Entire Connection

Application Programming Interface

Version 9.3.3

October 2025

ADABAS & NATURAL

This document applies to Entire Connection Version 9.3.3 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PCC-API-933-20251001

Table of Contents

Preface

Using the application programming interface (API), you can invoke Entire Connection functions directly from a program. An ActiveX control provides a common interface for development with Visual Basic .NET, C++ and C#.

This section provides the following information:

General Information

Overview of API Calls

Other Events, Key Codes and Return/Error Codes

It is assumed that you are familiar with ActiveX controls (with Visual Basic .NET, C++ or C#) and Entire Connection.

This description should be read in conjunction with the sample code which is provided on the Entire Connection installation medium. The sample code can be found in the *Windows\API* folder of the installation medium.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://techcommunity.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

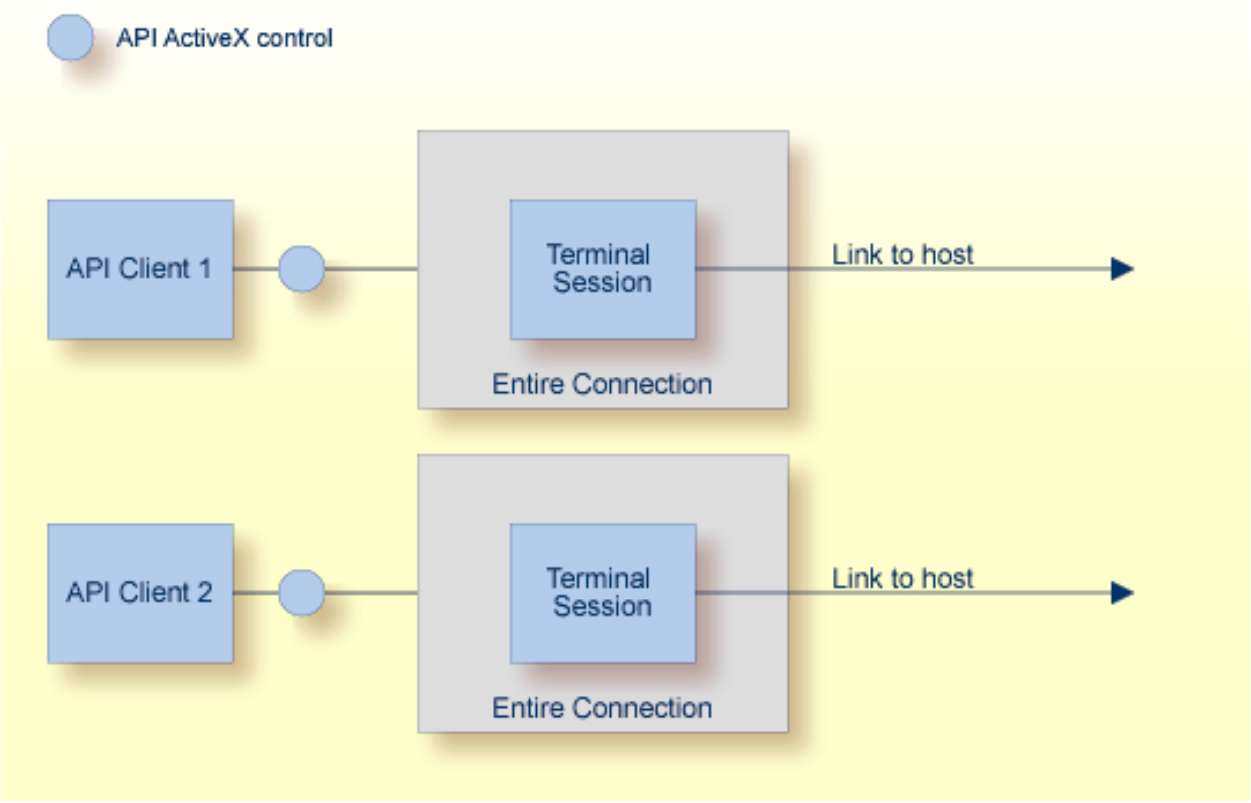
Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 General Information

API Controls and Terminal Sessions

Each API control can link to an existing terminal session or create a new terminal session. Each terminal session can have one API control attached at any one time, the only exception being a terminal running in unattended mode when attaching is not allowed. It is also impossible to set an API-controlled terminal to unattended mode.



