



Entire Connection

Programmierschnittstelle (API)

Version 4.3.1



Dieses Handbuch gilt für Entire Connection ab Version 4.3.1.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

© März 2002, Software AG
Alle Rechte vorbehalten

Software AG und/oder Software AG Produkte sind entweder Warenzeichen oder eingetragene Warenzeichen der Software AG. Andere hier erwähnte Produkte und Unternehmensnamen können Warenzeichen ihrer jeweiligen Eigentümer sein.

Inhaltsverzeichnis

Programmierschnittstelle (API)	1
Glossar	1
Allgemeine Informationen	2
Synchrone und asynchrone Aufrufe	3
Übersicht der API-Aufrufe	4
Initialisierung	5
Session öffnen	7
Allgemeine Steuerung	8
Bildschirmdaten	10
Datentransfer	13
Tasks und Prozedurdateien	17
Session beenden	20
Andere Methoden	20
Andere Ereignisse	21
Tasten-Codes	21
Fehler-Codes	21

Programmierschnittstelle (API)

Mit der Programmierschnittstelle (Application Programming Interface - API) können Sie Funktionen von Entire Connection direkt aus einem Programm heraus aufrufen. Ein ActiveX-Control stellt eine gemeinsame Schnittstelle für die Entwicklung mit Visual Basic oder C++ zur Verfügung.

Dieser Abschnitt behandelt die folgenden Themen:

- Glossar
- Allgemeine Informationen
- Synchrone und asynchrone Aufrufe
- Übersicht der API-Aufrufe
- Andere Ereignisse
- Tasten-Codes
- Fehler-Codes

Es wird vorausgesetzt, dass Sie bereits mit ActiveX-Controls (entweder mit Visual Basic oder C++) und Entire Connection vertraut sind.

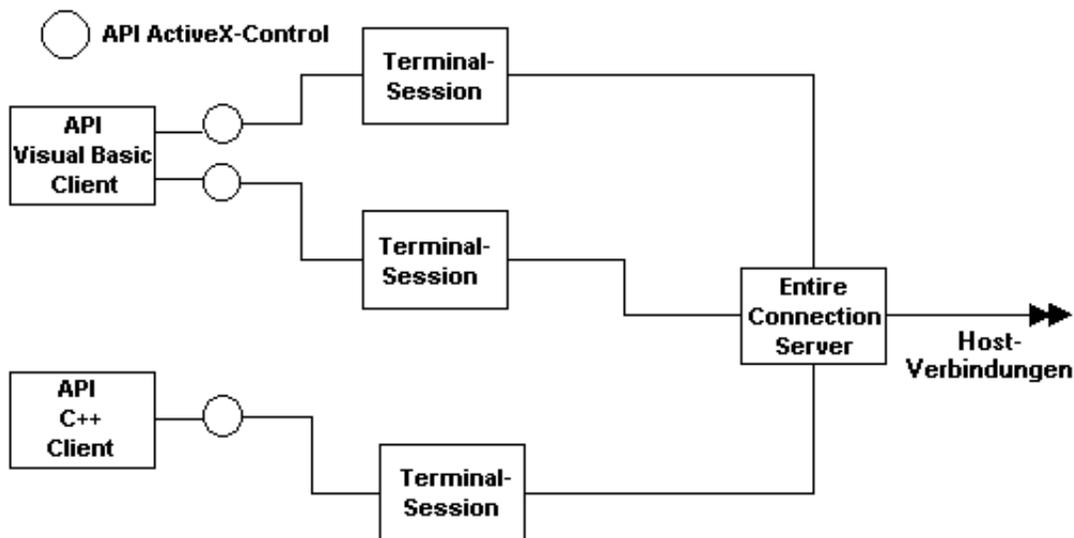
Diese Beschreibung sollte zusammen mit dem Beispielcode gelesen werden, der sich auf der CD-ROM befindet.

