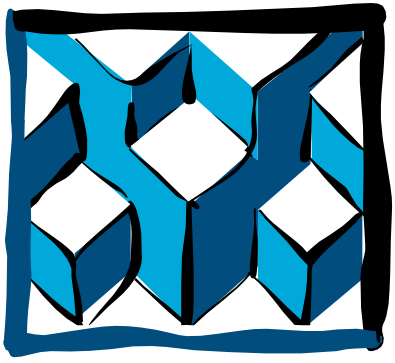
SBClient Programmer's Guide



SBClient®

A 4GL Tool Set for Rapid Application Development and Guitization

Ardent Software, Inc.

SBClient Release 5
Documentation DSC-5003
Rev. 02032000

Ardent Software, Inc.
50 Washington Street
Westboro, Massachusetts USA 01581-1021
Telephone: (508) 366-3888
Facsimile: (508) 366-3669
World Wide Web: www.ardentsoftware.com

Principal Technical Writer
Tony Watkin

Contributors and Technical Reviewers
Mark Chazan, and Mike Goddard

Contents

PART 5: APPENDICES

Preface

Introduction to Ardent Product Documentation

This preface contains useful information about Ardent's conventions for documenting command syntax and system output, as well as providing explanations of the specially designed icons that indicate important notes, tips, and warnings in the documentation.

Features of Ardent Manuals

Ardent strives to produce useful, high quality documentation while maintaining a consistent look and feel. It will help you to know the following conventions which are common to this and all Ardent manuals. This section explains the following:

- conventions
- elements of syntax statements
- notes, warnings, and tips
- screen captures

Conventions

All Ardent manuals depict command syntax according to industry standards. The following table lists the syntax conventions:

Convention	Description
`bold courier font`	Bold courier font indicates required commands that you must enter in the manner shown. You must enter all punctuation marks that appear in bold, unless otherwise indicated.
`courier font`	Courier font indicates system output, such as prompt signs, responses to commands, and program code. Courier font is also used to depict paths to directories or files.
bold	Bold font indicates the name of an element on the screen, such as a menu, a form, or a prompt. Also indicates the names of files or directories.
lower case italics	Lower case italics is used for syntactical expression of user-supplied words, variables, and expressions
xx	Italic lower-case xx indicates a placeholder for a system id. For example, *xx*`CONTROL`.

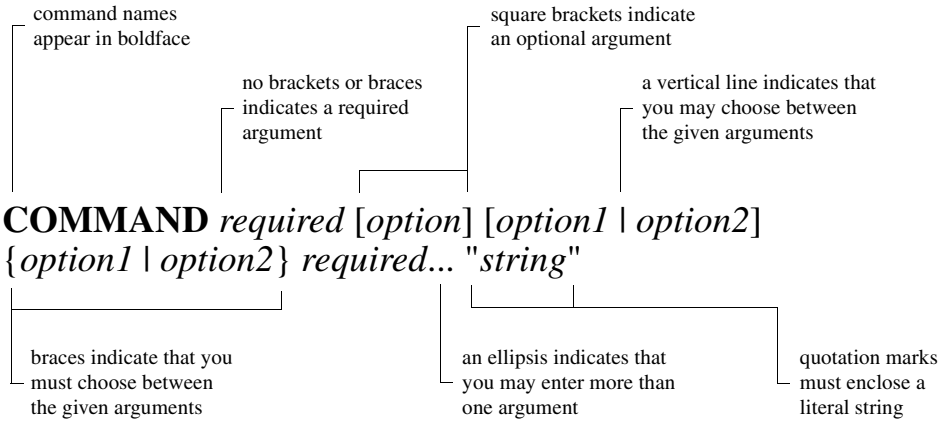
Conventions

Convention	Description
>	The > character separates each option in a menu hierarchy. For example, “choose **Setup > Miscellaneous**”, means choose the Setup menu then the Miscellaneous option.
[ ]	Square brackets indicate an item that is optional. You may include a number of items enclosed in brackets in a Unidata command or function.
{ }	Braces indicate that you must choose one of the items separated by the vertical bar (|).
|	Vertical bar separates option arguments from which you may choose.
. . .	An ellipsis (...) after an argument indicates that you may use more than one argument on a single command line.

Conventions (continued)

Elements of Syntax Statements

in some cases commands may be entered as part of paragraphs. The syntax statement includes the command name, required arguments, and options that you can use with the command. Italics represents a variable that you can replace with any valid option. The following figure illustrates the elements of a syntax statement:



Important Notes, Warnings, and Tips

You will notice several different icons throughout this manual. These icons draw attention to important information about the product.

Note

A **Note** icon indicates important additional information on the subject.

Warning

A **Warning** icon alerts you to the danger of deleting or corrupting data.

Reminder

A **Reminder** icon marks information that is discussed in detail in another chapter or another Ardent manual.

Tip

A **Tip** icon denotes shortcuts, commands, or procedures that may help you use Ardent products more efficiently.

