответ длиной Size в буфер Buffer.

Разбор сообщений LinAPI

Получение сообщения LinAPI – LINTER_ErrorMessage

Прототип функции

```
L_LONG LINTER_ErrorMessage(
  L_LONG ApiError,          /* код завершения LinAPI */
  L_LONG LinError,          /* код завершения ЛИНТЕР */
  L_CHAR *Message,          /* буфер сообщения */
  L_SWORD *MessLen);        /* длина буфера Message */
```

Входные параметры

Параметр	Описание
ApiError	Код сообщения LinAPI
LinError	Код сообщения СУБД ЛИНТЕР
MessLen	Размер (в байтах) буфера Message

Выходные параметры

Параметр	Описание
Message	Буфер сообщения

Параметр	Описание
MessLen	Размер (в байтах) сообщения

Описание

Помещает в Message сообщение об указанной ошибке.

Примечание
Функция оставлена для совместимости с предыдущей версией. В последующих версиях поддерживаться не будет.

Получение кодов завершения – LINTER_Error

Прототип функции

```
L_LONG LINTER_Error(
  L_WORD ConnectID,     /* идентификатор соединения */
  L_WORD CursorID,      /* идентификатор курсора */
  L_WORD StatementID,   /* идентификатор оператора */
  L_LONG *ApiCode,      /* код завершения LinAPI */
  L_LONG *LinCode,      /* код завершения ЛИНТЕР */
  L_LONG *SysCode,      /* код завершения ОС */
  L_CHAR *Message,      /* буфер сообщения */
  L_SWORD *MessLen);    /* длина буфера сообщения */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
CursorID	Идентификатор курсора
StatementID	Идентификатор оператора
MessLen	Размер (в байтах) буфера Message

Выходные параметры

Параметр	Описание
ApiCode	Код завершения LinAPI
LinCode	Код завершения СУБД ЛИНТЕР
SysCode	Код ошибки операционной среды сервера
Message	Буфер сообщения
MessLen	Размер (в байтах) сообщения

Описание

Помещает в выходные параметры (если они отличны от NULL-значения) коды завершения LinAPI, СУБД ЛИНТЕР и операционной системы, а также сообщение, соответствующее коду завершения LinAPI или СУБД ЛИНТЕР.

Коды завершения LinAPI приведены в приложении 1.

Параметр `SysCode` в некоторых случаях несет в себе совсем иное содержание:

- при ошибке трансляции запроса (коды завершения СУБД ЛИНТЕР от 2000 до 2999) в `SysCode` будет представлена дополнительная информация о местоположении ошибки: первые два байта – номер строки в тексте SQL-запроса, последующие два байта – номер позиции в строке, где выявлен ошибочный фрагмент;
- при ошибках выполнения оператора с кодами завершения СУБД ЛИНТЕР 1051, 1052 и 1055 в `SysCode` находится номер параметра, при обработке которого произошла ошибка.

Если необходимо получить ошибки по одному объекту (соединению, курсору или оператору), то указывается его ID. Остальные идентификаторы должны быть равны нулю.

Параметры `LinCode` и `SysCode` несут в себе полезную информацию, только если код ошибки LinAPI равен `eLINTERError`.

В пользовательском приложении лучше определить собственную функцию обработки ошибок, которая может иметь вид:

```
void processing_error(
  L_LONG ret_cod,  /* код завершения функции LinAPI */
  L_WORD con_id,   /* ID соединения */
  L_WORD cur_id,   /* ID курсора */
  L_WORD stmt_id,  /* ID оператора */
  L_CHAR *message) /* пользовательское сообщение */
{
  L_LONG lRet;
  L_LONG apierr =0  /* код LinAPI */
  Error =0          /* код ЛИНТЕР */
  Syserr =0         /* код ОС */

