Dokument zawiera opis funkcji i rozszerzeń dla aplikacji graficznych i konsolowych pracujących z oprogramowaniem OTC Terminal.

Terminal GUI
Terminal Console
V. 2.4

Podręcznik programisty

Spis Treści

I. Wstęp

Jednym z głównych celów dla których został opracowany Terminal GUI i Terminal Console było umożliwienie wygodnej i efektywnej eksploatacji dedykowanych aplikacji biznesowych w trybie terminalowym.
Wiele firm posiada kody źródłowe wykorzystywanych systemów transakcyjnych lub ściśle współpracuje z firmami rozwijającymi i dostarczającymi te systemy. Dlatego Terminal GUI/Console został wyposażony w zbiór interfejsów i bibliotek umożliwiających ściślejszą integrację aplikacji ze środowiskiem Terminala. Wykorzystanie tych dodatkowych cech często umożliwia rozwiązanie problemów które w inny sposób nie dają się rozwiązać lub inne rozwiązania są ułomne.

II. Rozszerzanie funkcjonalności aplikacji

Terminal GUI umożliwia uruchomienie istniejących aplikacji bez wprowadzania do nich żadnych zmian. W niektórych przypadkach wskazane jest rozszerzenie możliwości aplikacji poprzez lepsze zintegrowanie jej ze środowiskiem Terminala. Rozszerzenie możliwości aplikacji pracującej w trybie terminalowym jest realizowane poprzez wykorzystanie dodatkowej funkcjonalności Terminala udostępnianej poprzez *interfejs aplikacji*. Interfejs aplikacji umożliwia korzystanie z funkcji należących do dwóch klas:

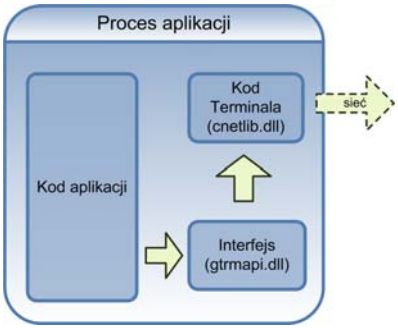
1. funkcji wbudowanych,
2. funkcji użytkownika dołączonych po stronie terminala (RPC).

Korzystanie z *interfejsu aplikacji*

Interfejs aplikacji składa się z biblioteki *gtrmapi.dll* zawierającej funkcje umożliwiające dostęp do rozszerzonej funkcjonalności Terminala. Biblioteka *Gtrmapi.dll* została napisana w języku C i udostępnia (eksportuje) na zewnątrz wszystkie funkcje interfejsu. Funkcje zawarte w bibliotece mogą zostać udostępnione dla aplikacji na dwa sposoby:

1. Poprzez statyczne dołączenie biblioteki *gtrmapi.lib* na etapie łączenia aplikacji. Wówczas biblioteka *gtrmapi.dll* zostanie automatycznie załadowana do pamięci podczas uruchamiania aplikacji. Biblioteka *gtrmapi.lib* nie zawiera żadnych statycznych referencji do funkcji bądź zmiennych Terminala, więc połączone z nią aplikacje mogą być również uruchamiane w trybie nieterminalowym. Dopiero wywołanie funkcji `TApiInitialize()` powoduje dołączenie się biblioteki do Terminala lub ew. zwrócenie błędu jeżeli aplikacja pracuje w trybie nieterminalowym.
2. Poprzez ręczne załadowanie biblioteki do pamięci funkcją `LoadLibrary()` i następnie pobranie adresów funkcji interfejsu funkcją `GetProcAddress()`. Również w tym przypadku konieczne jest wywołanie funkcji `TApiInitialize()` zanim aplikacja rozpocznie korzystanie z interfejsu.

Nazwy funkcji interfejsu aplikacji rozpoczynają się od przedrostka `TApi`... Jeżeli aplikacja została napisana w języku C lub C++, wówczas należy do niej dołączyć plik nagłówkowy *gtrmapi.h* zawierający niezbędne deklaracje funkcji. W przypadku aplikacji opracowanych w innych językach można zastosować inne metody deklarowania i importowania funkcji interfejsu. Ważne jest zachowanie właściwych typów parametrów (zgodnych z *gtrmapi.h*) oraz właściwej konwencji wywołania funkcji (`cdecl`). Rysunek poniżej ilustruje elementy składowe procesu aplikacji wykonującej się w trybie terminalowym z załadowaną biblioteką *gtrmapi.dll*. Strzałki pokazują w jaki sposób realizowane jest wywołanie funkcji interfejsu. W niektórych przypadkach (takich jak zdalne wywołanie procedury) konieczne jest odwołanie się do procesu *gte.exe* działającego na terminalu co ilustruje strzałka przerywana.

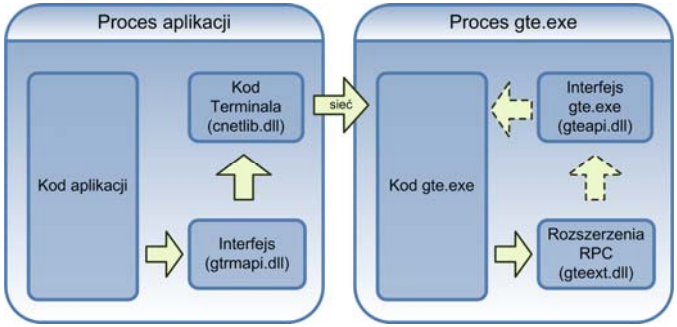


Przykładem aplikacji korzystającej z opisanego interfejsu aplikacji jest aplikacja *testcapi.exe*.
Szczegółowy opis funkcji interfejsu aplikacji znajduje się w rozdziale III na stronie 13.

Zdalne wywołania procedur (RPC)

W środowisku Terminala GUI możliwe jest tworzenie własnych bibliotek DLL zawierających funkcje użytkownika i dołączanie ich do wykonującego się na terminalu *gte.exe*. Odpowiednio przygotowane funkcje mogą być wołane z aplikacji przy pomocy jednej z funkcji: `TApiSyncRPC()`, `TApiSyncRPC_VSR()` i `TApiAsyncRPC()`. Dzięki temu możliwe jest całkowite przeniesienie wykonywania pewnych zadań na terminal. Takie rozwiązanie może być pomocne np.

przy obsłudze niestandardowej metody autentykacji po stronie terminala lub dla efektywnej obsługi drukarki fiskalnej dołączonej do terminala.
Funkcje wywoływane za pośrednictwem mechanizmu RPC muszą spełniać specjalne wymagania odnośnie parametrów i sposobu wywołania. Przykładem biblioteki DLL zawierającej funkcje wołane przez RPC jest biblioteka *gteext.dll*. Przykład wywołania zawartych w niej funkcji znajduje się w aplikacji *testcapi.exe*. Biblioteki DLL zawierające funkcje RPC są ładowane do przestrzeni adresowej *gte.exe* z poziomu aplikacji za pośrednictwem funkcji `TApiRemoteLoadLibraryEx()`. Załadowane w ten sposób biblioteki DLL mają dostęp do pewnych funkcji *gte.exe* za pośrednictwem *interfejsu rozszerzeń* implementowanego przez bibliotekę *gteapi.dll*. Biblioteka ta jest niezbędna jedynie wówczas gdy pisana przez nas biblioteka rozszerzeń (DLL) chce skorzystać z funkcji *gte.exe*. Nazwy funkcji interfejsu rozszerzeń rozpoczynają się od przedrostka `GteApi`... a ich deklaracje można znaleźć w pliku nagłówkowym *gteapi.h*. Biblioteka *gteapi.dll* może zostać dołączona do naszej biblioteki rozszerzeń statycznie przez dołączenie biblioteki *gteapi.lib* lub dynamicznie przez jawne załadowanie funkcją `LoadLibrary()` z naszego kodu inicjalizującego bibliotekę rozszerzeń.



Rysunek powyżej przedstawia proces aplikacji na serwerze oraz proces *gte.exe* wykonujący się na terminalu. Strzałki ilustrują wywołania pomiędzy poszczególnymi elementami procesów podczas realizacji zdalnego wywołania procedury (RPC). Założono, że *gteext.dll* odwołuje się do dodatkowych funkcji *gte.exe*, więc rysunek zawiera również załadowaną bibliotekę *gteapi.dll*. Ponieważ odwołania te nie muszą wystąpić, strzałki są naniesione linią przerywaną. Przedstawiona na rysunku biblioteka *gteext.dll* zawierająca funkcje rozszerzeń użytkownika, w rzeczywistości może się nazywać inaczej. Co więcej, rozszerzenia użytkownika mogą znajdować się

w kilku niezależnych bibliotekach DLL, z których każda przed użyciem powinna zostać załadowana wywołaniem `TApiRemoteLoadLibraryEx()`.

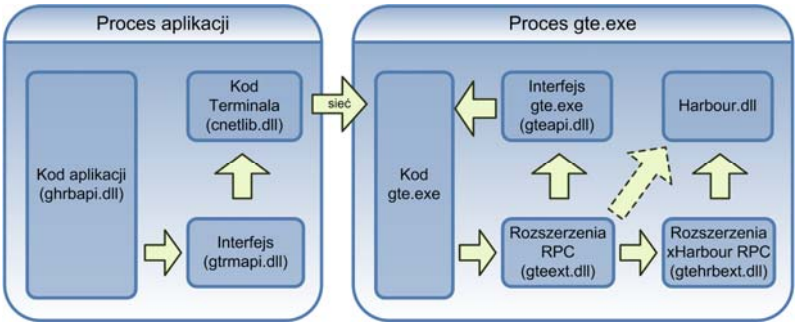
Korzystanie z *interfejsu aplikacji* w aplikacji xHarbour

Aby ułatwić korzystanie z interfejsu aplikacji z poziomu języka xHarbour opracowano bibliotekę *ghrbapi.lib* zawierającą funkcje xHarbour wywołujące funkcje z biblioteki *gtrmapi.dll* (wrapery). Oprócz wraperów do funkcji interfejsu, *ghrbapi.lib* zawiera również funkcje zapewniające zgodność wstecz z funkcjami Terminala OTC, np. `TrmTrmRPC()`, `TrmPutFile()`, `TrmFlPrint()` i inne. Implementacja starych funkcji zdecydowanie ułatwia migrację aplikacji do nowego środowiska Terminala GUI.
Przykładem aplikacji xHarbour korzystającej z interfejsu aplikacji jest aplikacja *testhbapi.exe*.
Szczegółowy opis funkcji xHarbour zawartych w bibliotece *ghrbapi.lib* znajduje się w rozdziale V na stronie 48.