When a terminal session is in API mode, it is usually hidden to prevent user input. If the API makes the terminal visible, the user has full control of the terminal, including executing procedure files and closing down the terminal session. All data transfer operations and procedure files will still remain under the control of the API client.

Synchronous and Asynchronous Calls

Synchronous (blocking) and asynchronous (non-blocking) calls are available in Visual Basic .NET, C++ and C#. At design time, you decide which of these two modes is appropriate.

If the control is set to asynchronous mode, nearly all API calls will return immediately with an appropriate return code. The main exceptions to this are the functions used for initialization and closing down a terminal session. These functions will always block regardless of the mode selected.

When the API is running asynchronously and a command completes, the control will fire a completion event. The parameters for this event contain the completion code from the call and any data requested.

The descriptions in the ***Overview of API Calls*** indicate when a call is only available synchronously. In all other cases, a completion event will be fired, for example `LogonEntireConnection` will fire `LogonComplete`.

In certain situations, the API control will also fire notification events regardless of the mode it is running in. These can include error messages, information messages and all data transfer data.

Glossary

API	Functionality available to third-party applications.
API Client	The application controlling Entire Connection using the application programming interface.
API Control	The ActiveX used by the API client.
Terminal Session	The terminal application of Entire Connection.
Asynchronous	Non-blocking mode. The application programming interface immediately returns to the calling application. When processing has completed, the application programming interface sends a message to the application.
Synchronous	Blocking mode. The application programming interface only returns to the calling application when processing of the API call has completed.

3 Overview of API Calls

This section provides an overview of all available API calls, grouped according to the following functional areas:

- **Initialization**
  - GetRunningTerminalSessions
  - Initialize
  - LogonEntireConnection
- **Opening a Session**
  - GetAvailableSessions
  - OpenSession
- **General Control**
  - RunHostCommand
  - PutData
  - SetDataNotificationFlag
- **Screen Data**
  - GetScreenText
  - GetScreenRawText
  - GetScreenAttributes
  - GetExtendedAttributes
  - GetCursorPosition
  - SetCursorPosition
  - ClearScreenText
  - CheckForScreenText
- **Data Transfer**
  - SetAPIFileDetails
  - SetWorkFileDetails
  - GetFileName
  - CancelFileTransfer
- **Tasks and Procedure Files**
  - RunEntConTask
  - SetGlobalParameter
  - GetGlobalParameter
  - CancelRunningTask
- **Closing a Session**

- CloseSession
- CloseAllSessions
- BreakConnection

- **Other Methods**
  - GetScreenSize

See the descriptions below for detailed information on these API calls (including associated events).

Initialization

When starting a session, the API client can either attach to a running terminal or create a new terminal.

To find out the session names of any running terminals (synchronous call only)

- Call the following:

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

This returns an array of currently running terminals that can be attached. GetRunningTerminalSessions is the only call that can be made before calling Initialize.

To attach to a terminal

- Call the following:

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

The parameters are:

CreateSession	Boolean. "true" indicates that a new terminal is to be created.
LinkSessionName	String. The name of an existing terminal to attach to. The name is one of the terminal names that is returned by the GetRunningTerminalSessions function.
UserLoggedOn	Boolean. Returns "true" if the logon to Entire Connection has already taken place on the workstation. In order to use a terminal, a user has to log on once per workstation. If UserLoggedOn is "false", the API client has to log on now.
OpenSession	String. Normally empty. In a special case, this contains the name of an open session.

If "true" was returned for CreateSession or if it is not possible to attach to the specified terminal, the API control creates a new session.