  /* ошибка получается только по одному объекту */
  if(ret_cod ==LINAPI_ERROR )
  /* получение кодов ошибок по требуемому объекту */
  if(lRet =LINTER_Error(con_id, cur_id, stmt_id, &apierr, &error,
 &syserr, NULL, NULL))
 printf("\n Diagnostic error: %ld",lRet);
  else
 {
      printf("\n ApiErr=%ld, LinErr=%ld, SysErr=%ld\n%s", apierr,
 error, syserr, message);
      if ((apierr ==eLINTERError) && error > 2000 && error < 3000)
 {
  /* получение строки и позиции при синтаксической ошибке */
  printf("\n Syntax error: line %d, position %d\n", (short)syserr,
 *(short*)((char*)&syserr +2));
  }
```



```
 }
        else
        printf("\n Return code = %ld", ret_cod);
}
```

Проверка завершения операции по соединению – LINTER_ConnectComplete

Прототип функции

```
L_LONG LINTER_ConnectComplete(
  L_WORD ConnectID,           /* идентификатор соединения */
  L_SWORD *IsComplete,        /* флаг завершения */
  L_LONG *ApiCode,            /* код завершения LinAPI */
  L_LONG *LinCode,            /* код завершения ЛИНТЕР */
  L_LONG *SysCode);           /* код завершения ОС */
```

Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения

Выходные параметры

Параметр	Описание
IsComplete	Флаг завершения операции
ApiCode	Код завершения LinAPI
LinCode	Код завершения СУБД ЛИНТЕР
SysCode	Код ошибки операционной среды сервера

Описание

Проверяет завершение операции (по соединению). Если операция завершена, то флаг IsComplete будет отличен от нуля и нулем в противном случае. В случае IsComplete не равно 0 переменные ApiCode, LinCode и SysCode содержат соответствующие коды.

Проверка завершения операции по курсору – LINTER_CursorComplete

Прототип функции

```
L_LONG LINTER_CursorComplete(
  L_WORD CursorID,           /* идентификатор курсора */
  L_SWORD *IsComplete,       /* флаг завершения */
  L_LONG *ApiCode,           /* код завершения LinAPI */
  L_LONG *LinCode,           /* код завершения ЛИНТЕР */
  L_LONG *SysCode);          /* код завершения ОС */
```

Входные параметры

Параметр	Описание
CursorID	Идентификатор курсора

Выходные параметры

Параметр	Описание
IsComplete	Флаг завершения операции
ApiCode	Код завершения LinAPI
LinCode	Код завершения СУБД ЛИНТЕР
SysCode	Код ошибки операционной среды сервера

Описание

Проверяет завершение операции по курсору. Если операция завершена, то значение IsComplete будет отлично от нуля; если не завершена – равно нулю. В случае если IsComplete не равно 0, то переменные ApiCode, LinCode и SysCode содержат код завершения операции соответствующих программных средств.

Пример

```
short nCrsID;
short nIsComplete;
long lErr, lApiErr, lLinErr, lSysErr;
...
if(lErr =LINTER_CursorComplete(nCrsID, &nIsComplete, &lApiErr,
 &lLinErr, &lSysErr))
  processing_error(lError, 0, nCursID, 0, "LINTER_GetBlobLength");
else
  if(!nIsComplete) printf("\n Not complete");
  else
    if(!lApiErr ) printf("\n Complete");
    else
      printf("\n Error: API %ld, LINTER %ld, System %ld", lApiErr,
 lLinErr, lSysErr);
```

Получение описания объекта БД – LINTER_GetObjDesc

Прототип функции

```
L_LONG LINTER_GetObjDesc(
  L_WORD ConnectID,       /* идентификатор соединения */
  L_SWORD ObjType,        /* тип объекта */
  L_CHAR *ObjName,        /* имя объекта */
  L_SWORD NameLen,        /* длина имени объекта */
  void *Buffer,           /* буфер для описания объекта */
  L_LONG *BufLen);        /* длина буфера описания объекта */
```


Входные параметры

Параметр	Описание
ConnectID	Идентификатор соединения
ObjType	Тип объекта (в настоящий момент допустимо использование только типа – oView)
ObjName	Имя объекта (допустимое в ЛИНТЕР-SQL имя объекта)
NameLen	Длина имени объекта
BufLen	Размер буфера описания объекта

Выходные параметры

Параметр	Описание
Buffer	Буфер описания объекта
BufLen	Реальный размер описания объекта

Описание

Помещает в Buffer описание требуемого объекта.

Завершение работы – LINTER_CloseAPI

Прототип функции