Glossar

API	Funktionalität, die Drittanbieteranwendungen zur Verfügung steht.
API-Client	Die Anwendung, die Entire Connection mit Hilfe der Programmierschnittstelle kontrolliert.
API-Control	Das vom API-Client verwendete ActiveX.
Terminal-Session	Die Terminal-Anwendung von Entire Connection.
Asynchron	Nicht-blockierender Modus. Die Programmierschnittstelle kehrt sofort zu der aufrufenden Anwendung zurück. Wenn die Verarbeitung abgeschlossen ist, sendet die Programmierschnittstelle eine Nachricht an die Anwendung.
Synchron	Die Programmierschnittstelle kehrt nur dann zu der aufrufenden Anwendung zurück, wenn die Verarbeitung des Funktionsaufrufs abgeschlossen ist.

Allgemeine Informationen

Jedes ActiveX kann ein einzelnes Terminal von Entire Connection unterstützen. Deshalb kann eine einzelne Anwendung mit beliebig vielen Terminal-Sessions arbeiten, wobei viele Controls gleichzeitig aktiv sind.



Jedes API-Control kann die Verbindung zu einer bestehenden Terminal-Session herstellen oder eine neue Terminal-Session erstellen. Jede Terminal-Session kann jederzeit mit einem API-Control verbunden sein. Die einzige Ausnahme ist der UA-Modus, bei dem das Verbinden nicht erlaubt ist. Außerdem ist es nicht möglich, den UA-Modus bei einem Terminal einzuschalten, das durch die Programmierschnittstelle kontrolliert wird.

Im API-Modus ist eine Terminal-Session in der Regel unsichtbar, um Benutzereingaben zu verhindern. Wenn das Terminal durch die Programmierschnittstelle sichtbar gemacht wird, hat der Benutzer die volle Kontrolle über das Terminal. Unter anderem kann er dann Prozedurdateien ausführen und die Terminal-Session beenden. Jeder Datentransfer und jede Prozedurdatei bleibt unter der Kontrolle des API-Client.

Synchrone und asynchrone Aufrufe

Synchrone (blockierende) und asynchrone (nicht-blockierende) Aufrufe stehen sowohl mit Visual Basic als auch mit C++ zur Verfügung. Während der Planungsphase entscheiden Sie, welche dieser beiden Aufrufarten für Ihre Zwecke am Besten geeignet ist.

Im asynchronen Modus kehren fast alle API-Aufrufe sofort mit einem entsprechenden Return-Code zurück. Ausnahmen sind die Initialisierungsfunktionen und die Funktionen zum Beenden der Verbindung mit einer Terminal-Session. Diese Funktionen blockieren immer, unabhängig vom gewählten Modus.

Nachdem ein Befehl bei der asynchronen API-Ausführung abgearbeitet wurde, sendet das Control ein Ereignis, das bestätigt, dass der Befehl abgearbeitet wurde. Die Parameter dieses Ereignisses enthalten das Ergebnis des Aufrufs (d.h. eine Rückmeldung wie der Befehl abgearbeitet wurde) und alle angeforderten Daten.

Die Beschreibungen in der nachstehenden Übersicht der API-Aufrufe informieren Sie darüber, ob ein Aufruf nur synchron ausgeführt werden kann. In allen anderen Fällen wird ein Ereignis gesendet, das bestätigt, dass der Befehl abgearbeitet wurde; `LogonEntireConnection` sendet zum Beispiel `LogonComplete`.

In bestimmten Situationen erzeugt das API-Control auch Benachrichtigungsereignisse ohne Berücksichtigung des Modus, in dem es gerade ausgeführt wird. Dies können Fehlermeldungen, Informationen und Datentransferdaten sein.

Übersicht der API-Aufrufe

Dieser Abschnitt enthält einen Überblick über alle verfügbaren API-Aufrufe. Sie sind in die folgenden funktionalen Bereiche aufgeteilt:

- **Initialisierung**
 - `GetRunningTerminalSessions`
 - `Initialize`
 - `LogonEntireConnection`
- **Session öffnen**
 - `GetAvailableSessions`
 - `OpenSession`
- **Allgemeine Steuerung**
 - `RunHostCommand`
 - `PutData`
 - `SetDataNotificationFlag`
- **Bildschirmdaten**
 - `GetScreenText`
 - `GetScreenRawText`
 - `GetScreenAttributes`
 - `GetExtendedAttributes`
 - `GetCursorPosition`
 - `SetCursorPosition`
 - `ClearScreenText`
 - `CheckForScreenText`
- **Datentransfer**
 - `SetAPIFileDetails`
 - `SetWorkFileDetails`
 - `GetFileName`
 - `CancelFileTransfer`
- **Tasks und Prozedurdateien**
 - `RunEntConTask`
 - `SetGlobalParameter`
 - `GetGlobalParameter`
 - `CancelRunningTask`
- **Session beenden**
 - `CloseSession`
 - `CloseAllSessions`
 - `BreakConnection`
- **Andere Methoden**
 - `GetScreenSize`

Diese API-Aufrufe (und alle damit verbundenen Ereignisse) sind nachstehend ausführlich beschrieben.