Zdalne wywołania procedur (RPC) w aplikacjach xHarbour

Korzystanie ze zdalnego wywołania procedur z poziomu aplikacji xHarbour zostało istotnie ułatwione poprzez wprowadzenie dołączanej do aplikacji biblioteki *ghrbapi.lib* zawierającej funkcję `TrmTrmRPC()` oraz umożliwienie tworzenia bibliotek rozszerzeń bezpośrednio w języku xHarbour.
Aby umożliwić bezpośrednie wywołania RPC funkcji xHarbour opracowano dodatkową bibliotekę rozszerzeń *gtehrb.dll* realizującą specyficzny dla xHarbour tryb wywołania procedur. Za pośrednictwem funkcji `TrmTrmRPC()` wywoływane mogą być zarówno procedury/funkcje standardowe xHarbour zawarte w *harbour.dll* jak i procedury/funkcje użytkownika zawarte w bibliotece DLL. Przykładem biblioteki rozszerzeń napisanej w xHarbour jest biblioteka *gtehrbext.dll* natomiast przykład jej użycia znajduje się w aplikacji *testhbapi.exe*.
Rysunek poniżej przedstawia proces aplikacji xHarbour na serwerze oraz proces *gte.exe* wykonujący się na terminalu. Strzałki ilustrują wywołania pomiędzy

poszczególnymi elementami procesów podczas realizacji zdalnego wywołania procedury (RPC) xHarbour z poziomu xHarbour. Jak widać kod rozszerzający z biblioteki użytkownika (*gtehrbext.dll*) wołany jest za pośrednictwem standardowej biblioteki rozszerzającej *gtehrb.dll*. W przypadku wywołania procedury użytkownika *gtehrb.dll* odwołuje się do *gtehrbext.dll*. W przypadku wywołania funkcji standardowej (np. SQRT) *gtehrb.dll* odwołuje się do *harbour.dll* (strzałka przerywana). Niezależnie od tego czy wołana jest funkcja standardowa czy użytkownika, biblioteka *harbour.dll* musi być obecna jako środowisko wykonania. Jeżeli wołane będą tylko funkcje standardowe zawarte w *harbour.dll,* wówczas biblioteka *gtehrbext.dll* jest zbędna. Przy inicjalizacji środowiska RPC dla xHarbour (funkcja `THbApiInitialize(.T.)`) można podać alternatywne nazwy dla *harbour.dll* i *gtehrbext.dll*. Dopuszczalna jest tylko jedna biblioteka rozszerzeń zawierająca kod xHarbour.



III. Funkcje interfejsu aplikacji (TApi)

Poniżej w kolejności alfabetycznej opisano dostępne funkcje interfejsu aplikacji. Nazwy funkcji zaczynają się od przedrostka TApi... a funkcje znajdują się w bibliotece `gtrmapi.dll`. Dostępna jest biblioteka importów gtrmapi.lib. Prototypy funkcji oraz konieczne typy i stałe zdefiniowane są w pliku nagłówkowym gtrmapi.h. Wszystkie parametry tekstowe są przekazywane z/do funkcji interfejsu jako UNICODE.
Przed korzystaniem z funkcji interfejsu należy go zainicjalizować wywołaniem `TApiInitialize()`. Następujące funkcje stanowią wyjątek i mogą być wołane przed inicjalizacją interfejsu:

- ✓ `TApiInitialized()`
- ✓ `TApiTerminalMode()`
- ✓ `TApiGetApiVersion()`

Jeżeli któraś z funkcji interfejsu zwraca kod `TAPI_SYSERR` oznacza to, że podczas jej wykonywania wystąpił błąd systemu. W zależności od funkcji błąd mógł wystąpić na serwerze lub terminalu. Kod ten jest pobierany z systemu funkcją `GetLastError()` i może być odczytany przez użytkownika interfejsu przy pomocy jednej z trzech funkcji:

- ✓ `TApiGetLastError()`
- ✓ `TApiGetLastSrvError()`
- ✓ `TApiGetLastTrmError()`

1. TApiAsyncRPC

Składnia

```
int TApiAsyncRPC( HREMMODULE hRemoteModule,
                  WCHAR *pFunName,
                  void *pCallData,
                  int callDataSize );
```

Parametry

- ✓ `hRemoteModule` – uchwyt zdalnej biblioteki DLL uzyskany z funkcji `TApiRemoteLoadLibraryEx()`.
- ✓ `pFunName` – unikodowa nazwa funkcji z biblioteki DLL, która ma zostać wywołana.
- ✓ `pCallData` – wskaźnik na bufor zawierający dane, które zostaną przekazane do wołanej funkcji lub `NULL`.
- ✓ `callDataSize` – wielkość danych (w bajtach) wskazywanych przez `pCallData` lub `0`, jeżeli `pCallData == NULL`.

Wynik

Funkcja zwraca `TAPI_SUCCESS` w przypadku powodzenia lub jeden z kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia sieciowego z terminalem
- ✓ `TAPI_BADPARAMS` – błędne parametry wywołania

Opis

Funkcja umożliwia asynchroniczne wywołanie funkcji użytkownika rezydującej na terminalu i dołączonej do *gte.exe* w postaci biblioteki DLL. Przed wywołaniem biblioteka zawierająca wołaną funkcję musi zostać załadowana do przestrzeni adresowej *gte.exe* poprzez wywołanie funkcji `TApiRemoteLoadLibraryEx()`.
Asynchroniczne wywołanie polega na przekazaniu do *gte.exe* rozkazu wywołania funkcji wraz z parametrami i natychmiastowy powrót do aplikacji wywołującej bez oczekiwania na zakończenie wykonywania wołanej funkcji. Dlatego, wołając funkcję użytkownika przez `TApiAsyncRPC()` nie możemy

uzyskać informacji zwrotnej z wywoływanej funkcji. Wynik `TAPI_SUCCESS` oznacza jedynie, że polecenie wywołania funkcji zostało przekazane do *gte.exe*. Funkcje wołane za pośrednictwem `TApiAsyncRPC()` muszą być prawidłowo wyeksportowane z biblioteki DLL i zadeklarowane z typem wywołania cdecl. Typ zwracany oraz typy parametrów powinny być zgodne z typem ASYNCRPCFUN zdefiniowanym w gtrmapi.h. Nagłówek wołanej funkcji bez uwzględnienia dyrektywy eksportu oraz konwencji wywołania powinien wyglądać następująco:

```
void MyAsyncRPCFun( void *pCallData,
                    int callDataSize);
```

Przykładem wołanej asynchronicznie funkcji użytkownika jest funkcja `UserAsyncRPCBeepCallback()` z biblioteki *gteext.dll*. Przykład wywołania tej funkcji znajduje się w programie *testcapi.exe*.

2. TApiCheckConnected

Składnia

```
int TApiCheckConnected( void );
```

Wynik

Funkcja zwraca `1` jeżeli istnieje połączenie pomiędzy aplikacją i *gte.exe* lub `0` w przeciwnym przypadku.

Opis

Funkcja sprawdza czy istnieje połączenie między aplikacją (procesem) wołającą funkcję a odpowiadającym jej *gte.exe*

3. TApiGetApiVersion

Składnia

```
int TApiGetApiVersion( GTRMVERSION *pApiVer );
```

Parametry

- ✓ `pApiVer` – wskaźnik na strukturę `GTRMVERSION`, w której zostanie umieszczony wynik.

Wynik

Funkcja wypełnia strukturę `GTRMVERSION` i zwraca `0`.

Opis

Funkcja zwraca w strukturze `GTRMVERSION` informacje o wersji interfejsu czyli biblioteki *gtrmapi.dll*. Wersja ta powinna być zgodna na czterech pierwszych pozycjach z wersją wykorzystywanego Terminala. W przeciwnym przypadku inicjalizacja API przy pomocy `TApiInitialize()` nie powiedzie się.

4. TApiGetApiVersion

Składnia

```
int TApiGetApiVersion( GTRMVERSION *pApiVer );
```

Parametry

- ✓ `pApiVer` – wskaźnik na strukturę `GTRMVERSION`, w której zostanie umieszczony wynik.

Wynik

Funkcja wypełnia strukturę `GTRMVERSION` i zwraca `0`.

Opis

Funkcja zwraca w strukturze `GTRMVERSION` informacje o wersji interfejsu czyli biblioteki *gtrmapi.dll*. Wersja ta powinna być zgodna na czterech pierwszych pozycjach z wersją wykorzystywanego Terminala. W przeciwnym przypadku inicjalizacja API przy pomocy `TApiInitialize()` nie powiedzie się.

5. TApiGetClientDir

Składnia

```
wchar_t * TApiGetClientDir( int dirType );
```

Wynik

Funkcja zwraca wskaźnik na nazwę katalogu (ścieżkę) po stronie terminala lub `NULL` w przypadku błędu.

Opis

Funkcja umożliwia pobranie nazwy katalogu określonego parametrem *dirType* po stronie terminala. W chwili obecnej obsługiwane są następujące katalogi:

- `TAPI_DIRTYPE_GTE (1)` – katalog w którym znajduje się program *gte.exe*.

Zwracany wskaźnik wskazuje na dynamicznie zaalokowany blok pamięci który należy po wykorzystaniu zwolnić wywołując funkcję `TApiMemGlobalFree()`.

6. TApiGetExpirationDate

Składnia

```
int TApiGetExpirationDate( void );
```

Wynik

Funkcja zwraca datę ważności ewaluacyjnej wersji serwera Terminal lub `0` jeżeli jest to wersja produkcyjna. Bity 16-31 wyniku zawierają rok, bity 8-15 – miesiąc, bity 0-7 – dzień.

Opis

Funkcja umożliwia sprawdzenie czy aplikacja pracuje pod kontrolą ewaluacyjnej wersji serwera Terminal oraz sprawdzenie daty ważności wersji ewaluacyjnej. Zwrócona informacja może być wykorzystana np. w celu wyświetlenia przypomnienia o zbliżającym się końcu okresu ewaluacji i konieczności zakupu wersji produkcyjnej oprogramowania.

7. TApiGetFileFromTerminal

Składnia

```
int TApiGetFileFromTerminal(  WCHAR *pSrvFileName,
                              WCHAR *pTrmFileName,
                              REMFILEERROR *pFlError,
                              DWORD flags );
```

Parametry

- ✓ `pSrvFileName` – unikodowa nazwa pliku po stronie serwera, pod którą ma zostać zapisany pobierany z terminala plik.
- ✓ `pTrmFileName` – unikodowa nazwa pliku, który ma zostać pobrany z terminala.
- ✓ `pFlError` – wskaźnik do struktury `REMFILEERROR`, w której zapisane zostaną dodatkowe informacje o ewentualnym błędzie. Przekazanie `NULL` oznacza brak zainteresowania dodatkową diagnostyką. W polu `pFlError->error` zwracana jest wartość identyczna z wartością zwracaną przez funkcję. W polu `pFlError->nbytes` zwracana jest informacja o liczbie przesłanych bajtów pliku.
- ✓ `flags` – dodatkowe flagi dotyczące transferu pliku.
  Flaga `TAPI_F_FILEOVERWRITE` oznacza, że jeżeli plik docelowy istnieje, ma on zostać nadpisany. Brak flagi `TAPI_F_FILEOVERWRITE` oznacza, że istnienie pliku docelowego zostanie potraktowane jako błąd i transfer pliku nie zostanie zrealizowany.

Wynik

Funkcja zwraca `TAPI_SUCCESS` jeżeli transfer został wykonany poprawnie lub jeden z następujących kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia z terminalem (gte.exe)
- ✓ `TAPI_BADPARAMS` – podano błędne parametry, np. pominięto jedną z wymaganych nazw plików
- ✓ `TAPI_SYSERR` – wystąpił błąd systemowy. Pola `pFlError->lastSrvSysError` oraz `pFlError->lastTrmSysError` zawierają informacje o rodzaju

błędu zwróconą przez funkcję `GetLastError()` systemu Windows odpowiednio na serwerze i terminalu.