```
L_LONG LINTER_CloseAPI (void);
```

Без параметров.

Описание

Закрывает все активные соединения, курсоры, операторы, освобождает все связанные с ними ресурсы.

Программирование с использованием библиотеки LinAPI

Асинхронное программирование

Библиотека LinAPI позволяет реализовывать в приложениях асинхронное выполнение запросов и обработку ответов. Выполнение функции называют асинхронным, если программа продолжает работу, не дожидаясь завершения выполнения функции. Программист может определить свою функцию обработки ответа. В данном случае, когда выполнение функции завершится, работа программы будет прервана, и произойдёт вызов функции обработки ответа, которая может произвести какие-либо действия. Асинхронная работа бывает удобна в тех случаях, когда приложение, кроме работы с БД, ведёт другие активные действия (например, прорисовку изображений или чтение данных с устройств внешней памяти), или приложение посылает к серверу БД несколько вспомогательных запросов, никак не связанных между собой (или наоборот, ответы от которых должны быть обработаны одновременно), или приложение посылает запрос, про который заранее известно, что он будет выполняться долго, а тем временем можно делать что-то другое и т.п. В дистрибутивной поставке СУБД ЛИНТЕР есть пример асинхронного загрузчика данных.

Написание асинхронных программ требует особенно тщательного проектирования приложения, так как последовательное выполнение программы может быть нарушено. Все функции библиотеки LinAPI реентерабельны, и асинхронное выполнение не может привести к порче данных. О реентерабельности своих функций обработки ответа программист должен заботиться сам, т.к. нет гарантий, что при выполнении такой функции не может быть вызвана другая (или та же) функция обработки ответа, работающая с теми же данными. Это похоже на многопоточное программирование, и не реентерабельная функция обработки ответа может привести к неконтролируемому поведению приложения, а в худшем случае к его аварийному завершению.

При асинхронной работе некорректные действия приложения могут привести к ситуации, когда происходит потеря адреса пользовательской функции обработки ответа, и она не будет вызвана. Во избежание подобных ситуаций, если ответ на запрос не приходит слишком долго, имеет смысл предусмотреть вызов функции `LINTER_ConnectComplete`/`LINTER_CursorComplete` в режиме ожидания ответа.

Если `LINTER_ConnectComplete`/`LINTER_CursorComplete` флаг `IsComplete` выставит равным нулю, следовательно, ответ пришёл, но по каким-то причинам не был обработан. Это значит, что приложение произвело некорректную операцию и должно быть завершено.

Существует приём программирования с использованием асинхронных функций, когда создаётся неявный цикл асинхронных вызовов. Если в теле функции обработки ответа есть асинхронный вызов функции LinAPI с указанием той же функции обработки ответа, то произойдет зацикливание в кольцо асинхронных вызовов. То есть самый первый асинхронный вызов функции LinAPI создаст как бы подпроцесс (поток), который будет прерывать выполнение основной программы только на время выполнения функции обработки ответа. Функция обработки ответа должна быть написана так, чтобы асинхронный вызов стал последним, и после него следовал выход из функции обработки ответа. В противном случае может возникнуть ситуация, когда функция обработки ответа ещё не завершена, а уже пришёл второй ответ и т.д., что приведёт к переполнению очереди асинхронных запросов.

Примечание
Из функции обработки ответа можно производить только **асинхронный** вызов функции LinAPI. Синхронный вызов делать нельзя, это приведёт к конфликту и зависанию приложения.