If the connection to an existing terminal has been established and if in the meantime a session has been opened in this terminal, the `OpenSession` parameter contains the name of the session. In this special case, the API client has to decide whether it wants to work with this session which has not been opened under its control. This can only happen if an existing terminal is attached that is currently in the process of opening a session, and this process takes a while and has not yet been completed.

To log on to Entire Connection

- Call the following:

```
APIReturn = LogonEntireConnection(UserName, Password)
```

Opening a Session

The API client can either query the available session names from the share file or open a known session directly.

To query all sessions defined for the Entire Connection user

- Call the following:

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

The parameters are:

`SessionNames`	Variant Array(Strings). The names of all defined sessions.
`DefaultSession`	String. The name of the default session.

To open one of these sessions

- Call the following:

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

The parameter is:

SessionName	String. The name of the session that is to be opened.

The session is now open and can be used.

Associated Events:

- FirstScreenArrived

  Fired when the session receives the first data from the host.

- ScreenSizeChanged(NumRow, NumColumns)

  Notifies the initial screen size, and also whether the terminal changes dynamically during a session.

- SessionOpened(SessionName)

  Fired if a session opens without the API client calling the OpenSession method. This may happen, for example, when a startup task is used. The parameter is:

SessionName	String. The name of the open session.

General Control

To send commands to the open session

- Call the following:

```
APIReturn = RunHostCommand(CommandName)
```

  The parameter is:

CommandName	String. The name of the command that is to be executed on the host.

  The string is sent to the host and then to the function key ENTER.

To send general text and key codes

- Call the following:

```
APIReturn = PutData(Text, KeyCode)
```

The parameters are:

`Text`	String. The text that is to be transferred to the host.
`KeyCode`	Integer. The key that is to be sent after the text has been transferred.

The text that is sent with this command can contain line feeds. These are interpreted as if the function key NEWLINE has been pressed. If you only want to send a key code, you have to pass an empty string for the text.

To enable data notifications (synchronous call only)

- Call the following:

```
APIReturn = SetDataNotificationFlag(Enable)
```

The parameter is:

`Enable`	Boolean. When you set this to "true", data notifications are switched on. Default: off.

To show and hide the terminal window

- Set the API control property `TerminalInteractive` (boolean).

  If you connect to a terminal, it stays visible until this value is set to "false".

  If you create a new terminal, it is invisible until this value is set to "true".

Associated Events:

- `CursorPositionChanged(XPosition, YPosition)`

  Fired when the terminal is in interactive mode and the cursor position is changed with the mouse (not when the cursor moves due to typing).

- `NewScreenDataArrived()`

  If enabled, this indicates that new data has arrived from the host.

Screen Data

Screen text is available as the raw text as it is received by the host and as the processed text as it is displayed on the terminal. The raw text contains all characters - including those that are not to be displayed (for example, password) - and can contain zero values.

Since the raw text can contain zero values, it can only be returned as an array of unsigned characters. The screen text is returned as an array of strings.

To return screen text

- Call the following:

```
APIReturn = GetScreenText(ScreenTextArray, TopLeftX, TopLeftY, BottomRightX, ↩
BottomRightY)
```

The parameters are:

`ScreenTextArray`	Variant Array(Strings). One string per line of text requested.
`TopLeftX`	Integer. Starting coordinate.
`TopLeftY`	Integer. Starting coordinate.
`BottomRightX`	Integer. Ending coordinate.
`BottomRightY`	Integer. Ending coordinate.

If any of the coordinates is set to -1, the entire screen is returned.

To return raw data

- Call the following:

```
APIReturn = GetScreenRawText(ScreenTextArray)
```

The parameter is:

`ScreenTextArray`	Variant Array(Unsigned chars). Raw data buffer.

To return screen attributes

- Call the following:

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

The parameters are:

Attributes	Variant Array(Unsigned chars). Attribute buffer.	
AttributesDescription	Variant Array(Unsigned chars). The description of an attribute is an array of 6 values containing the bit patterns for the attribute properties:	
	Member 0:	Attribute
	Member 1:	Protected
	Member 2:	Numeric
	Member 3:	No display
	Member 4:	High display
	Member 5:	Modify data tag

To return extended screen attributes

- Call the following:

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

The parameter is:

ExtendedAttributes	Variant Array(Unsigned chars). Extended attribute buffer.