Opis

Funkcja umożliwia pobranie z terminala pliku o nazwie `pTrmFileName` i zapisanie go na serwerze pod nazwą `pSrvFileName`. Nazwa pliku na serwerze powinna być nazwą widoczną z systemu serwera, natomiast nazwa pliku na terminalu powinna być nazwą widzianą z systemu terminala.
W przypadku błędu `TAPI_SYSERR` można skorzystać z funkcji `GetLastError()`, `GetLastSrvError()` lub `GetLastTrmError()` dla odczytania dodatkowej informacji o błędzie.

8. TApiGetLastError

Składnia

```
DWORD TApiGetLastError( void );
```

Wynik

Funkcja zwraca kod ostatniego błędu systemowego który wystąpił na terminalu lub jeżeli brak błędu z terminala – kod ostatniego błędu z systemu serwera. `0` oznacza brak błędu.

Opis

Jeżeli w wyniku wywołania jednej z funkcji interfejsu otrzymaliśmy kod błędu `TAPI_SYSERR`, wówczas niniejsza funkcja zwróci systemowy kod błędu odczytany wcześniej przy pomocy `GetLastError()`. Funkcja zwróci kod błędu niezależnie od tego czy wystąpił on na terminalu czy na serwerze. W przypadku wystąpienia błędów na obu systemach zwrócony zostanie kod z terminala.

9. TApiGetLastSrvError

Składnia

```
DWORD TApiGetLastSrvError( void );
```

Wynik

Funkcja zwraca kod ostatniego błędu systemowego który wystąpił na serwerze lub `0`.

Opis

Jeżeli w wyniku wywołania jednej z funkcji interfejsu otrzymaliśmy kod błędu `TAPI_SYSERR` i błąd systemowy wystąpił na serwerze, wówczas niniejsza funkcja zwróci kod błędu odczytany wcześniej przy pomocy `GetLastError()`. W przypadku braku błędu na serwerze funkcja zwróci `0`.

10. TApiGetLastTrmError

Składnia

```
DWORD TApiGetLastTrmError( void );
```

Wynik

Funkcja zwraca kod ostatniego błędu systemowego który wystąpił na terminalu lub `0`.

Opis

Jeżeli w wyniku wywołania jednej z funkcji interfejsu otrzymaliśmy kod błędu `TAPI_SYSERR` i błąd systemowy wystąpił na terminalu, wówczas niniejsza funkcja zwróci kod błędu odczytany wcześniej przy pomocy `GetLastError()`. W przypadku braku błędu na terminalu funkcja zwróci `0`.

11. TApiGetRemoteIPAddr

Składnia

```
unsigned TApiGetRemoteIPAddr( void );
```

Wynik

Funkcja zwraca adres IP terminala lub routera implementującego NAT za którym znajduje się komputer terminala. Najstarsze osiem bitów wyniku zawiera najbardziej znaczący człon adresu IP i.t.d.

Opis

Funkcja umożliwia pobranie adresu IP terminala.

12. TApiGetRemoteIPPort

Składnia

```
unsigned TApiGetRemoteIPPort( void );
```

Wynik

Funkcja zwraca numer portu połączenia IP po stronie terminala.

Opis

Funkcja umożliwia pobranie numeru portu IP utworzonego przez *gte.exe* po stronie terminala dla komunikacji z aplikacją.

13. TApiGetSrvOSVer

Składnia

```
int TApiGetSrvOSVer( TAPI_OSVERSIONINFO *pOSVInfo );
```

Parametry

- ✓ `pOSVInfo` – wskaźnik na strukturę `TAPI_OSVERSIONINFO`, do której wpisywane są informacje dotyczące wersji systemu operacyjnego serwera uzyskane z funkcji `GetVersionEx()` systemu Windows.

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub `TAPI_NOTCONN` przy braku połączenia z terminalem.

Opis

Funkcja umożliwia odczytanie informacji identyfikujących system operacyjny serwera.

14. TApiGetTrmOSVer

Składnia

```
int TApiGetTrmOSVer( TAPI_OSVERSIONINFO *pOSVInfo );
```

Parametry

- ✓ `pOSVInfo` – wskaźnik na strukturę `TAPI_OSVERSIONINFO`, do której wpisywane są informacje dotyczące wersji systemu operacyjnego terminala uzyskane z funkcji `GetVersionEx()` systemu Windows.

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub `TAPI_NOTCONN` przy braku połączenia z terminalem.

Opis

Funkcja umożliwia odczytanie informacji identyfikujących system operacyjny terminala.

15. TApiGetUserName

Składnia

```
int TApiGetUserName( WCHAR *pUserName );
```

Parametry

- ✓ `pUserName` – wskaźnik na bufor mogący pomieścić `TAPI_MAXUSERNAME` znaków UNICODE (wielkości

`sizeof(wchar_t)`). W przypadku powodzenia, w buforze umieszczana jest zakończona zerem nazwa użytkownika serwera Terminal, który uruchomił aplikację.

Wynik

Funkcja zwraca `TAPI_SUCCESS` w przypadku powodzenia lub jeden z kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia sieciowego z terminalem,
- ✓ `TAPI_BADPARAMS` – błędne parametry wywołania, np. `pUserName == NULL`.

Opis

Funkcja umożliwia odczytanie nazwy użytkownika serwera Terminal, który uruchomił aplikację.

16. TApiGetTrmVersion

Składnia

```
int TApiGetTrmVersion( GTRMVERSION *pCallerVer,
                       GTRMVERSION *pCnetlibVer );
```

Parametry

- ✓ `pCallerVer` – zarezerwowany – należy zawsze przekazywać `NULL`.
- ✓ `pCnetlibVer` – wskaźnik na strukturę `GTRMVERSION`, w której zostanie umieszczony wynik.

Wynik

Funkcja zwraca `TAPI_SUCCESS`.

Opis

Funkcja umożliwia odczytanie wersji biblioteki *cnetlib.dll* z którą współpracuje aplikacja. Wersja ta jest dokładnie zgodna z wersją wykorzystywanego serwera Terminal.

17. TApiHwndToNetId

Składnia

```
int TApiHwndToNetId( HWND hWnd );
```

Parametry

- ✓ `hWnd` – uchwyt okna aplikacji.

Wynik

Sieciowy identyfikator okna lub `0`.

Opis

Funkcja umożliwia skonwertowanie uchwytu okna aplikacji na jego identyfikator sieciowy. Identyfikator sieciowy może zostać przekazany do funkcji wykonywanej na terminalu i tam skonwertowany na odpowiednie okno systemu przy pomocy funkcji `GteApiNetIdToHwnd()`. Jeżeli `hWnd` jest uchwytem okna nie mającego bezpośredniego odpowiednika po stronie terminala wówczas funkcja zwraca `0`.

18. TApiHwndToRemotedNetId

Składnia

```
int TApiHwndToRemotedNetId( HWND hWnd );
```

Parametry

- ✓ `hWnd` – uchwyt okna aplikacji.

Wynik

Sieciowy identyfikator okna lub jego rodzica lub `0`.

Opis

Funkcja umożliwia skonwertowanie uchwytu okna aplikacji na jego identyfikator sieciowy. Identyfikator sieciowy może zostać przekazany do funkcji wykonywanej na terminalu i tam skonwertowany na odpowiednie okno systemu przy pomocy funkcji `GteApiNetIdToHwnd()`. Uchwyty do okien

bezpośrednio odtwarzanych po stronie terminala (top level) są przekształcane na identyfikatory sieciowe tych okien. Uchwyty okien nie mających bezpośrednich odpowiedników po stronie terminala są przekształcane na identyfikatory okien top level których dieckiem jest okno `hWnd`.

19. TApiInitialize

Składnia

```
int TApiInitialize( void );
```

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub jeden z kodów błędów:

- ✓ `TAPI_NOCNETLIB` – nie znaleziono biblioteki *cnetlib.dll* – najprawdopodobniej proces aplikacji nie został uruchomiony w trybie terminalowym.
- ✓ `TAPI_BADAPIVERSION` – pierwsze cztery cyfry wersji interfejsu aplikacji (*gtrmapi.dll*) nie zgadzają się z pierwszymi czterema cyframi wersji oprogramowania Terminala (*cnetlib.dll*). Trzeba sprawdzić czy skopiowano właściwe biblioteki DLL.
- ✓ `TAPI_CANTIMPORTFUN` – nie jest możliwe pobranie adresu jednej z funkcji API wewnątrz *cnetlib.dll*.

Opis

Funkcja powoduje zainicjalizowanie wewnętrznych struktur interfejsu aplikacji zawartego w *gtrmapi.dll*. Podczas inicjalizacji następuje faktyczne połączenie funkcji *gtrmapi.dll* z funkcjami pomocniczymi zawartymi w *cnetlib.dll*. Interfejs powinien zostać zainicjalizowany przed pierwszym wywołaniem innych funkcji API. Wyjątek stanowią trzy funkcje, które można wołać przed zainicjalizowaniem API:

- ✓ `TApiInitialized()`
- ✓ `TApiTerminalMode()`
- ✓ `TApiGetApiVersion()`

20. TApiInitialized

Składnia

```
int TApiInitialized( void );
```

Wynik

Funkcja zwraca `1` jeżeli interfejs aplikacji został już prawidłowo zainicjalizowany przez wywołanie `TApiInitialize()` lub `0` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana do sprawdzenia w dowolnym miejscu aplikacji czy interfejs aplikacji został już zainicjalizowany i można korzystać z funkcji API.

21. TApiMemGlobalAlloc

Składnia

```
void *TApiMemGlobalAlloc( unsigned size );
```

Parametry

- ✓ `size` – rozmiar bloku pamięci, który ma zostać zaalokowany.

Wynik

Funkcja zwraca wskaźnik na zaalokowany blok pamięci o rozmiarze size lub `NULL` jeżeli pamięć nie może być przydzielona.

Opis

Funkcja umożliwia dynamiczne przydzielanie pamięci na potrzeby aplikacji. Przydzielony blok pamięci jest zainicjowany na zero. Pamięć przydzielona przy pomocy funkcji `TApiMemGlobalAlloc()` musi być zwalniana funkcją `TApiMemGlobalFree()`.

22. TApiMemGlobalFree

Składnia

```
void TApiMemGlobalFree( void *ptr );
```

Parametry

- ✓ `ptr` – wskaźnik na blok pamięci zaalokowany funkcją `TApiMemGlobalAlloc()` lub uzyskany jako rezultat z funkcji `TApiSyncRPC_VSR()`.

Opis

Funkcja zwalnia pamięć dynamiczne przydzieloną przez `TApiMemGlobalAlloc()`. Jeżeli funkcja `TApiSyncRPC_VSR()` zwróciła blok rezultatu o rozmiarze niezerowym, wówczas blok ten również musi zostać zwolniony przy pomocy niniejszej funkcji.

23. TApiNetIdToHwnd

Składnia

```
HWND TApiNetIdToHwnd( WNDNETID netid );
```

Parametry

- ✓ `netid` – sieciowy identyfikator okna uzyskany z funkcji `TApiHwndToNetId()`, `TApiHwndToRemotedNetId()` lub `GteApiHwndToNetId()`.

Wynik

Uchwyt okna Windows lub `NULL`.

Opis

Funkcja umożliwia skonwertowanie sieciowego identyfikatora okna na odpowiadające mu okno systemu Windows. Sieciowy identyfikator okna najczęściej będzie przekazywany do aplikacji z funkcji RPC bibliotek rozszerzających dołączonych do *gte.exe*. Funkcja RPC może skonwertować na terminalu faktyczny uchwyt okna systemu Windows na identyfikator sieciowy

przy pomocy `GteApiHwndToNetId()` i zwrócić go do aplikacji. Aplikacja może znaleźć okno odpowiadające na serwerze oknu terminala, przekształcając identyfikator sieciowy na okno serwera przy pomocy niniejszej funkcji. Jeżeli nie istnieje już okno Windows o podanym identyfikatorze sieciowym, wówczas funkcja zwraca `NULL`.