Работа с хранимыми процедурами

В LinAPI реализован полный набор функций для удобной и гибкой работы с хранимыми процедурами СУБД ЛИНТЕР. Библиотека LinAPI позволяет создавать процедуры, обрабатывать ошибки трансляции процедур, выполнять хранимые процедуры, получать ответы (в том числе возвращаемые курсоры).

Для вызова хранимых процедур можно использовать оператор EXECUTE/CALL (эти два ключевых слова являются синонимами).

Обработка ошибок трансляции

Если при создании хранимой процедуры произошла ошибка трансляции, то будет возвращен код завершения 7200 СУБД ЛИНТЕР («Ошибка трансляции хранимой процедуры»).

Ошибок трансляции процедуры может быть несколько. Чтобы получить все ошибки, необходимо вызывать функцию `LINTER_Error` до тех пор, пока функция `LINTER_Error` не завершится ошибкой LinAPI `eNoMoreErrors` (ошибок больше нет).

При помощи функции `LINTER_GetCursorOption` можно узнать количество возникших при трансляции ошибок (характеристика курсора `cProcErrNum`).

Можно получить конкретную ошибку по её номеру. Для этого необходимо установить номер ошибки функцией `LINTER_SetCursorOption` (характеристика курсора `cProcErrNum`) и, затем, вызвать функцию `LINTER_Error`, которая вернёт требуемую ошибку (повторный вызов `LINTER_Error` вернёт следующую по номеру ошибку). Получать ошибки можно в произвольном порядке произвольное число раз.

Обработка возвращаемого значения и выходных параметров

Получить возвращаемое значение и выходные параметры можно тремя различными способами:

1) с помощью функции `LINTER_GetData`;
2) с помощью функции `LINTER_BindParameter` (при использовании претранслированных запросов с параметрами);
3) самостоятельным разбором буфера ответа.

Первый способ можно использовать всегда, и когда запрос на выполнение процедуры был претранслирован, и когда он выполнялся непосредственно. После выполнения процедуры можно узнать количество пришедших в ответе выходных параметров, используя функцию `LINTER_GetCursorOption` (характеристика `cProcArgNum`). Возвращаемое значение в количество выходных параметров не входит, так как оно есть всегда, и имеет нулевой номер. С помощью характеристики курсора `cAnswerDesc`

можно получить описание возвращаемого значения и каждого выходного параметра по номеру. Если выполнялся отладочный вариант процедуры, то можно узнать имя выходного параметра, используя характеристику `cProcArgName` (если значение имеет флаг `fName`).

Если для выполнения процедуры был создан оператор (statement) с параметрами, то функцией `LINTER_BindParameter` можно привязать буфера как входных, так и выходных параметров. При привязке буферов значений параметров запроса вместе с типом значения необходимо указать тип параметра `pInput`, `pOutput` или `pInputOutput`. Если параметр имеет тип `pOutput` или `pInputOutput`, то после выполнения оператора в привязанные буфера будут помещены значения выходных параметров.

Третий способ подразумевает получение буфера ответа с помощью функции `LINTER_GetRowBuffer` и самостоятельный его разбор на низком уровне. Для этого необходимо тщательно изучить соответствующие главы документации по использованию хранимых процедур в СУБД ЛИНТЕР.

Обработка возвращаемого значения типа «курсор»