PART I: INTRODUCTION

This section of the manual introduces you to overall concepts built into SBClient.

Chapter 1 - Overview

This chapter provides an introduction to the capabilities of SBClient. Programmers can use these facilities to integrate legacy systems with Windows.

Introduction

Programmers can use SBClient to enhance traditional host-based applications and run them from the Microsoft Windows environment.

There are two categories of enhancements:

- **Windows Integration**. This refers to integrating a host-based application within the Windows desktop without changing the look and feel of the host-based application itself; the application still runs in character mode using terminal emulation. GUI objects (push buttons and so on) may be added to provide commonly used facilities, such as launching and interacting with Window applications, and transferring files between the host and the PC.
- **GUItization**. This refers to changing the look and feel of the host-based application from character mode to Windows GUI mode.

Windows integration and GUItization are facilitated by the SBClient host library – a collection of BASIC subroutines that can be installed on the host (see SBClient Host Library).

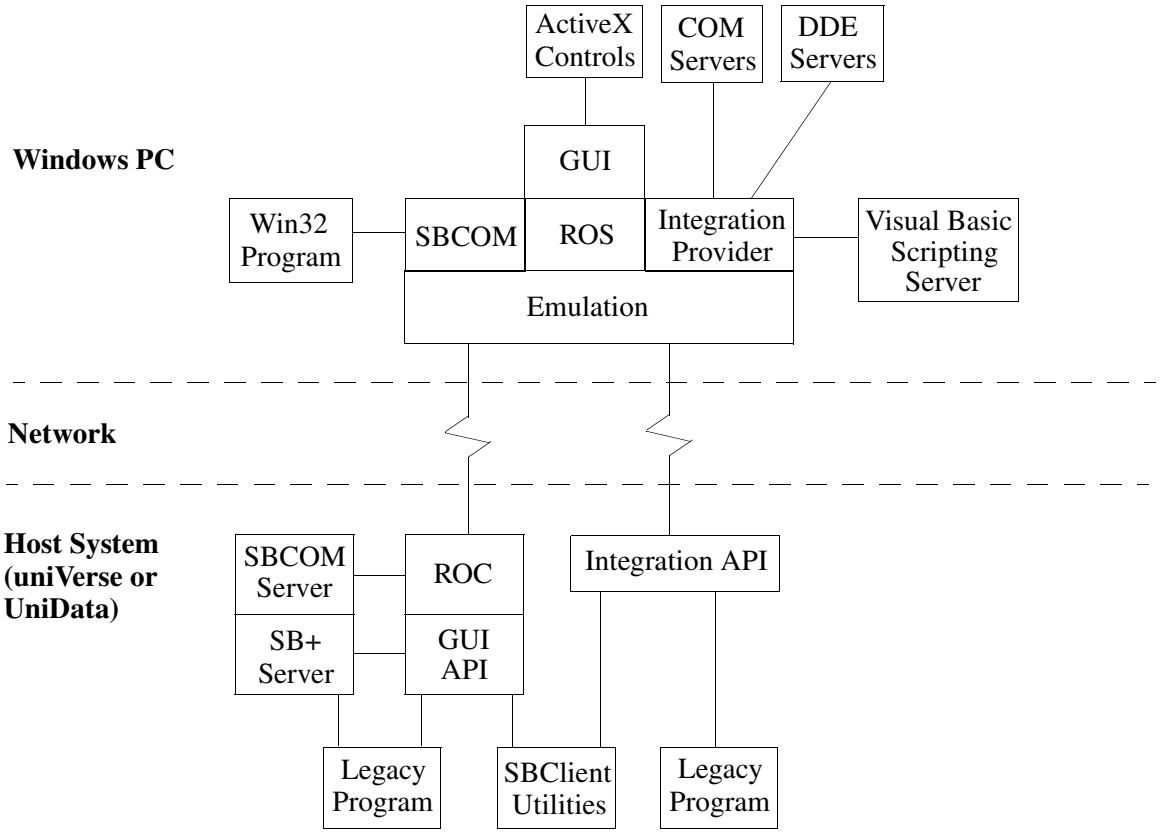
SBClient employs a client/server architecture that is a logical extension of the basic client/server model of a character terminal.

In essence, the character terminal is a character display server. It receives commands (escape sequences) which are interpreted as terminal functions (for example, clear screen, position cursor and so on). The character terminal also generates events (character strings) to inform the host that the user has carried out an action, such as pressing a key.

SBClient's client/server model extends this basic model. In the same way that commands are sent to a character terminal, the Remote Object Server (ROS) receives and processes commands that create and manipulate user interface objects. Additionally, ROS notifies the host of events such as the user clicking a button or closing a window.

SBClient Architecture

SBClient is an object-oriented application. It incorporates objects that control a range of functions (Windows integration, GUItization, file transfer, communication, terminal emulation and so on). The following diagram shows the structure of SBClient and it's relationship with SB+ Server.



Chapter 2 - Introduction to the Host Library

This chapter describes the integration of SBClient with the Windows environment.