To read and set the current cursor position

- Call the following:

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

The parameters are:

XPosition	Integer. X indicates the cursor position in the column.
YPosition	Integer. Y indicates the cursor position in the line.

To remove all editable text in the specified area

- Call the following:

```
APIReturn = ClearScreenText(TopLeftX, TopLeftY, BottomRightX, BottomRightY)
```

The parameters are:

TopLeftX	Integer. Starting coordinate.
TopLeftY	Integer. Starting coordinate.
BottomRightX	Integer. Ending coordinate.
BottomRightY	Integer. Ending coordinate.

-1 in any value indicates the whole screen.

To call the IF command used to check for screen text

- Call the following:

```
APIReturn = CheckForScreenText(Text, Result, Position, TopLeftX, TopLeftY, ↩
Length, CaseSensitive)
```

The parameters are:

Text	String. Text to check for.
Result	Boolean. "true" if the text was found.
Position	Integer. Screen position where the text was found.
TopLeftX	Integer. Starting coordinate.
TopLeftY	Integer. Starting coordinate.
Length	Integer. Text length.
CaseSensitive	Boolean. True if case-sensitive check.

Data Transfer

To prepare for data transfer to be processed directly by the API client

- Call the following:

```
APIReturn = SetAPIFileDetails(WorkFileNumber, UploadFlag, BinaryFlag, ReportFlag)
```

The parameters are:

WorkFileNumber	Integer. Work file number.
UploadFlag	Boolean. Is set for upload.
BinaryFlag	Boolean. Is set for binary transfer.
ReportFlag	Boolean. Is set for report format.

This results in the following events being fired during upload:

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

```
GetBinaryUploadFileBuffer(ErrorCode, WorkFileNumber, Data, DataLength)
```

and the following events being fired during download:

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

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

The event parameters are:

ErrorCode	Integer. Must be set to 0 by the API to indicate that all was processed without error.
FileNumber	Integer. The work file to be processed.
DataLength	Integer. Upload: the expected size is passed; the actual size is returned. Download: is set to the size of the transmitted data.
Data	Variant Arry(unsigned char). Data that are to be transferred.
DataFormat	String. Description of the record format.

For a normal transfer operation, the API client has to to provide a file name. This can be done by presetting a file name.

To preset a file name

- Call the following:

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

The parameters are:

Name	String. The file name that is to be used.
FileNumber	Integer. The work file being processed.
Upload	Boolean. Is set for upload.
Binary	Boolean. Is set for binary transfer.
Report	Boolean. Is set for report format.

If no preset values are found for the work file being processed, the API client will be asked for a file name.

To return a file name

- Respond to the following event:

```
APIReturn = GetFileName(ErrorCode, FileNumber, Upload, Binary, ToPrinter, ↩
Landscape, ControlChars, DosFormat, FileName)
```

The parameters are:

ErrorCode	Integer. If set to zero, the file name is used and processing starts. If set to any other value, processing is canceled.
FileNumber	Integer. Work file being processed.
Upload	Boolean. Is set for uploading a file name.
Binary	Boolean. Is set for binary transfer.
ToPrinter	Boolean. Is set to download to a printer.
Landscape	Boolean. Is set to print in landscape format.
ControlChars	Boolean. Is set to interpret control characters.
FileName	String. The file name to be used.

To cancel a running data transfer

- Call the following:

```
APIReturn = CancelFileTransfer(FileNumber)
```

The parameter is:

FileNumber	Integer. The number of the work file for which the data transfer is to be canceled.

This call is synchronous. It queues a cancelation request. When data transfer has completed, the FileTransferComplete event is fired.