Initialisierung

Beim Starten einer Session kann der API-Client entweder die Verbindung zu einem aktiven Terminal herstellen oder ein neues Terminal erstellen.

▶ Session-Namen aller aktiven Terminals abfragen (nur synchroner Aufruf)

- Rufen Sie Folgendes auf:

```
APIReturn = GetRunningTerminalSessions(TerminalNames, NumTerminals)
```

Dies gibt alle zur Zeit aktiven Terminals, zu denen eine Verbindung hergestellt werden kann, in einem Array aus. Der Aufruf der Funktion `GetRunningTerminalSessions` ist der einzige Aufruf, der vor dem Aufruf von `Initialize` erfolgen kann.

▶ Verbindung zu einem Terminal herstellen

- Rufen Sie Folgendes auf:

```
APIReturn = Initialize(CreateSession, LinkSessionName,
UserLoggedIn, OpenSession)
```

Die Parameter sind:

<code>CreateSession</code>	Boolean "true" bedeutet, dass ein neues Terminal erstellt werden soll.
<code>LinkSessionName</code>	String Name eines vorhandenen Terminals, zu dem eine Verbindung hergestellt werden soll. Der Name ist einer der Terminal-Namen, der von der Funktion <code>GetRunningTerminalSessions</code> geliefert wird.
<code>UserLoggedIn</code>	Boolean Gibt "true" zurück, wenn die Anmeldung bei Entire Connection auf der Workstation bereits erfolgt ist. Damit ein Terminal benutzt werden kann, muss sich der Benutzer pro Arbeitsplatz einmal anmelden. Wenn <code>UserLoggedIn</code> "false" ist, muss sich der API-Client jetzt anmelden.
<code>OpenSession</code>	String Normalerweise leer. In einer besonderen Situation wird der Name einer geöffneten Session eingetragen.

Wenn für `CreateSession` "true" übergeben wird oder keine Verbindung zum angegebenen Terminal hergestellt werden kann, erstellt das API-Control ein neues Terminal.

Wenn die Verbindung zu einem vorhandenen Terminal hergestellt wurde und in der Zwischenzeit eine Session in diesem Terminal geöffnet wurde, enthält der Parameter `OpenSession` den Namen der Session. Der API-Client muss in dieser speziellen Situation entscheiden, ob er mit dieser Session, die nicht unter seiner Kontrolle geöffnet wurde, arbeiten will. Diese Situation kann nur entstehen, wenn zum Zeitpunkt der Abfrage der vorhandenen Terminals ein Terminal gerade im Begriff war, eine Session zu öffnen, der Vorgang jedoch länger dauerte und noch nicht abgeschlossen war.

 **Bei Entire Connection anmelden**

- Rufen Sie Folgendes auf:
`APIReturn = LogonEntireConnection(UserName, Password)`

Session öffnen

Der API-Client kann entweder die zur Verfügung stehenden Session-Namen aus der Share-Datei abfragen oder eine namentlich bekannte Session direkt öffnen.

▶ Alle Sessions abfragen, die für einen Benutzer von Entire Connection definiert sind

- Rufen Sie Folgendes auf:

```
APIReturn = GetAvailableSessions(SessionNames, DefaultSession)
```

Die Parameter sind:

```
SessionNames    Variant Array(Strings)
                  Die Namen aller definierten Sessions.
```

```
DefaultSession  String
                  Der Name der Standard-Session.
```

▶ Eine dieser Sessions öffnen

- Rufen Sie Folgendes auf:

```
APIReturn = OpenSession(SessionName)
```

Der Parameter ist:

```
SessionName     String
                  Der Name der zu öffnenden Session.
```

Die Session ist nun offen und kann benutzt werden.