Если во флаге возвращаемого значения стоит признак `fCursor`, это значит, что возвращаемое значение является курсором. В данном случае при получении значения функцией `LINTER_GetData` будет возвращен идентификатор курсора в LinAPI. При получении значение должно иметь целый тип `tSmallInt` или `tInteger`. Полученный идентификатор может использоваться в LinAPI как обычный идентификатор курсора. С этим курсором можно производить любые операции, возможные в LinAPI. Исключение составляют характеристики курсора `cCursorName`, `cTransMode`, `cPriority`, истинные значения которых получить нельзя. По окончании работы с курсором он должен быть закрыт функцией `LINTER_CloseCursor`.

Обработка результатов выполнения

Для получения результатов выполнения хранимой процедуры выполнить функцию `LINTER_GetCursorOption` со следующими параметрами:

- тип характеристики – `cProcArgDesc`;
- буфер ответа – адрес структуры ARGPROC_OUT;
- номер параметра хранимой процедуры (`Column number`) должен быть равен 0 («результат хранимой процедуры»).

Проверить код завершения функции `LINTER_GetCursorOption`:

- если функция завершилась успешно, то структура ARGPROC_OUT содержит описание результата;
- если функция завершилась с кодом `LINAPI_ERROR` (не успешно), то это означает, что процедура закончилась с исключением. При этом код завершения ядра СУБД ЛИНТЕР равен 7201 («`Процедура завершилась с исключением`»), а системный код завершения – номер сгенерированного процедурой исключения;
- в случае отрицательных значений номеров исключений, меньших -13, исключение является пользовательским (CUSTOM) с номером на 1 больше, чем значение системного кода *-1 (например, в случае пользовательского исключения 2005 возвращаемое значение номера исключения будет -2004).

Установка бита `fNULL` в флагах описания результата (поле `Flags`) означает возврат NULL-значения.

Если установлен бит `fCursor` флагов дескриптора результата, то результат является курсором.

Получить результат выполнения хранимой процедуры и возвращаемые ею параметры, можно также с помощью функции `LINTER_GetData`. В случае если результатом выполнения процедуры является курсор, `LINTER_GetData` возвращает идентификатор курсора, соответствующего результату.

Этот идентификатор может быть использован для выполнения `LINTER_Fetch`, `LINTER_GetData`, `LINTER_GetRowBuffer` и т. п.

Приложение 1
Коды завершения LinAPI

Идентификатор	Числовое значение	Сообщение
eLinterError	14000	Ошибка СУБД ЛИНТЕР
eNoMemory	14001	Нехватка памяти для работы LinAPI
eNullPointer	14003	Неверный (нулевой) указатель
eIllealSequence	14004	Неверная последовательность действий (например, GetData до Execute)
eSmallBuffer	14005	Недостаточен размер буфера
eIllegalDirection	14006	Непредусмотренное (в LinAPI) направление
eFunctionNotAvailable	14007	Функция (в данной версии ЛИНТЕР) не реализована
eIllegalParamNumber	14009	Неверный номер параметра
eIllegalParam	14010	Не найден указанный параметр
eInvalidBlobOperation	14011	Неверная операция для BLOB-столбца
eExistsActiveChannel	14012	Существует активный канал
eMaxIDNumberExceeded	14013	Исчерпано все количество идентификаторов
eInvalidContext	14014	Внутренний сбой в работе (фатальная ошибка)
eErrorInternalDiagnostic	14015	Ошибка внутренней диагностики
eOperationCanceled	14016	Операция прервана
eConnectNotFound	14100	Не найдено указанное соединение
eTooLongUserName	14101	Длинное имя пользователя
eTooLongPassWord	14102	Длинный пароль пользователя
eTooLongServerName	14103	Длинное имя сервера
eIllegalConnectOption	14104	Непредусмотренный (в LinAPI) тип характеристики соединения
eConnectIsBusy	14105	Соединение занято
eIncompatibilityVersions	14106	Несовместимость версий
eWrongLocale	14107	Ошибочная кодировка на сервере
eCharsetChanged	14108	Кодировка была изменена
eTooLongCfgName	14109	Слишком длинное имя конфигурации
eEnvVarNotDef	14110	Переменная среды окружения не определена