Associated Events:

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

  The parameters are:

ErrorCode	Integer. If set to zero, the file name is used and processing starts. If set to any other value, processing is canceled.
FileNumber	Integer. The work file being processed.
Upload	Boolean. Is set for uploading a file name.
Binary	Boolean. Is set for binary transfer.
Headings	Variant Array (Strings). Contains the field names of the record for the transfer.

- FileTransferComplete(FileNumber, Upload, ErrorCode)

  The parameters are:

FileNumber	Integer. Work file being processed.
Upload	Boolean. Is set if upload is completed.
ErrorCode	Integer. Is set to zero if the data transfer was processed without error.

- FileTransferProgress(ProgressMessage)

  The parameter is:

ProgressMessage	String. Message that normally appears in the output window of the terminal application window.

Tasks and Procedure Files

To run an Entire Connection task or procedure file

- Call the following:

```
APIReturn = RunEntConTask(TaskName)
```

The parameter is:

TaskName	String. The name of an Entire Connection task or procedure file.

Note: For a synchronous connection, the application programming interface returns to the calling application after the TaskName has been checked and the task or procedure file has been started (not when the task or procedure file is completed). For an asynchronous call, the application programming interface immediately returns to the calling application.

To access the global parameters +PARM0 to +PARM9

- Call the following:

```
APIReturn = SetGlobalParameter(ParamNumber, Value) APIReturn = ↩
GetGlobalParameter(ParamNumber, Value)
```

The parameters are:

ParamNumber	Integer. From 0 to 9 for the required parameter.
Value	String. Value of the parameter.

To cancel a running procedure file (synchronous call only)

- Call the following:

```
APIReturn = CancelRunningTask()
```

This will return immediately. The procedure file will notify termination by firing the `EntConTaskComplete` event.

Associated Events:

- `EntConTaskStarting(ErrorCode, TaskName)`

  Is called when a task is started other than explicitly by the application programming interface (for example, a logon task).

  The parameters are:

`ErrorCode`	Integer. Has to be set to 0 (zero) so that the task can be started.
`TaskName`	String. The name of the task that has been started.

- `EntConTaskComplete(ErrorCode, TaskName)`

  The parameters are:

`ErrorCode`	Integer. Is set to zero if the task has completed without error.
`TaskName`	String. Task name.

- `TaskInputRequest(ErrorCode, DisplayOne DisplayTwo, Flags, ReturnData)`

  This event is fired if an `INPUT` statement is executed in a procedure file.

  The parameters are:

`ErrorCode`	Integer. Is set to zero after input has been provided.	
`DisplayOne`	String. First line of prompt text.	
`DisplayTwo`	String. Second line of prompt text.	
`Flags`	Variant Array. Display flags (see below).	
	`Flags(0)`	Must return some data; blank is invalid.
	`Flags(1)`	Numeric data only.
	`Flags(2)`	Password field.
	`Flags(3)`	Maximum length of the requested data.
`ReturnData`	String. Data that are to be returned to the procedure.	

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

This event is fired if a `WAIT` statement is executed in a procedure file.

The parameters are:

`ErrorCode`	Integer. Is set to 0 (zero) if the procedure is to continue. If zero is not set, the procedure is canceled.
`Text`	String. The message to be displayed.
`DialogBox`	Boolean. "true" if a message box is expected.
`MessageType`	Variant. Display parameters.
`Response`	Integer. Standard Microsoft response code of the `MessageBox` (for example, "IDOK") if `DialogBox` is "true".

- `TaskError(ErrorCode, ErrorText)`

  The parameters are:

`ErrorCode`	Integer. Error code from the task.
`ErrorText`	String. Message to be displayed.

Closing a Session

To close an open session and leave the connection to Entire Connection active

- Call the following:

```
APIReturn = CloseSession()
```

To close all terminals (asynchronous call only)

- Call the following:

```
APIReturn = CloseAllSessions()
```

  This will close any terminal session on the workstation, including those opened directly. This call should be used with caution. It also breaks the connection to the terminal. There is no completion event.