Hiermit verbundene Ereignisse:

- `FirstScreenArrived`
Wird gesendet, wenn die Session die ersten Daten vom Host erhält.
- `ScreenSizeChanged(NumRow, NumColumns)`
Gibt die anfängliche Bildschirmgröße an und auch, ob sich das Terminal während der Session dynamisch verändert.
- `SessionOpened(SessionName)`
Wird gesendet, wenn eine Session geöffnet wird ohne dass der API-Client die Methode `OpenSession` aufruft. Dies kann zum Beispiel durch einen Start-Task geschehen.

Der Parameter ist:

```
SessionName     String
                  Der Name der geöffneten Session.
```

Allgemeine Steuerung

▶ Befehle an die geöffnete Session senden

- Rufen Sie Folgendes auf:
`APIReturn = RunHostCommand(CommandName)`

Der Parameter ist:

`CommandName` String
 Der Name des Befehls, der auf dem Host aufgerufen werden soll.

Der String wird an den Host gesendet und anschließend an die Funktionstaste ENTER.

▶ Beliebigen Text und Tasten-Codes senden

- Rufen Sie Folgendes auf:
`APIReturn = PutData(Text , KeyCode)`

Die Parameter sind:

`Text` String
 Text, der an den Host übertragen werden soll.

`KeyCode` Integer
 Taste, die nach der Übertragung des Textes gesendet werden soll.

Der Text, der mit diesem Befehl gesendet wird, kann Zeilenvorschübe enthalten. Diese werden wie das Drücken der Funktionstaste NEWLINE interpretiert. Wenn Sie nur einen Tasten-Code senden wollen, müssen Sie als Text eine leere Zeichenkette übergeben.

▶ Datenbenachrichtigungen erlauben (nur synchroner Aufruf)

- Rufen Sie Folgendes auf:
`APIReturn = SetDataNotificationFlag(Enable)`

Der Parameter ist:

`Enable` Boolean
 Wenn man dies auf "true" setzt, werden Datenbenachrichtigungen eingeschaltet.
 Vorgabe: ausgeschaltet.

▶ Anzeige des Terminal-Fensters ein- und ausschalten

- Setzen Sie die API-Control-Eigenschaft `TerminalInteractive` (boolean).
 Wenn Sie die Verbindung zu einem Terminal herstellen, bleibt es solange sichtbar bis dieser Wert auf "false" gesetzt wird.
 Wenn Sie ein neues Terminal erstellen, ist es solange unsichtbar bis dieser Wert auf "true" gesetzt wird.

Hiermit verbundene Ereignisse:

- `CursorPositionChanged(XPosition, YPosition)`
Wird gesendet, wenn sich das Terminal im interaktiven Modus befindet und die Cursor-Position mit der Maus verändert wird (nicht wenn sich der Cursor durch Tippen bewegt).
- `NewScreenDataArrived()`
Wenn eingeschaltet bedeutet dies, dass neue Daten vom Host angekommen sind.

Bildschirmdaten

Bildschirmtext steht als Rohtext zur Verfügung wie er vom Host empfangen wird und als verarbeiteter Text wie er im Terminal angezeigt wird. Der Rohtext enthält alle Zeichen - auch die, die nicht angezeigt werden sollen (z.B. Passwort) - und kann auch Nullwerte enthalten.

Da der Rohtext Nullwerte enthalten kann, kann er nur in ein Array mit vorzeichenlosen Zeichen ausgegeben werden. Der Bildschirmtext wird in ein Array mit Zeichenketten ausgegeben.

► Bildschirmtext abfragen

- Rufen Sie Folgendes auf:
`APIReturn = GetScreenText(ScreenTextArray, TopLeftX, TopLeftY, BottomRightX, BottomRightY)`

Die Parameter sind:

<code>ScreenTextArray</code>	Variant Array(Strings) Eine Zeichenkette pro angeforderter Textzeile.
<code>TopLeftX</code>	Integer Startkoordinate.
<code>TopLeftY</code>	Integer Startkoordinate.
<code>BottomRightX</code>	Integer Endkoordinate.
<code>BottomRightY</code>	Integer Endkoordinate.