24. TApiPutFileToTerminal

Składnia

```
int TApiPutFileToTerminal( WCHAR *pSrvFileName,
                           WCHAR *pTrmFileName,
                           REMFILEERROR *pFlError,
                           DWORD flags );
```

Parametry

- ✓ `pSrvFileName` – unikodowa nazwa pliku po stronie serwera, który ma zostać przesłany na terminal.
- ✓ `pTrmFileName` – unikodowa nazwa pliku po stronie terminala, pod którą ma zostać zapisany przesyłany plik.
- ✓ `pFlError` – wskaźnik do struktury `REMFILEERROR` w której zapisane zostaną dodatkowe informacje o ewentualnym błędzie. Przekazanie `NULL` oznacza brak zainteresowania dodatkową diagnostyką. W polu `pFlError->error` zwracana jest wartość identyczna z wartością zwracaną przez funkcję. W polu `pFlError->nbytes` zwracana jest informacja o liczbie przesłanych bajtów pliku.
- ✓ `flags` – dodatkowe flagi dotyczące transferu pliku. Flaga `TAPI_F_FILEOVERWRITE` oznacza, że jeżeli plik docelowy istnieje, ma on zostać nadpisany. Brak flagi `TAPI_F_FILEOVERWRITE` oznacza, że istnienie pliku docelowego zostanie potraktowane jako błąd i transfer pliku nie zostanie zrealizowany.

Wynik

Funkcja zwraca `TAPI_SUCCESS` jeżeli transfer został wykonany poprawnie lub jeden z następujących kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia z terminalem (gte.exe)
- ✓ `TAPI_BADPARAMS` – podano błędne parametry, np. pominięto jedną z wymaganych nazw plików
- ✓ `TAPI_SYSERR` – wystąpił błąd systemowy. Pola `pFlError->lastSrvSysError` oraz `pFlError->lastTrmSysError` zawierają informacje o rodzaju błędu zwróconą przez funkcję `GetLastError()` systemu Windows odpowiednio na serwerze i terminalu.

Opis

Funkcja umożliwia przesłanie na terminal pliku o nazwie `pSrvFileName` i zapisanie go na terminalu pod nazwą `pTrmFileName`. Nazwa pliku na serwerze powinna być nazwą widoczną z systemu serwera, natomiast nazwa pliku na terminalu powinna być nazwą widzianą z systemu terminala. W przypadku błędu `TAPI_SYSERR` można skorzystać z funkcji `GetLastError()`, `GetLastSrvError()` lub `GetLastTrmError()` dla odczytania dodatkowej informacji o błędzie.

25. TApiRaiseFinalError

Składnia

```
int TApiRaiseFinalError( wchar_t *pDescription,
                         wchar_t *pFunction,
                         int ival1,
                         int ival2 );
```

Parametry

- ✓ `pDescription` – opis błędu jako string w unikodzie.
- ✓ `pFunction` – unikodowa nazwa funkcji w której wystąpił błąd.
- ✓ `ival1`, `ival2` – dodatkowe wartości diagnostyczne, które zostaną zapisane i wyświetlone wraz z błędem.

Wynik

Funkcja nigdy nie wraca, więc nie zwraca żadnych wartości.

Opis

Funkcja umożliwia zgłoszenie błędu terminalnego. W ramach jego obsługi wykonywane są następujące czynności:

- Komunikat oraz pozostałe dane błędu zapisywane są w logu po stronie serwera. Log znajduje się w podkatalogu *applogs* katalogu serwera Terminal i nazywa się *tapplog.txt*.
- Jeżeli istnieje połączenie pomiędzy *gte.exe* i aplikacją, to dane błędu wysyłane są do *gte.exe* i wyświetlane na terminalu w postaci komunikatu.
- Komunikat o błędzie wpisywany jest do logu po stronie terminala (*gtelog.txt*).
- Wymuszany jest koniec działania *gte.exe* oraz aplikacji z kodem większym od 0.

Jak wynika z powyższego opisu funkcja kończy działanie programu, więc sterowanie nie wraca do wywołującego i funkcja nie zwraca żadnych wartości.

26. TApiRemoteFreeLibrary

Składnia

```
BOOL TApiRemoteFreeLibrary( HREMMODULE hRemoteModule );
```

Parametry

- ✓ `hRemoteModule` – uchwyt zdalnej biblioteki rozszerzeń uzyskany przy pomocy funkcji `TApiRemoteLoadLibraryEx()`.

Wynik

Funkcja zwraca `1` jeżeli biblioteka została pomyślnie zwolniona, `0` w przeciwnym przypadku.

Opis

Funkcja umożliwia zwolnienie (usunięcie z przestrzeni adresowej *gte.exe*) biblioteki DLL załadowanej uprzednio przy pomocy funkcji `TApiRemoteLoadLibrary()`. Po zwolnieniu biblioteki nie jest możliwe wywoływanie zawartych w niej funkcji RPC. W przypadku błędu, dodatkową informację można uzyskać korzystając z funkcji `TApiGetLastTrmError()`.

27. TApiRemoteLoadLibraryEx

Składnia

```
HREMMODULE TApiRemoteLoadLibraryEx( WCHAR *pFileName,
                                    DWORD dwFlags );
```

Parametry

- ✓ `pFileName` – unikodowa nazwa zdalnej biblioteki rozszerzeń DLL tak jak jest ona widziana na terminalu przez proces *gte.exe*.
- ✓ `dwFlags` – parametr bezpośrednio przekazywany do funkcji Windows `LoadLibraryEx()` na terminalu. Najczęściej stosowaną wartością jest `0`.

Wynik

Funkcja zwraca uchwyt do zdalnie załadowanej biblioteki DLL lub `0` jeżeli wystąpił błąd.

Opis

Funkcja umożliwia załadowanie do przestrzeni adresowej procesu *gte.exe* wykonującego się na terminalu, biblioteki rozszerzeń o podanej nazwie. Biblioteki rozszerzeń zazwyczaj zawierają funkcje użytkownika, które mogą być zdalnie wywoływane przez aplikację przy pomocy jednej z funkcji RPC: `TApiAsyncRPC()`, `TApiSyncRPC()` i `TApiSyncRPC_VSR()`. Jeżeli załadowana biblioteka rozszerzeń nie będzie więcej potrzebna, to można ją zwolnić przy pomocy funkcji `TApiRemoteFreeLibrary()`. W przypadku błędu, dodatkową informację można uzyskać korzystając z funkcji `TApiGetLastTrmError()`.

28. TApiRemotePrintFile

Składnia

```
int TApiRemotePrintFile( WCHAR *pFileName,
                         WCHAR *pPrinterName,
                         WCHAR *pDatatype );
```

Parametry

- ✓ `pFileName` – unikodowa nazwa pliku zawierającego dane do wydrukowania.
- ✓ `pPrinterName` – unikodowa nazwa drukarki podłączonej do terminala na której ma być wykonany wydruk. Nazwa drukarki może zawierać postfix `@ERATERM`, który zostanie usunięty przed otwarciem drukarki na terminalu. Aby uzyskać wydruk na domyślnej drukarce terminala można jako nazwę drukarki przekazać `L"DEFPRN"`.
- ✓ `pDataType` – typ danych w pliku. W chwili obecnej wspierany jest jedynie format `L"TEXT"`.

Wynik

Funkcja zwraca `TAPI_SUCCESS` jeżeli drukowanie zostało wykonane poprawnie lub jeden z następujących kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia z terminalem (gte.exe)
- ✓ `TAPI_BADPARAMS` – podano błędne parametry
- ✓ `TAPI_SYSERR` – wystąpił błąd systemowy. Dodatkowe informacje dostępne za pośrednictwem funkcji `TApiGetLastSrvError()` i `TApiGetLastTrmError()`.

Opis

Funkcja umożliwia wydrukowanie zawartości podanego pliku na wskazanej drukarce dołączonej do terminala. Powrót z funkcji następuje po wysłaniu pliku na drukarkę. W chwili obecnej jedynym wspieranym formatem danych jest `L"TEXT"`.

29. TApiSendUpdates

Składnia

```
int TApiSendUpdates( HWND hWnd );
```

Parametry

- ✓ `hWnd` – uchwyt okna, którego obraz ma zostać zaktualizowany lub `NULL` dla aktualizacji wszystkich okien aplikacji.

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub `TAPI_NOTCONN` jeżeli brak jest połączenia z terminalem.

Opis

W celu optymalizacji transmisji sieciowej w środowisku Terminala GUI, zmiany nanoszone przez aplikację na okna Windows są wysyłane do terminala z pewnym opóźnieniem. Funkcja `TApiSendUpdates()` umożliwia wymuszenie natychmiastowego wysłania oczekujących zmian dla okna `hWnd` lub wszystkich okien aplikacji. Wymuszenie wysłania zmian może być przydatne na przykład przed wywołanie funkcji RPC która zakłada, że pewne informacje zostały już wyświetlone na ekranie terminala.

30. TApiSetDiscTmt

Składnia

```
DWORD TApiSetDiscTmt( DWORD timeout );
```

Parametry

- ✓ `timeout` – maksymalny czas w sekundach, po którym aplikacja rozłączy stację kliencką (terminal), która w wyniku uszkodzenia sieci lub wyłączenia utraciła łączność z aplikacją. Parametr może przybierać wartości z przedziału 20-10000. Wartość 0 oznacza powrót do wartości domyślnej, wartość 65535 całkowicie wyłącza mechanizm sprawdzania połączenia bazując jedynie na mechanizmach Windows.

Wynik

Funkcja zwraca poprzednio ustawioną wartość parametru lub `0` w przypadku nieprawidłowej wartości parametru timeout.

Opis

Funkcja umożliwia sterowanie maksymalnym czasem, po którym aplikacja rozłączy stację kliencką (terminal), która w wyniku uszkodzenia sieci lub wyłączenia utraciła łączność z aplikacją. Oczywiście rozłączenie może nastąpić

wcześniej, jeżeli mechanizmy protokołu TCP/IP zasygnalizują zerwanie połączenia.

31. TApiSyncRPC

Składnia

```
int TApiSyncRPC(  HREMMODULE hRemoteModule,
                  WCHAR *pFunName,
                  void *pCallData,
                  int callDataSize,
                  void *pResData,
                  int *pMaxResDataSize );
```

Parametry

- ✓ `hRemoteModule` – uchwyt zdalnej biblioteki DLL uzyskany z funkcji `TApiRemoteLoadLibraryEx()`.
- ✓ `pFunName` – unikodowa nazwa funkcji z biblioteki DLL, która ma zostać wywołana.
- ✓ `pCallData` – wskaźnik na bufor zawierający dane, które zostaną przekazane do wołanej funkcji lub `NULL`.
- ✓ `callDataSize` – wielkość danych (w bajtach) wskazywanych przez `pCallData` lub `0` jeżeli `pCallData == NULL`.
- ✓ `pResData` – wskaźnik na bufor, w którym zostanie umieszczony wynik wywołania funkcji lub `NULL` jeżeli rezultat nie jest oczekiwany.
- ✓ `pMaxResDataSize` – wskaźnik na zmienną typu `int`, w której wpisany jest rozmiar bufora wskazywanego przez `pResData`. Rozmiar ten określa maksymalną liczbę bajtów oczekiwanego wyniku. Jeżeli rezultat nie jest oczekiwany należy przekazać `NULL`. Po zakończeniu wywołania zmienna `*pMaxResDataSize` zawierała będzie faktyczną liczbę bajtów skopiowaną do bufora `pResData`.