eIncEnvVar	14111	Переменная среды окружения задана некорректно
eInvConnStr	14112	Строка с параметрами соединения некорректна
eEnvVarError	14113	Ошибка установки переменной среды окружения

Идентификатор	Числовое значение	Сообщение
eErrOpenCfgFile	14114	Ошибка открытия файла конфигурации
eCursorNotFound	14200	Не найден указанный курсор
eIllegalCursorOption	14201	Непредусмотренный (в LinAPI) тип характеристики курсора
eCursorIsBusy	14202	Курсор занят
eStatementNotFound	14300	Не найден указанный оператор
eIllegalStatementOption	14301	Непредусмотренный (в LinAPI) тип характеристики оператора
eInvalidConnectID	14302	Неверный номер соединения
eStatementIsBusy	14303	Оператор занят
eQueryTooLong	14400	Длинный запрос (длиннее 4096 байтов)
eQueryNotSelect	14401	Не было SELECT-запроса
eNoBlobColumn	14403	BLOB-столбец отсутствует
eIllegalObjectType	14500	Непредусмотренный тип объекта
eObjectNotView	14501	Объект не является представлением
eNoMoreErrors	14601	Список ошибок исчерпан (в ответ на запрос обо всех ошибках, обнаруженных при выполнении процедуры)
eNoProcArgName	14602	Неизвестное имя аргумента процедуры

Приложение 2
Характеристики объектов LinAPI

Характеристики соединения, которые можно получить

Характеристика	Тип данных	Описание
cDBDesc	L_WORD	Информация о БД, с которой связан ЛИНТЕР-сервер
cNodeName	L_CHAR(8)	Имя ЛИНТЕР-сервера
cConnFlags	L_LONG	Режимы работы по соединению (транзакции и кодировки)
cChannelID	L_LONG	Идентификатор соединения

Характеристики соединения/курсора, которые можно получить/установить

Характеристика	Тип данных	Свойство	Описание
cPriority	L_LONG	R/W	Приоритет соединения
cApiCode	L_LONG	R	Код сообщения LinAPI
cLinCode	L_LONG	R	Код завершения СУБД ЛИНТЕР последней операции по соединению
cSysCode	L_LONG	R	Код сообщения ОС, сопровождающий код завершения СУБД ЛИНТЕР
cStrNumber	L_LONG	R	Номер строки запроса, где встречена ошибка трансляции
cPosNumber	L_LONG	R	Номер позиции в строке запроса, где встречена ошибка трансляции
cTransMode	L_LONG	R	Режим обработки транзакций соединения
cWaitComplete	L_BOOL	W	Ждать/не ждать завершения асинхронной операции
cWaitTimeout	L_LONG	W	Длительность ожидания завершения асинхронной операции
cUserData	void *	R/W	Пользовательское значение параметра асинхронной функции

Характеристики курсора, которые можно получить/установить

Характеристика	Тип данных	Свойство	Описание
cSelectRowCount	L_LONG	R	Число записей в выборке данных
cRowCount	L_LONG	R	Количество записей, обработанных последним SQL-оператором
cColumnCount	L_LONG	R	Количество столбцов в выборке данных
cStmtType	L_LONG	R	Тип последнего выполненного оператора (см. Идентификаторы операторов)

Характеристика	Тип данных	Свойство	Описание
cCurrentRow	L_LONG	R	Номер текущей записи в выборке данных
cCurrentRowID	L_LONG	R	**Примечание** Характеристика устарела, использовать не рекомендуется.
cExecStmt	L_LONG	R	Количество выполненных (за один вызов Execute) операторов
cExecRow	L_LONG	R	Количество обработанных (за один вызов Execute) записей
cAnswerDesc	структура t_ParamDesc	R	Описание столбца выборки данных
cAnswerSize	L_LONG	R	Размер записи выборки данных (в байтах)
cStmtNumber	L_LONG	R	Номер текущего оператора
cParamNumber	L_LONG	R	Номер параметра, при обработке которого произошла ошибка
cPriority	L_LONG	R/W	Приоритет курсора
cConnectID	L_LONG	R	Номер соединения, по которому открыт курсор
cTransMode	L_LONG	R	Режим обработки транзакций курсора
cApiCode	L_LONG	R	Код завершения LinAPI
cLinCode	L_LONG	R	Код завершения СУБД ЛИНТЕР последней операции по курсору
cSysCode	L_LONG	R	Код завершения ОС, детализирующий код завершения СУБД ЛИНТЕР