Wenn eine dieser Koordinaten auf -1 gesetzt wird, wird der gesamte Bildschirm ausgegeben.

► Rohdaten abfragen

- Rufen Sie Folgendes auf:
`APIReturn = GetScreenRawText(ScreenTextArray)`

Der Parameter ist:

<code>ScreenTextArray</code>	Variant Array(Unsigned chars) Rohdaten-Buffer.
------------------------------	---

▶ **Bildschirmattribute abfragen**

- Rufen Sie Folgendes auf:

```
APIReturn = GetScreenAttributes(Attributes, AttributesDescription)
```

Die Parameter sind:

`Attributes` Variant Array(Unsigned chars)
Attribut-Buffer.

`AttributesDescription` Variant Array(Unsigned chars)
Die Beschreibung eines Attributs ist ein Array mit 6 Werten, das die Bit-Muster für die Attributeigenschaften enthält:

Member 0: Attribut

Member 1: Geschützt

Member 2: Numerisch

Member 3: Keine Anzeige

Member 4: Erhöhte Helligkeit

Member 5: Feldinhalt wurde geändert

▶ **Erweiterte Bildschirmattribute abfragen**

- Rufen Sie Folgendes auf:

```
APIReturn = GetExtendedAttributes(ExtendedAttributes)
```

Der Parameter ist:

`ExtendedAttributes` Variant Array(Unsigned chars)
Buffer mit erweiterten Attributen.

▶ **Aktuelle Cursor-Position ermitteln und verändern**

- Rufen Sie Folgendes auf:

```
APIReturn = GetCursorPosition(XPosition, YPosition)
```

```
APIReturn = SetCursorPosition(XPosition, YPosition)
```

Die Parameter sind:

`XPosition` Integer
X steht für die Spalte der Cursor-Position.

`YPosition` Integer
Y steht für die Zeile der Cursor-Position.

▶ **Editierbaren Text im angegebenen Bereich entfernen**

- Rufen Sie Folgendes auf:
`APIReturn = ClearScreenText(TopLeftX, TopLeftY, BottomRightX, BottomRightY)`

Die Parameter sind:

TopLeftX Integer
 Startkoordinate.

TopLeftY Integer
 Startkoordinate.

BottomRightX Integer
 Endkoordinate.

BottomRightY Integer
 Endkoordinate.

-1 in einem dieser Werte heißt: der gesamte Bildschirm.

▶ **IF-Befehl zum Suchen von Bildschirmtext aufrufen**

- Rufen Sie Folgendes auf:
`APIReturn = CheckForScreenText(Text, Result, Position, TopLeftX, TopLeftY, Length, CaseSensitive)`

Die Parameter sind:

Text String
 Zu suchender Text.

Result Boolean
 "true" wenn der Text gefunden wurde.

Position Integer
 Bildschirmposition, in der der Text gefunden wurde.

TopLeftX Integer
 Startkoordinate.

TopLeftY Integer
 Startkoordinate.

Length Integer
 Textlänge.

CaseSensitive Boolean
 "true" wenn Überprüfung auf Groß-/Kleinschreibung.

Datentransfer

▶ Datentransfer vorbereiten für den direkten Transfer mit dem API-Client

- Rufen Sie Folgendes auf:
`APIReturn = SetAPIFileDetails(WorkFileNumber, UploadFlag, BinaryFlag, ReportFlag)`

Die Parameter sind:

<code>WorkFileNumber</code>	Integer Nummer des Work File.
<code>UploadFlag</code>	Boolean Wird zum Hochladen gesetzt.
<code>BinaryFlag</code>	Boolean Wird zum Übertragen von Binärdateien gesetzt.
<code>ReportFlag</code>	Boolean Wird für das Report-Format gesetzt.