Wynik

Funkcja zwraca `TAPI_SUCCESS` w przypadku powodzenia lub jeden z kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia sieciowego z terminalem.

- ✓ `TAPI_BADPARAMS` – błędne parametry wywołania.
- ✓ `TAPI_RESULTTOLARGE` – bufor na rezultat jest za mały.
- ✓ `TAPI_NOFUNCTION` – podana funkcja nie została znaleziona w module `hRemoteModule`.

Opis

Funkcja umożliwia synchroniczne wywołanie funkcji użytkownika rezydującej na terminalu i dołączonej do *gte.exe* w postaci biblioteki DLL. Przed wywołaniem biblioteka zawierająca wołaną funkcję musi zostać załadowana do przestrzeni adresowej gte.exe poprzez wywołanie funkcji `TApiRemoteLoadLibraryEx()`. Synchroniczne wywołanie funkcji polega na przekazaniu sterowania do funkcji RPC i odczekaniu na wynik jej wykonania. Wynik wykonania funkcji jest zwracany w buforze `pResData`, a jego rozmiar jest z góry ograniczony wielkością bufora. Funkcje wołane za pośrednictwem `TApiSyncRPC()` muszą być prawidłowo wyeksportowane z biblioteki DLL i zadeklarowane z typem wywołania `cdecl`. Typ zwracany oraz typy parametrów powinny być zgodne z typem `SYNCRPCFUN` zdefiniowanym w *gtrmapi.h*. Nagłówek wołanej funkcji bez uwzględnienia dyrektywy eksportu oraz konwencji wywołania powinien wyglądać następująco:

```
void MySyncRPCFun(  void *pCallData,
                    int callDataSize,
                    void *pResData,
                    int *pMaxResDataSize  );
```

Przed powrotem funkcja RPC powinna skopiować maksymalnie `*pMaxResDataSize` bajtów wyniku do bufora `pResData` i wstawić do zmiennej `*pMaxResDataSize` faktyczną liczbę bajtów skopiowanych do bufora. Zawartość bufora oraz informacja o jego wielkości zostanie przekazana z powrotem do funkcji wywołującej.
Przykładem wołanej synchronicznie funkcji użytkownika jest funkcja `UserSyncRPCMsgBoxCallback` z biblioteki *gteext.dll*. Przykład wywołania tej funkcji znajduje się w programie *testcapi.exe*.

32. TApiSyncRPC_VSR

Składnia

```
int TApiSyncRPC_VSR( HREMMODULE hRemoteModule,
                     WCHAR *pFunName,
                     void *pCallData,
                     int callDataSize,
                     void **ppResData,
                     int *pResDataSize );
```

Parametry

- ✓ `hRemoteModule` – uchwyt zdalnej biblioteki DLL uzyskany z funkcji `TApiRemoteLoadLibraryEx()`.
- ✓ `pFunName` – unikodowa nazwa funkcji z biblioteki DLL, która ma zostać wywołana.
- ✓ `pCallData` – wskaźnik na bufor zawierający dane, które zostaną przekazane do wołanej funkcji lub `NULL`.
- ✓ `callDataSize` – wielkość danych (w bajtach) wskazywanych przez `pCallData` lub `0` jeżeli `pCallData == NULL`.
- ✓ `ppResData` – wskaźnik na zmienną typu `void*`, w której zostanie umieszczony wskaźnik do bufora zawierającego wynik wykonania funkcji RPC lub `NULL` jeżeli funkcja nie zwróci wyniku.
- ✓ `pResDataSize` – wskaźnik na zmienną typu int zainicjowaną na `0`. Po powrocie w zmiennej tej będzie umieszczony rozmiar bufora rezultatu wskazywanego przez `*ppResData`.

Wynik

Funkcja zwraca `TAPI_SUCCESS` w przypadku powodzenia lub jeden z kodów błędu:

- ✓ `TAPI_NOTCONN` – brak połączenia sieciowego z terminalem.
- ✓ `TAPI_BADPARAMS` – błędne parametry wywołania.
- ✓ `TAPI_NOFUNCTION` – podana funkcja nie została znaleziona w module `hRemoteModule`.

Opis

Funkcja umożliwia synchroniczne wywołanie funkcji użytkownika rezydującej na terminalu i dołączonej do *gte.exe* w postaci biblioteki DLL. Przed wywołaniem biblioteka zawierająca wołaną funkcję musi zostać załadowana do przestrzeni adresowej *gte.exe* poprzez wywołanie funkcji `TApiRemoteLoadLibraryEx()`. Synchroniczne wywołanie funkcji polega na przekazaniu sterowania do funkcji RPC i odczekaniu na wynik jej wykonania. Wynik wykonania funkcji jest zwracany w buforze `*ppResData` zaalokowanym przez funkcję `TApiSyncRPC_VSR()`. Różnica pomiędzy funkcjami `TApiSyncRPC_VSR()` i `TApiSyncRPC()` polega na tym, że ta pierwsza umożliwia zwrócenie z RPC i przyjęcie przez wywołującego rezultatu o dowolnej wielkości, podczas gdy druga ma z góry ograniczoną wielkość wyniku. Tam gdzie z góry znamy wielkość wyniku wykonania funkcji RPC, należy korzystać z funkcji `TApiSyncRPC()`.

BARDZO WAŻNE!!!

Ponieważ bufor z rezultatem przydzielany jest dynamicznie przez funkcję `TApiSyncRPC_VSR()`, konieczne jest jego zwolnienie po wykorzystaniu. Aplikacja musi zwolnić bufor wyniku przy pomocy funkcji `TApiMemGlobalFree()`. Zastosowanie innej funkcji spowoduje błąd ochrony pamięci (GPF) lub inne problemy.
Funkcje wołane za pośrednictwem `TApiSyncRPC_VSR()` muszą być prawidłowo wyeksportowane z biblioteki DLL i zadeklarowane z typem wywołania `cdecl`. Typ zwracany oraz typy parametrów powinny być zgodne z typem `SYNCRPC_VSRFUN` zdefiniowanym w *gtrmapi.h*. Nagłówek wołanej funkcji bez uwzględnienia dyrektywy eksportu oraz konwencji wywołania powinien wyglądać następująco:

```
void MySyncRPCFun(  void *pCallData,
                    int callDataSize,
                    void **ppResData,
                    int *pResDataSize  );
```

Przed powrotem funkcja RPC wykonująca się na Terminalu (np. `MySyncRPCFun`) powinna zaalokować odpowiedni bufor przy pomocy funkcji `GteApiMemGlobalAlloc()` i skopiować do niego rezultat wykonania. Wskaźnik na bufor rezultatu powinien zostać podstawiony do zmiennej

`*ppResData`, a faktyczna wielkość wyniku wpisana do zmiennej `*pResDataSize`. Zawartość bufora oraz informacja o jego wielkości zostanie przekazana z powrotem do funkcji wywołującej. Zaalokowany w funkcji RPC bufor zostanie zwolniony przez kod *gte.exe*. Opisana sekwencja może wyglądać następująco:

```
// placing result in buffer pointed by *ppResData
*ppResData = GteApiMemGlobalAlloc(RESULT_SIZE);
if( *ppResData )
{
  *pResDataSize = RESULT_SIZE; // returning result size
}
else
{
  *pResDataSize = 0; // no result
}
```

Przykładem wołanej synchronicznie funkcji użytkownika ze zmiennym rozmiarem wyniku jest funkcja `UserSyncRPCRandomReplicateCallback` z biblioteki *gteext.dll*. Przykład wywołania tej funkcji znajduje się w programie *testcapi.exe*.

33. TApiTerminalMode

Składnia

```
int TApiTerminalMode( void );
```

Wynik

Funkcja zwraca `1` jeżeli aplikacja wykonuje się w trybie terminalowym lub `0` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana do sprawdzenia czy aplikacja wykonuje się w trybie terminalowym. Może być wywołana przed inicjalizacją interfejsu aplikacji funkcją `TApiInitialize()`. Umożliwia to łatwe pisanie aplikacji korzystających z rozszerzeń Terminala w trybie terminalowym lecz wykonujących się również w trybie nieterminalowym. `TApiTerminalMode()` będzie często pierwszą funkcją API wołaną przez

aplikację. Jeżeli aplikacja wykonuje się terminalowo, wówczas wywołaniem `TApiInitialize()` zostanie zainicjalizowany interfejs aplikacji.

IV. Funkcje interfejsu rozszerzeń gte.exe (GteApi)

Poniżej w kolejności alfabetycznej opisano dostępne funkcje interfejsu rozszerzeń *gte.exe*. Nazwy funkcji zaczynają się od przedrostka `GteApi`..., a funkcje znajdują się w bibliotece *gteapi.dll*. Dostępna jest biblioteka importów *gteapi.lib*. Prototypy funkcji oraz konieczne typy i stałe zdefiniowane są w plikach nagłówkowych *gteapi.h* i *gtrmapi.h*. Wszystkie parametry tekstowe są przekazywane z/do funkcji interfejsu jako UNICODE. Funkcje interfejsu rozszerzeń *gte.exe* są przeznaczone do wykorzystania z poziomu bibliotek DLL zawierających funkcje RPC i rozszerzających funkcjonalność standardowego *gte.exe*. Biblioteki rozszerzające mogą lecz nie muszą korzystać z funkcji interfejsu rozszerzeń. Przed korzystaniem z funkcji interfejsu należy go zainicjalizować wywołaniem funkcji `GteApiInitialize()`. Następujące funkcje stanowią wyjątek i mogą być wołane przed inicjalizacją interfejsu:

- ✓ `GteApiInitialized()`
- ✓ `GteApiGetApiVersion()`

34. GteApiCheckConnected

Składnia

```
int GteApiCheckConnected( void );
```

Wynik

Funkcja zwraca `1` jeżeli istnieje połączenie między gte.exe a aplikacją lub `0` w przeciwnym przypadku.

Opis

Funkcja sprawdza czy istnieje połączenie pomiędzy *gte.exe*, do którego jest dołączona wołająca biblioteka rozszerzeń DLL a aplikacją, która wywołała funkcję RPC.

35. GteApiGetApiVersion

Składnia

```
int GteApiGetApiVersion( GTRMVERSION *pApiVer );
```

Parametry

- ✓ `pApiVer` – wskaźnik na strukturę `GTRMVERSION`, w której zostanie umieszczony wynik.

Wynik

Funkcja wypełnia strukturę `GTRMVERSION` i zwraca `0`.

Opis

Funkcja zwraca w strukturze `GTRMVERSION` informacje o wersji interfejsu rozszerzeń czyli biblioteki *gteapi.dll*. Wersja ta powinna być zgodna na czterech pierwszych pozycjach z wersją wykorzystywanego Terminala (*gte.exe*). W przeciwnym przypadku inicjalizacja API przy pomocy `GteApiInitialize()` nie powiedzie się.