To break the link to the terminal (synchronous call only)

- Call the following:

```
APIReturn = BreakConnection(Closedown)
```

The parameter is:

`Closedown`	Boolean. Is set to "true" to close the terminal window on disconnect.

If `Closedown` is set to "false" and the Entire Connection terminal is not logged on, the terminal will be closed anyway. If the terminal was hidden, it will be automatically shown when the connection is broken.

Associated Events:

- `CurrentSessionClosed`

  The session has closed down without a request from the application programming interface. This can happen if the terminal is interactive and the user closes the session, or if a session times out.

- `TerminalClosedown`

  The terminal has completely closed down with no request from the application programming interface. This can happen in interactive mode if the user closes the application, or if `CloseAllSessions` is called from another API session.

Other Methods

To return the current size of the open terminal

- Call the following:

```
APIReturn = GetScreenSize(NumberOfRows, NumberOfColumns)
```

4 Other Events, Key Codes and Return/Error Codes

Other Events

- `ServerRequestedFileName(ErrorCode, OpenFile, Flags, Title, DefExtension, Filter, InitFileName, InitDirectory, FileName)`

  Is called if the session needs a file name.

  The parameters are:

`ErrorCode`	Integer. Is set to zero when the file name has been set.
`FileName`	String. The file name to be used.

  The other parameters are those expected by the common open file dialog.

- `TerminalWarningMessage(Message, DisplayFlag)`

  The parameters are:

`Message`	String. Message to be displayed.
`DisplayFlag`	Boolean. The call is expected to show a message in a blocking dialog box (for example, using the `MessageBox` function).

Key Codes

The table below shows the key codes that can be passed using the `PutData` function. The first column contains the function key name. The second column contains the function key constant as it is defined in the include file *ECAPI.H*, and the third column contains the key code value for the function key. Only these values should be used. If other values are passed, the effects are not defined.

The include file *ECAPI.H* is provided on the Entire Connection installation medium as part of the samples.

Function Key	Key Code Definition	Key Code Value
PF1	EC_PF1	20
PF2	EC_PF2	21
PF3	EC_PF3	22
PF4	EC_PF4	23
PF5	EC_PF5	24
PF6	EC_PF6	25

Function Key	Key Code Definition	Key Code Value
PF7	EC_PF7	26
PF8	EC_PF8	27
PF9	EC_PF9	28
PF10	EC_PF10	29
PF11	EC_PF11	30
PF12	EC_PF12	31
PF13	EC_PF13	32
PF14	EC_PF14	33
PF15	EC_PF15	34
PF16	EC_PF16	35
PF17	EC_PF17	36
PF18	EC_PF18	37
PF19	EC_PF19	38
PF20	EC_PF20	39
PF21	EC_PF21	40
PF22	EC_PF22	41
PF23	EC_PF23	42
PF24	EC_PF24	43
ATTN	EC_ATTN	46
CLEAR	EC_CLEAR	16
CR	EC_CR	13
DEVCNCL	EC_DEVCNCL	50
EEOF	EC_EEOF	54
ERASEINP	EC_ERASEINP	44
INSERT	EC_INSERT	82
NEWLINE	EC_NEWLINE	48
PRINT	EC_PRINT	49
SYSREQ	EC_SYSREQ	47
HOME	EC_HOME	71
PA1	EC_PA1	17
PA2	EC_PA2	18
PA3	EC_PA3	19
DELETE	EC_DELETE	83
BACKSPACE	EC_BACKSPACE	8
TAB	EC_TAB	9
BACKTAB	EC_BACKTAB	15

Function Key	Key Code Definition	Key Code Value
LEFT	EC_LEFT	75
RIGHT	EC_RIGHT	77
UP	EC_UP	72
DOWN	EC_DOWN	80
DUE2	EC_DUE2	56
EM	EC__EM	84
AFZ	EC_AFZ	11
EFZ	EC_EFZ	165
LZE	EC_LZE	89
RU	EC_RU	163
SDZ	EC_SDZ	160
SZA	EC_SZA	85
K1	EC_K1	193
K2	EC_K2	194
K3	EC_K3	195

Return/Error Codes

The return/error codes are all integer values. The constants listed below are defined in the include file *ECAPI.H*. The numbers in parentheses are the actual code values.