Die folgenden Ereignisse werden beim Hochladen gesendet:

```
GetAsciiUploadFileBuffer(ErrorCode, FileNumber, Data, DataLength,
DataFormat)
GetBinaryUploadFileBuffer(ErrorCode, WorkFileNumber, Data,
DataLength)
```

Die folgenden Ereignisse werden beim Herunterladen gesendet:

```
AsciiFileDataArrived(ErrorCode, FileNumber, DataLength, Data,
DataFormat)
BinaryFileDataArrived(ErrorCode, FileNumber, DataLength, Data,
DataFormat)
```

Die Ereignisparameter sind:

<code>ErrorCode</code>	Integer Muss von der Programmierschnittstelle auf 0 gesetzt werden, um daraufhinzuweisen, dass der Vorgang ohne Fehler bearbeitet wurde.
<code>FileNumber</code>	Integer Das zu verarbeitende Work File.
<code>DataLength</code>	Integer Hochladen: übergeben wird die erwartete Größe; zurückgegeben wird die tatsächliche Größe. Herunterladen: wird auf die Größe der übertragenen Daten gesetzt.
<code>Data</code>	Variant Array(unsigned char) Die zu übertragenden Daten.
<code>DataFormat</code>	String Beschreibung des Datensatzformats.

Bei einem normalen Datentransfer muss der API-Client einen Dateinamen angeben. Dieser Name kann vordefiniert werden.

 **Dateiname vordefinieren**

- Rufen Sie Folgendes auf:

```
APIReturn = SetWorkFileDetails(Name, FileNumber, Upload, Binary, Report)
```

Die Parameter sind:

Name	String	Zu verwendender Dateiname.
FileNumber	Integer	Das verwendete Work File.
Upload	Boolean	Wird zum Hochladen gesetzt.
Binary	Boolean	Wird zum Übertragen von Binärdateien gesetzt.
Report	Boolean	Wird für das Report-Format gesetzt.

Wenn keine vordefinierten Werte für das verwendete Work File gefunden werden, wird der API-Client nach einem Dateinamen gefragt.

▶ **Dateiname übergeben**

- Reagieren Sie auf Folgendes Ereignis:
`APIReturn = GetFileName(ErrorCode, FileNumber, Upload, Binary, ToPrinter, Landscape, ControlChars, DosFormat, FileName)`

Die Parameter sind:

<code>ErrorCode</code>	Integer Bei Null wird der Dateiname benutzt und die Verarbeitung beginnt. Bei einem anderen Wert wird die Verarbeitung abgebrochen.
<code>FileNumber</code>	Integer Das verwendete Work File.
<code>Upload</code>	Boolean Wird zum Hochladen eines Dateinamens gesetzt.
<code>Binary</code>	Boolean Wird zum Übertragen von Binärdateien gesetzt.
<code>ToPrinter</code>	Boolean Wird zum Herunterladen auf einen Drucker gesetzt.
<code>Landscape</code>	Boolean Wird zum Drucken im Querformat gesetzt.
<code>ControlChars</code>	Boolean Wird zum Interpretieren der Steuerzeichen gesetzt.
<code>FileName</code>	String Zu verwendender Dateiname.

▶ **Laufenden Datentransfer abbrechen**

- Rufen Sie Folgendes auf:
`APIReturn = CancelFileTransfer(FileNumber)`

Der Parameter ist:

<code>FileNumber</code>	Integer Die Nummer des Work File, für das der Datentransfer abgebrochen wird.
-------------------------	--

Dieser Aufruf ist synchron. Er stellt eine Anfrage zum Abbrechen in eine Warteschlange. Wenn der Datentransfer abgeschlossen ist, wird das Ereignis `FileTransferComplete` gesendet.

Hiermit verbundene Ereignisse:

- `FileTransferStarting(ErrorCode, FileNumber, Upload, Binary, Headings)`

Die Parameter sind:

<code>ErrorCode</code>	<code>Integer</code> Bei Null wird der Dateiname benutzt und die Verarbeitung beginnt. Bei einem anderen Wert wird die Verarbeitung abgebrochen.
<code>FileNumber</code>	<code>Integer</code> Das verwendete Work File.
<code>Upload</code>	<code>Boolean</code> Wird zum Hochladen eines Dateinamens gesetzt.
<code>Binary</code>	<code>Boolean</code> Wird zum Übertragen von Binärdateien gesetzt.
<code>Headings</code>	<code>Variant Array (Strings)</code> Enthält die Feldnamen des Datensatzes für den Datentransfer.

- `FileTransferComplete(FileNumber, Upload, ErrorCode)`

Die Parameter sind:

<code>FileNumber</code>	<code>Integer</code> Das verwendete Work File.
<code>Upload</code>	<code>Boolean</code> Wird gesetzt, wenn das Hochladen beendet ist.
<code>ErrorCode</code>	<code>Integer</code> Wird auf Null gesetzt, wenn der Vorgang ohne Fehler bearbeitet wurde.