36. GteApiGetGteVersion

Składnia

```
int GteApiGetGteVersion( GTRMVERSION *pCallerVer,
                         GTRMVERSION *pGteVer );
```

Parametry

- ✓ `pCallerVer` – zarezerwowany – należy zawsze przekazywać `NULL`.
- ✓ `pGteVer` – wskaźnik na strukturę `GTRMVERSION`, w której zostanie umieszczony wynik

Wynik

Funkcja zwraca `TAPI_SUCCESS`.

Opis

Funkcja umożliwia odczytanie wersji programu *gte.exe* z którą współpracuje biblioteka rozszerzeń.

37. GteApiGetSrvInfo

Składnia

```
int GteApiGetSrvInfo( GTEAPI_TRMSVINFO *pSrvInfo );
```

Parametry

- ✓ `pSrvInfo` – wskaźnik na strukturę `GTEAPI_TRMSVINFO`, w której zostanie umieszczony wynik.

Wynik

Funkcja zwraca `TAPI_SUCCESS`.

Opis

Funkcja umożliwia odczytanie informacji o wersji systemu operacyjnego na serwerze aplikacji, wersji serwera Terminal oraz datę startu aplikacji.

38. GteApiHwndToNetId

Składnia

```
int GteApiHwndToNetId( HWND hWnd );
```

Parametry

- ✓ `hWnd` – uchwyt jednego z okien *gte.exe* odpowiadających oknom aplikacji.

Wynik

Sieciowy identyfikator okna lub `0`.

Opis

Funkcja umożliwia skonwertowanie uchwytu okna *gte.exe* odpowiadającego jednemu z głównych okien aplikacji na jego identyfikator sieciowy. Identyfikator sieciowy może zostać zwrócony z RPC do aplikacji i tam skonwertowany na odpowiednie okno systemu przy pomocy funkcji `TApiNetIdToHwnd()`. Jeżeli `hWnd` nie jest uchwytem okna utworzonego przez *gte.exe*, wówczas funkcja zwraca `0`.

39. GteApiInitialize

Składnia

```
int GteApiInitialize( void );
```

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub jeden z kodów błędów:

- ✓ `TAPI_NOGTEEXE` – nie znaleziono *gte.exe* – najprawdopodobniej biblioteka *gteapi.dll* została załadowana do jakiegoś innego procesu
- ✓ `TAPI_BADAPIVERSION` – pierwsze cztery cyfry wersji interfejsu rozszerzeń *gte.exe* (*gteapi.dll*) nie zgadzają się z pierwszymi czterema cyframi wersji *gte.exe*. Trzeba sprawdzić czy skopiowano właściwe biblioteki DLL
- ✓ `TAPI_CANTIMPORTFUN` – nie jest możliwe pobranie adresu jednej z funkcji API wewnątrz *gte.exe*.

Opis

Funkcja powoduje zainicjalizowanie wewnętrznych struktur interfejsu rozszerzeń *gte.exe* zawartego w *gteapi.dll*. Podczas inicjalizacji następuje faktyczne połączenie funkcji *gteapi.dll* z funkcjami pomocniczymi zawartymi w *gte.exe*. Interfejs powinien zostać zainicjalizowany przed pierwszym wywołaniem innych funkcji API. Wyjątek stanowią dwie funkcje, które można wołać przed zainicjalizowaniem API:

- ✓ `GteApiInitialized()`
- ✓ `GteApiGetApiVersion()`

40. GteApiInitialized

Składnia

```
int GteApiInitialized( void );
```

Wynik

Funkcja zwraca `1` jeżeli interfejs rozszerzeń gte.exe został już prawidłowo zainicjalizowany przez wywołanie `GteApiInitialize()` lub `0` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana w dowolnym miejscu funkcji RPC do sprawdzenia czy iterfejs rozszerzeń został już zainicjalizowany i można korzystać z funkcji API.

41. GteApiMemGlobalAlloc

Składnia

```
void *GteApiMemGlobalAlloc( unsigned size );
```

Parametry

- ✓ `size` – rozmiar bloku pamięci, który ma zostać zaalokowany.

Wynik

Funkcja zwraca wskaźnik na zaalokowany blok pamięci o rozmiarze `size` lub `NULL`, jeżeli pamięć nie może być przydzielona.

Opis

Funkcja umożliwia dynamiczne przydzielanie pamięci na potrzeby funkcji RPC. Przydzielony blok pamięci jest zainicjowany na zero. Pamięć przydzielona przy pomocy funkcji `GteApiMemGlobalAlloc()` musi być zwalniana funkcją `GteApiMemGlobalFree()`. Jeżeli funkcja RPC jest wołana poprzez `TApiSyncRPC_VSR()` i zwraca niezerowy rezultat, wówczas blok pamięci na wynik musi być przydzielony za pomocą niniejszej funkcji.

42. GteApiMemGlobalFree

Składnia

```
void GteApiMemGlobalFree( void *ptr );
```

Parametry

- ✓ `ptr` – wskaźnik na blok pamięci zaalokowany funkcją `GteApiMemGlobalAlloc()`.

Opis

Funkcja zwalnia pamięć dynamiczne przydzieloną przez `GteApiMemGlobalAlloc()`.

43. GteApiNetIdToHwnd

Składnia

```
HWND GteApiNetIdToHwnd( WNDNETID netid );
```

Parametry

- ✓ `netid` – sieciowy identyfikator okna uzyskany z funkcji `TApiHwndToNetId()`, `TApiHwndToRemotedNetId()` lub `GteApiHwndToNetId()`.

Wynik

Uchwyt okna Windows utworzonego przez *gte.exe* lub `NULL`.

Opis

Funkcja umożliwia skonwertowanie sieciowego identyfikatora okna na odpowiadające mu okno systemu Windows obsługiwane przez *gte.exe*. Sieciowy identyfikator okna najczęściej będzie przekazywany do funkcji RPC z aplikacji. Funkcja aplikacji może skonwertować na serwerze faktyczny uchwyt okna systemu Windows na identyfikator sieciowy przy pomocy `TApiHwndToNetId()` lub `TApiHwndToRemotedNetId()` i przekazać go do funkcji RPC. Funkcja RPC może znaleźć okno odpowiadające na terminalu oknu serwera przekształcając identyfikator sieciowy na okno serwera przy

pomocy niniejszej funkcji. Jeżeli nie istnieje już okno Windows o podanym identyfikatorze sieciowym, wówczas funkcja zwraca `NULL`.

44. GteApiRaiseFinalError

Składnia

```
int GteApiRaiseFinalError( wchar_t *pDescription,
                           wchar_t *pFunction,
                           int ival1,
                           int ival2 );
```

Parametry

- ✓ `pDescription` – opis błędu jako string w unikodzie.
- ✓ `pFunction` – unikodowa nazwa funkcji, w której wystąpił błąd.
- ✓ `ival1`, `ival2` – dodatkowe wartości diagnostyczne, które zostaną zapisane i wyświetlone wraz z błędem.

Wynik

Funkcja nigdy nie wraca, więc nie zwraca żadnych wartości

Opis

Funkcja umożliwia zgłoszenie błędu terminalnego. W ramach jego obsługi wykonywane są następujące czynności:

- Komunikat oraz pozostałe dane błędu zapisywane są w logu po stronie terminala. Log znajduje się w tym samym katalogu co *gte.exe* i nazywa się *gtelog.txt*.
- Jeżeli istnieje połączenie pomiędzy *gte.exe* i aplikacją, to do logu aplikacji (*gapplog.txt* w podkatalogu *applogs* serwera) wpisywany jest błąd `"GTE shutdown notification received - quiting"` i działanie aplikacji jest przerywane.
- Wymuszany jest koniec działania *gte.exe* z kodem większym od 0.

Jak wynika z powyższego opisu funkcja kończy działanie programu, więc sterowanie nie wraca do wywołującego i funkcja nie zwraca żadnych wartości.

45. GteApiSetConsoleEventMask

Składnia

```
unsigned GteApiSetConsoleEventMask( unsigned newMask );
```

Parametry

- ✓ `newMask` – nowa maska określająca sposób obsługi zdarzeń konsoli. Maska może zawierać bity `KEY_EVENT` i `MOUSE_EVENT` (*wincon.h*) w dowolnej kombinacji.

Wynik

Poprzednia wartość maski sterującej obsługą zdarzeń konsoli.

Opis

Zastosowanie funkcji ma sens jedynie w przypadku aplikacji wykonujących się w trybie konsoli Windows. Funkcja umożliwia selektywne blokowanie i odblokowywanie obsługi niektórych zdarzeń konsoli. Domyślnie konsola obsługuje zarówno zdarzenia klawiatury (`KEY_EVENT`) jak i zdarzenia myszki (`MOUSE_EVENT`). Przekazując odpowiednią maskę można określać czy obsługa danego typu zdarzeń ma być realizowana (bit ustawiony) czy blokowana (bit wyzerowany). Obsługa zdarzeń polega na ich przekazywaniu do aplikacji. Blokowanie zdarzeń polega na ich ignorowaniu (nie są przekazywanie do aplikacji).

V. Funkcje interfejsu aplikacji xHarbour (THbApi)

Poniżej opisano funkcje bezpośrednio dostępne dla aplikacji xHarbour. Nazwy funkcji zaczynają się od przedrostka `THbApi`... lub `Trm`..., a kod funkcji znajduje się w bibliotece *ghrbapi.lib*, którą należy dołączyć na etapie konsolidacji. W pierwszej kolejności opisane zostały nowe funkcje (`THbApi`), następnie funkcje `Trm`... Funkcje, których nazwy rozpoczynają się od `Trm`... są dołączone dla zapewnienia zgodności interfejsu z wcześniejszymi wersjami Terminala OTC przeznaczonego dla xHarbour/Clipper.
Przed korzystaniem z funkcji interfejsu należy go zainicjalizować wywołaniem `THbApiInitialize()`. Wyjątek stanowi funkcja `THbApiTerminalMode()`, z której można korzystać przed inicjalizacją interfejsu. Po zakończeniu korzystania z interfejsu należy wywołać funkcję `THbApiShutdown()`.

46. THbApiGetClientDir

Składnia

`THbApiGetClientDir( <nDirectoryType> ) -> cResult`

Wynik

Funkcja zwraca wskaźnik na nazwę (ścieżkę) katalogu po stronie terminala lub string pusty w przypadku błędu.

Opis

Funkcja umożliwia pobranie nazwy katalogu określonego parametrem `nDirectoryType` po stronie terminala. W chwili obecnej obsługiwane są następujące katalogi:

- `TAPI_DIRTYPE_GTE (1)` – katalog, w którym znajduje się program *gte.exe*.

47. THbApiInitialize

Składnia

```
THbApiInitialize( [<lInitRPC>] [,<cHarbourDllPath>
                  [,<cExtentionDllPath>] ] )-> nResult
```

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub jeden z kodów błędów:

- ✓ `TAPI_NOCNETLIB` – nie znaleziono biblioteki *cnetlib.dll* – najprawdopodobniej proces aplikacji nie został uruchomiony w trybie terminalowym.
- ✓ `TAPI_BADPARAMS` – podano błędne parametry.
- ✓ `TAPI_BADAPIVERSION` – pierwsze cztery cyfry wersji interfejsu aplikacji (*gtrmapi.dll*) nie zgadzają się z pierwszymi czterema cyframi wersji oprogramowania Terminala (*cnetlib.dll*). Trzeba sprawdzić czy skopiowano właściwe biblioteki DLL.
- ✓ `TAPI_CANTIMPORTFUN` – nie jest możliwe pobranie adresu jednej z funkcji API wewnątrz *cnetlib.dll*.
- ✓ `TAPI_CNTLOAD_GTEHRBDLL` – nie jest możliwe załadowanie *gtehrb.dll* po stronie terminala.
- ✓ `TAPI_CNTLOAD_HARBOURDLL` – nie jest możliwe załadowanie *harbour.dll* po stronie terminala.
- ✓ `TAPI_BADHARBOURDLLVER` – nie jest możliwe załadowanie *harbour.dll* po stronie terminala ze względu na niezgodność wersji xHarbour pomiędzy *harbour.dll* i *gtehrb.dll*.