The include file *ECAPI.H* is provided on the Entire Connection installation medium as part of the samples.

`API_SUCCESS (0)`
: Returned from most functions if the operation was successful. Some functions have specific success return codes - see below.

`API_CALL_QUEUED (-1)`
: This return code is used in asynchronous (non-blocking) mode. It means that the request from the API application has successfully been sent to Entire Connection for processing. The return code for the request from Entire Connection is passed in a completion event to the API application.

`API_NEW_SESSION_OPENED (-2)`
: Returned by the `Initialize` API function if a new session has been created successfully.

`API_PROC_CANCELLED_OK (-3)`
: Sent as completion event for the `CancelRunningTask` API function if the Entire Connection task or procedure file has been canceled successfully.

`API_ERROR_CALL_BLOCKED (1)`
This return code is used internally. It is not passed to the API application.

`API_ERROR_INCORRECT_PARAMETERS (2)`
Each API function checks whether the passed parameters are valid. If not, this error code is returned.

`API_ERROR_NO_USER (10)`
In order to use a terminal, a user has to log on to Entire Connection. This error code is returned if you called a function requiring a terminal but no user has logged on yet. Use the API function `LogonEntireConnection` to log on.

`API_ERROR_NO_OPEN_SESSION (11)`
This error code is returned by API functions that work on an open terminal session if there is no open terminal session. You first have to open a session, for example, with the API functions `GetAvailableSessions` and `OpenSession`.

`API_ERROR_NO_FILE_TRANSFER (12)`
The API function `CancelFileTransfer` returns this error code if there is no active file transfer.

`API_ERROR_NO_SESSIONS_DEFINED (13)`
The API function `GetAvailableSessions` returns this error code if no sessions are defined in the share file for the current user.

`API_ERROR_NO_SCREEN_PRESENT (14)`
The API function `GetScreenText` returns this error code if no screen data is available because the first screen from the host has not yet arrived.

`API_ERROR_NO_SESSION_NAME (15)`
The API function `OpenSession` returns this error code if no session name was passed in the parameter `SessionName`.

`API_ERROR_NO_TASK_RUNNING (16)`
The API function `CancelRunningTask` returns this error code if there is no active task or procedure.

`API_ERROR_NOT_CONNECTED (20)`
This error code is returned by the API functions if the API ActiveX control is not connected to Entire Connection. For example, if Entire Connection was manually closed by a user.

`API_ERROR_ALREADY_CONNECTED (21)`
The API function `Initialize` returns this error code when the function has already been called before and returned successfully.

`API_ERROR_ALREADY_LOGGED_ON (22)`
The API function `LogonEntireConnection` returns this error code if the user is already logged on to Entire Connection.

`API_ERROR_ALREADY_INITIALIZED (23)`
The API function `Initialize` returns this error code if the API ActiveX control is already attached to Entire Connection.

`API_ERROR_SESSION_ALREADY_OPEN (24)`
The API function `OpenSession` returns this error code if there already is an open session.

`API_ERROR_SESSION_NOT_FOUND (30)`
This return code is currently not used.

`API_ERROR_API_CALL_ONLY (31)`
This error code is used in Entire Connection if API functions are called but there is no active API application.

`API_ERROR_INITIALIZATION_FAILED (40)`
The API function `Initialize` returns this error code if the API ActiveX control could not be initialized or if it could not be attached to Entire Connection.

`API_ERROR_CALL_FAILED (41)`
This error code is used by the API functions if Entire Connection could not complete the requested operation successfully and it did not return a specific error code.

`API_ERROR_COMMS_ERROR (200)`
This return code is currently not used.

`API_ERROR_INTERNAL_ERROR (201)`
This error code is returned when an unexpected error or exception occurred. At least the requested operation was aborted, and failed. Entire Connection may be instable. Restart Entire Connection and retry.