- `FileTransferProgress(ProgressMessage)`

Der Parameter ist:

<code>ProgressMessage</code>	<code>String</code> Nachricht, die normalerweise im Ausgabefenster der Terminal-Anwendung angezeigt wird.
------------------------------	--

Tasks und Prozedurdateien

▶ Task oder Prozedurdatei ausführen

- Rufen Sie Folgendes auf:
`APIReturn = RunEntConTask(TaskName)`

Der Parameter ist:

`TaskName` String
Der Name eines unter Entire Connection definierten Task oder einer Prozedurdatei.

Anmerkung:

Bei einer synchronen Verbindung kehrt die Programmierschnittstelle erst dann zur aufrufenden Anwendung zurück, nachdem der `TaskName` geprüft und der Task oder die Prozedurdatei gestartet wurde (nicht erst wenn der Task oder die Prozedurdatei beendet wird). Bei einem asynchronen Aufruf kehrt die Programmierschnittstelle sofort zur aufrufenden Anwendung zurück.

▶ Auf die globalen Parameter +PARM0 bis +PARM9 zugreifen

- Rufen Sie Folgendes auf:
`APIReturn = SetGlobalParameter(ParamNumber, Value)`
`APIReturn = GetGlobalParameter(ParamNumber, Value)`

Die Parameter sind:

`ParamNumber` Integer
Zwischen 0 und 9 für den erforderlichen Parameter.

`Value` String
Wert des Parameters.

▶ Prozedurdatei abbrechen (nur synchroner Aufruf)

- Rufen Sie Folgendes auf:
`APIReturn = CancelRunningTask()`

Die Programmierschnittstelle kehrt sofort zur aufrufenden Anwendung zurück. Die Prozedurdatei sendet beim Abbruch das Ereignis `EntConTaskComplete`.

Hiermit verbundene Ereignisse:

- `EntConTaskStarting(ErrorCode, TaskName)`
Wird aufgerufen, wenn ein Task gestartet wird, der nicht explizit von der Programmierschnittstelle aufgerufen wurde (z.B. ein Task zum Anmelden).

Die Parameter sind:

`ErrorCode` Integer
Muss auf 0 (Null) gesetzt werden, damit der Task starten kann.

`TaskName` String
Name des gestarteten Task.

- `EntConTaskComplete(ErrorCode, TaskName)`

Die Parameter sind:

`ErrorCode` Integer
Ist auf Null gesetzt, wenn der Task fehlerfrei ausgeführt wurde.

`TaskName` String
Name des Task.

- `TaskInputRequest(ErrorCode, DisplayOne DisplayTwo, Flags, ReturnData)`
Dieses Ereignis wird gesendet, wenn der Befehl INPUT in einer Prozedurdatei ausgeführt wird.

Die Parameter sind:

`ErrorCode` Integer
Wird auf Null gesetzt, wenn die Eingabe erfolgt ist.

`DisplayOne` String
Erste Zeile der Eingabeaufforderung.

`DisplayTwo` String
Zweite Zeile der Eingabeaufforderung.

`Flags` Variant Array
Flags für die Anzeige (siehe unten).

`Flags(0)` Muss Daten zurückgeben; leer ist nicht erlaubt.

`Flags(1)` Nur numerische Daten.

`Flags(2)` Passwortfeld.

`Flags(3)` Maximale Länge der angeforderten Daten.

`ReturnData` String
Daten, die an die Prozedurdatei übergeben werden sollen.

- `TaskDisplayMessageRequest(ErrorCode, Text, DialogBox, MessageType, Response)`

Dieses Ereignis wird gesendet, wenn der Befehl WAIT in einer Prozedurdatei ausgeführt wird.