Opis

Funkcja powoduje zainicjalizowanie wewnętrznych struktur interfejsu aplikacji xHarbour zawartego w *ghrbapi.lib*. Jeżeli parametr `lInitRPC` jest `.T.`, to następuje również inicjalizacja podsystemu RPC umożliwiającego wywoływanie zdalnych procedur xHarbour. Podczas inicjalizacji wykonywane są następujące czynności:

- Inicjalizacja bazowego interfejsu aplikacji (*gtrmapi.dll*) wywołaniem `TApiInitialize()`.

- Kolejne kroki są wykonywane jedynie wówczas jeżeli `lInitRPC` jest `.T.`
- Załadowanie zdalnej biblioteki rozszerzeń: *gtehrb.dll*.
- Załadowanie zdalnej biblioteki xHarbour: *harbour.dll* lub `cHarbourDllPath` jeżeli jest podane.
- Załadowanie zdalnej biblioteki rozszerzeń xHarbour: *gtehrbext.dll* lub `cExtentionDllPath` jeżeli jest podane.

Brak biblioteki rozszerzeń nie jest traktowane jako błąd. To, czy biblioteka rozszerzeń została załadowana, można sprawdzić funkcją `THbApiRPCExtInitialized()`.

48. THbApiInitialized

Składnia

```
THbApiInitialized( ) -> lResult
```

Wynik

Funkcja zwraca `.T.` jeżeli interfejs aplikacji xHarbour został już prawidłowo zainicjalizowany przez wywołanie `THbApiInitialize()` lub `.F.` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana w dowolnym miejscu aplikacji do sprawdzenia czy iterfejs aplikacji został już zainicjalizowany i można korzystać z funkcji API.

49. THbApiRPCInitialized

Składnia

```
THbApiRPCInitialized( ) -> lResult
```

Wynik

Funkcja zwraca `.T.` jeżeli interfejs aplikacji xHarbour został już prawidłowo zainicjalizowany z opcją wywołań RPC lub `.F.` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana w dowolnym miejscu aplikacji do sprawdzenia czy interfejs aplikacji został już zainicjalizowany i można korzystać z wywołań RPC. Jeżeli funkcja zwraca `.T.`, oznacza to, że interfejs jest zainicjalizowany i po stronie terminala załadowano *harbour.dll*.

50. THbApiRPCExtInitialized

Składnia

```
THbApiRPCExtInitialized( ) -> lResult
```

Wynik

Funkcja zwraca `.T.` jeżeli interfejs aplikacji xHarbour został już prawidłowo zainicjalizowany z opcją wywołań RPC i udało się załadować bibliotekę rozszerzającą *gtehrbext.dll* po stronie terminala. W przeciwnym przypadku funkcja zwraca `.F.`.

Opis

Funkcja może być wykorzystana w dowolnym miejscu aplikacji do sprawdzenia czy interfejs aplikacji został już zainicjalizowany i można korzystać z wywołań RPC i rozszerzeń użytkownika. Jeżeli funkcja zwraca `.T.`, oznacza to, że interfejs jest zainicjalizowany i po stronie terminala załadowano *harbour.dll* oraz bibliotekę rozszerzającą użytkownika *gtehrbext.dll*.

51. THbApiShutdown

Składnia

```
THbApiShutdown( ) -> nResult
```

Wynik

Funkcja zwraca `TAPI_SUCCESS` lub jeden z kodów błędów właściwych dla funkcji `TApiSyncRPC()`.

Opis

Funkcja umożliwia zwolnienie zasobów zajmowanych przez interfejs aplikacji xHarbour gdy nie jest on już potrzebny. W zależności od opcji inicjalizacji wykonywane są następujące czynności:

- Zwolnienie zdalnej biblioteki rozszerzeń xHarbour: *gtehrbext.dll*.
- Zwolnienie zdalnej biblioteki xHarbour: *harbour.dll*.
- Zwolnienie zdalnej biblioteki rozszerzeń: *gtehrb.dll*.

UWAGA!

Zachowana zostaje możliwość korzystania z interfejsu aplikacji na poziomie języka C/C++ przy pomocy funkcji `TApi`... z *gtrmapi.dll*.

52. THbApiTerminalMode

Składnia

```
THbApiTerminalMode( ) -> lResult
```

Wynik

Funkcja zwraca `.T.` jeżeli aplikacja wykonuje się w trybie terminalowym lub `.F.` w przeciwnym przypadku.

Opis

Funkcja może być wykorzystana do sprawdzenia czy aplikacja wykonuje się w trybie terminalowym. Może być wywołana przed inicjalizacją interfejsu aplikacji xHarbour funkcją `THbApiInitialize()`. Umożliwia to łatwe pisanie aplikacji korzystających z rozszerzeń Terminala w trybie terminalowym lecz wykonujących się również w trybie nieterminalowym. Funkcja `THbApiTerminalMode()` będzie często pierwszą funkcją API wołaną przez aplikację. Jeżeli aplikacja wykonuje się terminalowo, wówczas wywołaniem `THbApiInitialize()` zostanie zainicjalizowany interfejs aplikacji.

53. TrmAppOS

Dostępna wyłącznie w wersji 32-bitowej (Harbour/xHarbour).

Składnia:

`TrmAppOS() -> nAppSystemID`

Opis

Funkcja zwraca kod systemu operacyjnego, na którym wykonuje się aplikacja. Zwracane kody:

`2` – Windows NT
`6` – Windows 2000
`7` – Windows XP
`8` – Windows 2003
`32` – Linux

Patrz również: `TrmTeOS()`

54. TrmDiscTm

Składnia:

`TrmDiscTm( <nDisconnectTimeout> )`

Opis

Funkcja pozwala na określenie czasu, po którym aplikacja rozłączy stację kliencką (terminal), która w wyniku uszkodzenia sieci lub wyłączenia utraciła łączność z aplikacją. Parametr `NDisconnectTimeout` określa czas w sekundach i może przybierać wartości z przedziału 20-10000.
Funkcja z parametrem 0

`TrmDiscTm(0)`

przywraca domyślny czas rozłączania, natomiast

`TrmDiscTm(65535)`

całkowicie wyłącza mechanizm rozłączania terminala.

55. TrmFlPrint

Składnia:

`TrmFlPrint( <cFileToPrint>, <nLPTnumber> ) -> nResult`

Opis

Funkcja drukuje plik widziany przez aplikację xHarbour wykonującą się na serwerze jako `cFileToPrint` na drukarce dołączonej do końcówki terminala. Parametr `nLPTNumber` okresla numer portu LPT (1, 2, lub 3) na serwerze. W zależności od obowiązującego przekierowania plik zostanie wydrukowany na odpowiednim porcie LPT końcówki terminalowej.
Funkcja zwraca `0` w przypadku powodzenia. W przeciwnym przypadku funkcja zwraca kod błędu (<1000) identyczny z kodem zwracanym przez funkcje `FERROR()`. Jeżeli błąd wystąpił na terminalu podczas drukowania, zwracana jest liczba prawidłowo wydrukowanych znaków powiększona o 1000.

Przykład

```
Res = TrmFlPrint( ”myfile.txt”, 1 )
IF res == 0
   ? “Printed OK.”
ELSIF res < 1000
   ? “File access error (invalid file name?)”
ELSE
   ? “Error printing file, printed“, res-1000,“bytes”
ENDIF
```

56. TrmGetFile

Składnia:

```
TrmGetFile( <cTermFileName>, <cNTFileName> )-> nResult
```

Opis

Funkcja przesyła plik widziany przez aplikację *te.exe* wykonującą się na terminalu jako `cTermFileName` na serwer aplikacji i zapisuje go jako plik `cNTFileName`. Funkcja zwraca zero w przypadku powodzenia. W przeciwnym przypadku funkcja zwraca kod błędu identyczny z kodem zwracanym przez funkcje `FERROR()`. Jeżeli błąd wystąpił na terminalu, wówczas do kodu błędu dodana jest wartość 1000.

57. TrmGetPrty

Składnia:

```
TrmGetPrty() -> nCurrentPriority
```

Opis

Funkcja zwraca wartość określającą priorytet danego zadania na serwerze Windows. Możliwe wartości `nCurrentPriority`*:*

`0` - niski priorytet
`1` - normalny priorytet (priorytet domyślny)
`2` - wysoki priorytet
`3` - najwyższy priorytet

58. TrmIsTs

Składnia:

```
TrmIsTs() -> lResult
```

Opis

Mechanizm transakcji terminalowych nie jest zaimplementowany w Terminalu GUI. Funkcja została zachowana dla zapewnienia wstecznej kompatybilności istniejącego kodu. Funkcja zwraca .T. jeżeli ostatnio wołana była funkcja TrmTsBegin() lub .F. jeżeli ostatnio wołana była funkcja TrmTsEnd().

59. TrmPrCancl

Składnia:

```
TrmPrCancl( <nPrinterHandle> ) -> lResult
```

Opis

Funkcja przerywa drukowanie pliku wysłanego funkcją `TrmPrSubmt()` do drukarki identyfikowanej przez `nPrinterHandle`. Zwraca `.T.` jeżeli drukowanie zostało przerwane, `.F.` w przeciwnym przypadku.

60. TrmPrClose

Składnia:
`TrmPrClose( <nPrinterHandle> )`

Opis

Zamyka sesję drukarkową identyfikowaną przez `nPrinterHandle`. Zamknięcie sesji nie usuwa z kolejki wysłanych przedtem funkcją `TrmPrSubmt()` danych.

61. TrmPrFile

Składnia:
`TrmPrFile( <cPrinterName>, <cFileName> ) -> lResult`

Opis

Funkcja drukuje plik `cFileName` na drukarce `cPrinterName`. Nazwa drukarki powinna być identyczna z nazwą zwracaną przez funkcję `TrmPrList()`. Funkcja zwraca `.T.` jeżeli zadanie wykonano pomyślnie, w przeciwnym przypadku - `.F.` Funkcja nie wymaga otwartej sesji drukarkowej.

62. TrmPrList

Składnia:
`TrmPrList( ) -> aPrnTable`

Opis

Funkcja zwraca tablicę zawierającą nazwy i opisy wszystkich zdefiniowanych w systemie Windows NT/200x/XP/Vista drukarek lub `NIL` jeżeli takich drukarek nie ma. Zwracana tablica ma długość równą liczbie drukarek w systemie. Dla każdej drukarki zwracane są dwie wartości znakowe:

- ✓ nazwa - `aPrnTable[printerNumber][1]`.
- ✓ opis - `aPrnTable[printerNumber][2]`.

Nazwa drukarki może być przekazana do funkcji `TrmPrOpen()` lub `TrmPrFile()`.