cStrNumber	L_LONG	R	Номер строки запроса, где встречена ошибка трансляции
cPosNumber	L_LONG	R	Номер позиции в строке запроса, где встречена ошибка трансляции
cCursorName	L_CHAR(66)	R/W	Имя курсора
cNullIndicator	L_LONG	R	Признак NULL-значения
cIsAutoInc	L_LONG	R	Признак AUTOINC-значения
cProcErrNum	L_LONG	R/W	Количество ошибок при трансляции процедуры
cProcArgNum	L_LONG	R	Количество аргументов в процедуре
cProcArgDesc	структура ARGPROC_OUT	R	Описание аргумента процедуры
cProcArgName	L_CHAR(66)	R	Имя аргумента процедуры
cProcCursorArg	L_LONG	R	1 – если параметр является курсором, 0 – в противном случае

Характеристика	Тип данных	Свойство	Описание
cWaitComplete	L_BOOL	W	Ждать/не ждать завершения асинхронной операции
cBlobLength	L_LONG	R	Длина BLOB-данных
cBlobType	L_LONG	R	Тип BLOB-данных
cWaitTimeout	L_LONG	W	Длительность ожидания завершения асинхронной операции
cUserData	void *	R/W	Пользовательское значение параметра асинхронной функции
cIsUpdatable	L_LONG	R	Курсорная выборка данных обновляемая
cIsInCallback	L_LONG	R	Признак наличия функции обратного вызова

Характеристики оператора, которые можно получить/установить

Тип характеристики	Тип данных	Свойство	Описание
sParamCount	L_LONG	R	Число параметров в операторе
sAnswerDesc	структура t_ParamDesc	R	Описание столбца выборки данных
sColumnCount	L_LONG	R	Число столбцов в выборке данных
sParamDesc	структура t_ParamDesc	R	Описание параметра
sConnectID	L_LONG	R	Идентификатор соединения, по которому создан оператор
sApiCode	L_LONG	R	Код завершения LinAPI
sIsAutoInc	L_LONG	R	Признак AUTOINC-значения
sStmtType	L_LONG	R	Идентификатор типа оператора (см. Идентификаторы операторов)
sParamType	L_LONG	R	Тип параметра (pInput, pOutput, pInputOutput)
sParamMap	L_LONG	R	Порядковый номер столбца для параметра в выборке данных
sUserData	void *	R/W	Пользовательское значение параметра асинхронной функции

Идентификаторы операторов

Идентификатор оператора	Значение
sySelect	12
syInsert	13
syUpdate	14
syDelete	15
syCreate	16

Идентификатор оператора	Значение
syAlter	17
syDrop	18
syGrant	19
syRevoke	20
sySet	21
syCommit	22
syRollBack	23
syPress	24
syRebuild	25
syWait	26
syClear	27
syLock	28
syUnlock	29
syExecute	30
syStartAppend	31
syEndAppend	32
syTestTable	33
syTable	34
syValues	35
syExecuteBlock	36

Типы данных в LinAPI

Идентификатор типа LinAPI	Соответствующий тип данных call-интерфейса [1)]	Описание данных
tChar	DT_CHAR	Символьная строка (может содержать двоичный нуль)
tByte	DT_BYTE	Буфер байтов фиксированной длины
tString	DT_CHAR	Символьная строка, заканчивающаяся двоичным нулем
tSmallInt	DT_INTEGER	Короткое целое число
tInteger, tInt	DT_INTEGER	Целое число
tReal	DT_REAL	Действительное число
tDouble	DT_REAL	Действительное число двойной точности
tNumeric, tDecimal, tDec	DT_DECIMAL	Действительное число, соответствует типу DECIMAL в СУБД ЛИНТЕР
tDate, tTimeStamp	DT_DATE	Дата + время, тип, соответствующий типу DATE в СУБД ЛИНТЕР
tBlob	DT_BLOB	BLOB-значение
tBigInt	DT_INTEGER	Длинное целое число

Идентификатор типа LinAPI	Соответствующий тип данных call-интерфейса [1)]	Описание данных
tVarChar	DT_VARCHAR	Символьная строка переменной длины (может содержать двоичный нуль)
tVarByte	DT_VARBYTE	Буфер байтов переменной длины
tBoolean	DT_BOOL	Логическое значение
tNChar	DT_NCHAR	Символьная UNICODE-строка фиксированной длины
tNVarChar	DT_NVARCHAR	Символьная UNICODE-строка переменной длины
tExtFile	DT_EXTFILE	Внешний файл

[1)]Типы данных call-интерфейса описаны в документе «Интерфейс нижнего уровня», приложение 3 «Типы данных интерфейса нижнего уровня».

Указатель функций

L