Die Parameter sind:

<code>ErrorCode</code>	<code>Integer</code> Wird auf 0 (Null) gesetzt, wenn die Prozedur weiterlaufen soll. Wenn Null nicht gesetzt ist, wird die Prozedur abgebrochen.
<code>Text</code>	<code>String</code> Anzuzeigende Nachricht.
<code>DialogBox</code>	<code>Boolean</code> "true", wenn ein Dialogfeld mit einer Nachricht erwartet wird.
<code>MessageType</code>	<code>Variant</code> Anzeigeparameter.
<code>Response</code>	<code>Integer</code> Standard-Microsoft-Antwort-Code der MessageBox (z.B. "IDOK"), wenn DialogBox "true" ist.

- `TaskError(ErrorCode, ErrorText)`

Die Parameter sind:

<code>ErrorCode</code>	<code>Integer</code> Vom Task ausgegebener Fehler-Code.
<code>ErrorText</code>	<code>String</code> Anzuzeigende Nachricht.

Session beenden

▶ Geöffnete Session beenden, aber die Verbindung mit Entire Connection bestehen lassen

- Rufen Sie Folgendes auf:
`APIReturn = CloseSession()`

▶ Alle Terminals schließen (nur asynchroner Aufruf)

- Rufen Sie Folgendes auf:
`APIReturn = CloseAllSessions()`

Hiermit wird jede Terminal-Session beendet, einschließlich der Terminal-Sessions, die direkt geöffnet wurden. Dieser Aufruf sollte mit Vorsicht benutzt werden. Er bricht auch die Verbindung mit dem Terminal ab. Es wird kein Ereignis gesendet, das bestätigt, dass der Befehl abgearbeitet wurde.

▶ Verbindung mit dem Terminal beenden (nur synchroner Aufruf)

- Rufen Sie Folgendes auf:
`APIReturn = BreakConnection(Closedown)`

Der Parameter ist:

`Closedown` Boolean
 Ist auf "true" gesetzt, um das Terminal-Fenster beim Abbrechen der Verbindung zu schließen.

Wenn `Closedown` auf "false" gesetzt wird und das Terminal nicht angemeldet ist, wird das Terminal trotzdem geschlossen. Wenn das Terminal unsichtbar war, wird es beim Abbrechen der Verbindung automatisch angezeigt.

Hiermit verbundene Ereignisse:

- `CurrentSessionClosed`
 Die Session wurde beendet, aber nicht durch einen API-Aufruf. Dies kann passieren, wenn das Terminal interaktiv benutzt wird und der Benutzer die Session schließt oder wenn eine Zeitüberschreitung auftritt.
- `TerminalClosedown`
 Das Terminal wurde geschlossen, aber nicht durch einen API-Aufruf. Dies kann im interaktiven Modus passieren, wenn der Benutzer die Anwendung schließt oder wenn `CloseAllSessions` von einer anderen API-Session aufgerufen wird.

Andere Methoden

▶ Aktuelle Größe des geöffneten Terminals ermitteln

- Rufen Sie Folgendes auf:
`APIReturn = GetScreenSize(NumberOfRows, NumberOfColumns)`

Andere Ereignisse

- `ServerRequestedFileName(ErrorCode, OpenFile, Flags, Title, DefExtension, Filter, InitFileName, InitDirectory, FileName)`
Wird aufgerufen, wenn die Session einen Dateinamen benötigt.

Die Parameter sind:

`ErrorCode` Integer
Wird auf Null gesetzt, wenn der Dateiname angegeben wurde.

`FileName` String
Zu verwendender Dateiname.

Die anderen Parameter sind die, die im Standarddateiauswahldialog angegeben werden können.

- `TerminalWarningMessage(Message, DisplayFlag)`

Die Parameter sind:

`Message` String
Anzuzeigende Nachricht.

`DisplayFlag` Boolean
Es wird erwartet, dass der Aufruf eine Nachricht in einem blockierenden Dialogfeld anzeigt (z.B. mit der Funktion `MessageBox`).

Tasten-Codes

Die Tasten-Codes, die mit `PutData` übergeben werden können, sind in der Include-Datei enthalten, die als Bestandteil der Beispiele auf der Entire Connection CD-ROM enthalten ist. Nur diese Tasten-Codes sollten verwendet werden. Wenn andere Werte übergeben werden, kann dies unvorhergesehene Folgen haben.

Fehler-Codes

Dies sind Integer-Werte. Die Beschreibungen befinden sich in den Include-Dateien, die als Bestandteile der Beispiele auf der Entire Connection CD-ROM enthalten sind.