Przykład

```
pTab = TrmPrList()
IF pTab == NIL
  ? ”No printer(s) defined”
  QUIT
ENDIF
FOR pn = 1 TO LEN(pTab)
  ? ”Printer name: ”, pTab[pn][1]
  ? ”Printer description: ”, pTab[pn][2]
NEXT
```

Przykładowy fragment kodu wypisuje nazwy i opisy wszystkich drukarek zdefiniowanych w systemie Windows NT.

63. TrmPrOpen

Składnia:

```
TrmPrOpen( <cPrinterName> ) -> nPrinterHandle
```

Opis

Funkcja otwiera sesję drukarkową związaną z drukarką `cPrinterName` i zwraca jej identyfikator lub `0` jeżeli otwarcie sesji nie powiodło się. Zwrócony identyfikator sesji należy przekazywać do wszystkich funkcji, które tego wymagają. Nazwa drukarki powinna być nazwą zwróconą przez funkcję `TrmPrList`.

64. TrmPrPut

Składnia:

```
TrmPrPut( <nPrinterHandle>,
          <cStringToPrint>,
          <nStringLength> ) -> lResult
```

Opis

Funkcja wysyła do pliku sesji `nPrinterHandle` `nStringLength` bajtów z łańcucha `cStringToPrint`. Zwraca `.T.` jeżeli wysłanie powiodło się, `.F.` w przeciwnym przypadku.

65. TrmPrPutFl

Składnia:
`TrmPrPutFl( <nPrinterHandle>, <cFileName> ) -> lResult`

Opis

Funkcja wysyła do pliku sesji `nPrinterHandle` zawartość pliku `cFileName`. Zwraca `.T.` jeżeli wysłanie powiodło się, `.F.` w przeciwnym przypadku.

66. TrmPrSubmt

Składnia:
`TrmPrSubmt( <nPrinterHandle> ) -> lResult`

Opis

Funkcja wysyła do kolejki drukarki związanej z sesją `nPrinterHandle` przygotowane wcześniej funkcjami `TrmPrPut()` i `TrmPrPutFl()` dane. Zwraca `.T.` jeżeli wysłanie powiodło się, `.F.` w przeciwnym przypadku.

67. TrmPutFile

Składnia:
`TrmPutFile( <cNTFileName>, <cTermFileName> ) -> nResult`

Opis

Funkcja przesyła plik widziany przez aplikację CA-Clipper wykonującą się na serwerze Windows NT/200x/XP/Vista jako `cNTFileName` na terminal i zapisuje go jako plik `cTermFileName`. Funkcja zwraca zero w przypadku

powodzenia. W przeciwnym przypadku funkcja zwraca kod błędu identyczny z kodem zwracanym przez funkcje `FERROR()`. Jeżeli błąd wystąpił na terminalu do kodu błędu dodana jest wartość 1000.

68. TrmSetPrty

Składnia:

```
TrmSetPrty( nNewPriority )
```

Opis

Funkcja umożliwia zmianę priorytetu wykonania aplikacji terminalowej.na serwerze Windows NT. Możliwe do ustawienia wartości `nNewPriority`*:*

- `0` - niski priorytet
- `1` - normalny priorytet (priorytet domyślny)
- `2` - wysoki priorytet
- `3` - najwyższy priorytet

W przypadku dużej liczby sesji terminalowych, funkcja `TrmSetPrty()` może służyć np. do zmniejszenia priorytetu aplikacji podczas wykonywania długich i obciążających serwer obliczeń (zestawienia i raporty). Zmniejszenie priorytetu takich fragmentów aplikacji umożliwia zachowanie sprawności pracy aplikacji interaktywnych nawet przy bardzo dużej liczbie sesji terminalowych.

69. TrmSvName

Składnia:

```
TrmSvName( ) -> nServerName
```

Opis

Funkcja zwraca nazwę serwera Windows NT na którym wykonuje się aplikacja lub NIL.

70. TrmTeOS

Składnia:

```
TrmTeOS() -> nTeSystemID
```

Opis

Funkcja zwraca kod systemu operacyjnego, na którym wykonuje się *gte.exe* (emulator terminala).
Zwracane kody:

1 - DOS
2 – Windows NT
3 – Windows 95
4 – Windows 98
5 – Windows Me
6 – Windows 2000
7 – Windows XP
8 – Windows 2003
32 - Linux

Patrz również: `TrmAppOS()`

71. TrmTrmRPC

Składnia:

```
TrmTrmRPC( <cTermFunName>, ... ) -> Result
```

Opis

Funkcja wywołana przez aplikację powoduje wykonanie na terminalu funkcji rozszerzeń `cTermFunName` zawartej w rozszerzającej bibliotece DLL. Do zdalnej funkcji przekazywane są parametry wyspecyfikowane po nazwie funkcji. Funkcja zwraca wartość zwróconą przez zdalnie wykonaną funkcję. Zdalne funkcje nie mogą otrzymywać ani zwracać argumentów typu tablicowego lub

przekazywanych przez referencję (@). Aby możliwe było wywołanie funkcji konieczne jest zainicjalizowanie interfejsu funkcją `THbApiInitialize()` w trybie z RPC czyli z parametrem `.T.`

Przykład:

Po stronie aplikacji:

```
? TrmTrmRPC( ”FUN1”, 1, 2 )
```

Po stronie terminala:

```
FUNCTION FUN1
PARAMETERS P1, P2
RETURN P1+P2
```

Wykonanie programu spowoduje wyświetlenie wartości 3, która zostanie obliczona na terminalu.

UWAGA!

Jeżeli funkcja `TrmTrmRPC()` zostanie wywołana po zerwaniu łączności terminala z aplikacją - zwrócona zostanie wartość `NIL`.

72. TrmTSBegin

Składnia:
`TrmTSBegin()`

Opis

Mechanizm transakcji terminalowych nie jest zaimplementowany w Terminalu GUI. Funkcja została zachowana dla zapewnienia wstecznej kompatybilności istniejącego kodu. Po wywołaniu TrmTsBegin() funkcja TrmIsTs() będzie zwracała .T.

73. TrmTSEnd

Składnia:
`TrmTSEnd()`

Opis

Mechanizm transakcji terminalowych nie jest zaimplementowany w Terminalu GUI. Funkcja została zachowana dla zapewnienia wstecznej kompatybilności istniejącego kodu. Po wywołaniu TrmTsEnd() funkcja TrmIsTs() będzie zwracała .F.

74. TrmUser

Składnia:

```
TrmUser() -> cTermUserName
```

Opis

Funkcja zwraca nazwę użytkownika terminalowego (podaną w momencie łączenia się z serwerem Terminal).

75. TrmUpdate

Składnia:

```
TrmUpdate()
```

Opis

Funkcja wymusza natychmiastowe wysłanie do terminala wszystkich zmian ekranowych buforowanych przez aplikację.

VI. Migracja aplikacji xHarbour do środowiska Terminala GUI

Jeżeli aplikacja pracująca ze starszą, specjalizowaną dla xHarbour/Clipper wersją Terminala OTC, ma nadal pracować tylko w trybie tekstowym, to nie jest konieczne wykonywanie migracji. Można ją uruchamiać przy pomocy *te32.exe* po wykonaniu standardowej procedury dostosowania do nowej wersji, polegającej na połączeniu z nowymi wersjami bibliotek Terminala.
Opisana w tym rozdziale migracja dotyczy takiej modyfikacji aplikacji, aby mogła ona pracować w środowisku GUI. Możliwe jest zrobienie pojedynczego pliku EXE, który będzie pracował zarówno w trybie terminalowym jak i samodzielnie.

Migracja *te32.exe*

Jeżeli migrowana aplikacja korzysta z *te32.exe* z dołączonymi rozszerzeniami użytkownika (funkcje RPC), to konieczne jest ich przeniesienie do biblioteki DLL, która może zostać załadowana przez gte.exe. Przykładem takiej biblioteki jest biblioteka *gtehrbext.dll*. Bibliotekę tę można łatwo zmodyfikować aby zawierała również konieczne do migracji funkcje. W tym celu należy:

1. Dodać do skryptu *mkhbextdll.bat* polecenia kompilacji wszystkich plików PRG zawierających nasze funkcje.
2. Na końcu pliku *gtehrbext.c* dołożyć polecenia dołączenia plików *.c powstałych podczas kompilacji naszych PRG.
3. Utworzyć bibliotekę DLL z naszymi funkcjami wykonując skrypt *mkhbextdll.bat*. Dla ułatwienia migracji aplikacji z rozszerzoną funkcjonalnością *te.exe* i *te32.exe*, do biblioteki rozszerzeń (*gtehrbext.dll*) można dołączyć bibliotekę *gtehrblib.lib* zawierającą funkcje xHarbour tradycyjnie dostępne w *te/te32.exe*, czyli `TrmCltVMaj()`, `TrmCltVMin()`, `TrmCltVSub()`, `TrmCltVBfx()`, `TrmSrvSys()`, `TrmSrvVMaj()`, `TrmSrvVMin()`, `TrmTSVMaj()`, `TrmTSVMin()`, `TrmTSVSub()`, `TrmTSVBfx()`, `TrmSrvYear()`, `TrmSrvMnth()`, `TrmSrvDay()`, `TrmRcvBts()`, `TrmPrList()`,

`TrmPrOpen()`, `TrmPrClose()`, `TrmPrPut()`, `TrmPrPutFl()`, `TrmPrFile()`, `TrmPrSubmt()`, `TrmPrCancl()`, `TrmSvName()`, `TrmTeOS()`, `TrmAppOS()`, `TrmRKBOff()`, `TrmRKBOn()`.

4. Powstałą bibliotekę skopiować do katalogu, w którym znajduje się *gte.exe*.

Jeżeli aplikacja korzysta ze standardowej wersji *te32.exe* wówczas powyższe czynności są zbędne.

Migracja aplikacji

Jeżeli aplikacja nie korzysta z funkcji pakietu Terminal (`Trm`...), to wystarczy usunąć z procesu łączenia aplikacji biblioteki związane z terminalem. Powstałą aplikację będzie można uruchamiać samodzielnie lub w trybie terminalowym za pomocą *gte.exe*.
Jeżeli aplikacja korzysta z funkcji pakietu Terminal, wówczas należy postępować następująco:

1. Usunąć z procesu łączenia aplikacji stare biblioteki Terminala
2. Dołączyć do procesu łączenia aplikacji biblioteki ghrbapi.lib oraz gtrmapi.lib
3. Na początku aplikacji (przed wywołaniem którejkolwiek funkcji terminala) umieścić wywołanie

   `THbApiInitialize(.T.)`

   (szczegóły w opisie funkcji `THbApiInitialize()`).

4. Przed zakończeniem aplikacji umieścić wywołanie

   `THbApiShutdown()`

   (szczegóły w opisie funkcji `THbApiShutdown()`).

5. Utworzyć plik EXE aplikacji.

Tak utworzoną aplikację będzie można uruchamiać w trybie terminalowym przy pomocy *gte.exe*. Jeżeli aplikacja ma być również wykorzystywana samodzielnie, warto popracować nad utworzeniem pojedynczego EXE pracującego samodzielnie lub terminalowo. W tym celu należy skorzystać z funkcji `THbApiTerminalMode()` umożliwiającej sprawdzenie czy aplikacja wykonuje się w trybie terminalowym. Oczywiście, wszelkie kolejne odwołania do funkcji Terminala powinny być realizowane jedynie podczas pracy w trybie terminalowym.