SBClient Host Library

The SB+Client host library is a collection of BASIC subroutines that allow programmers to integrate host and PC functionality and to convert character screens and menus to GUI forms. The library subroutines are essentially collections of escape sequences. The escape sequences are sent to the PC when you invoke the subroutine, saving programmers from having to remember and execute complicated escape sequences.

Before using the SBClient host library, you must install the host library subroutines on the host. See Installing the Host Library.

The following header items may need to be included by users of the host library subroutines:

- SPECIAL.H. Contains equates for special keys, such as Enter and Esc.
- ROC.H. Include this header in applications that use the GUItization subroutines of the host library.
- OBJECT.H. Include this header in applications that use the GUItization subroutines of the host library.
- USER.INCLUDE.H. A header item for use by developers. This is initially empty, and allows developers to include their own subroutines.

The Host Commands topic provides details of commands for transferring data between the host and PC.

Windows integration subroutines fall into the following categories:

- DDE Client Subroutines
- Data Transfer Subroutines
- Windows Process Control Subroutines
- PC File Handling Subroutines
- PC Printer Control Subroutines
- Character Windows Subroutines
- SQL Gateway Subroutines

- MAPI Mail Subroutines
- VBScript
- Object Manipulation
- Miscellaneous Windows Integration Subroutines

Windows GUItization subroutines fall into the following categories:

- GUI Form Handling Subroutines
- GUI Menu Handling SubroutinesGeneric Object Routines
- Miscellaneous GUItization Subroutines

The Demonstration Programs topic provides a description of programs included in SBClient's host library demonstrating various aspects of the library's functions.

PART II: WINDOWS INTEGRATION

This section of the manual introduces you to overall concepts built into SBClient.

Chapter 3 - Character Windows

This chapter describes advanced aspects of SBClient's interaction with the Windows environment.

Character Windows

These subroutines are used to control character (that is, non-GUI) windows.

Character windows subroutines work in SBClient's character emulation mode. Character windows are composed from the box drawing characters available with dumb terminal emulations. They are not related to Windows standard graphical windows.

Character windows subroutines are:

- TU.WINDOW.DRAW . Draws a window frame and clears the contents of the window.
- TU.WINDOW.RESTORE . Restores a previously saved window.
- TU.WINDOW.SAVE . Saves the contents of a window so it can be redisplayed later.

TU.WINDOW.DRAW

TU.WINDOW.DRAW(*col*, *row*, *width*, *height*, *frame*, *shadow*)

Draws a window frame and clears the contents of the window.

col

[P] Position of window's left-most column (in characters).

row

[P] Position of the window's top row (in characters).

width

[P] Total width of window (in characters, including horizontal lines).

height

[P] Total height of window (in characters, including vertical lines).

frame

[P] Frame type. Valid values are:

0 no frame

1 single line

2 double line

shadow

[P] Shadow type. Valid values are:

0 no shadow

1 shadow

See also

TU.WINDOW.RESTORE, TU.WINDOW.SAVE

TU.WINDOW.RESTORE

TU.WINDOW.RESTORE(*level*)

Restores the contents of a previously saved window. Up to 51 window images can be saved on levels ranging from 0 to 50.

level

[P] Level of previously stored window. Valid values range from 0 to 50.

See also

TU.WINDOW.DRAW, TU.WINDOW.SAVE

TU.WINDOW.SAVE

TU.WINDOW.SAVE(*level*, *col*, *row*, *width*, *height*)

Saves the contents of a window so it can be redisplayed later. Up to 51 window images can be saved on levels ranging from 0 to 50.

level

[P] Level in which the window is stored. Valid values range from 0 to 50.

col

[P] Position of the window's left-most column (in characters).

row

[P] Top row (in characters).

width

[P] Total width of window (in characters).

height

[P] Total height of window (in characters).

Note

To save the entire screen, enter 0 for col, row, height and width.

See also

TU.WINDOW.DRAW, TU.WINDOW.RESTORE

Chapter 4 - Data Transfer

These subroutines are used to upload and download data between the PC and the host.

Data Transfer subroutines are:

- TU.ANSI.TO.OEM. Converts an ANSI file to an OEM file.
- TU.DOWNLOAD . Downloads a single host record to a PC file.
- TU.OEM.TO.ANSI. Converts an OEM file to an ANSI file.
- TU.PC.DOWNLOAD . Transfers multiple host records to PC files.
- TU.PC.UPLOAD . Uploads data from a PC file into host records.
- TU.TO.EXCEL . Transfers data from host records to an Excel spreadsheet.
- TU.TO.EXCEL.GRAPH . Transfers data from host records to an Excel chart.
- TU.TO.MONOLOG . Transfers a phrase to Monolog, the Sound Blaster text-to-speech application.
- TU.TO.WORD . Transfers data from host records into a Word document in tab delimited format.
- TU.TO.WORD.BOOKMARK . Transfers data from a single host record into a Word bookmark. The record name should be the same as the bookmark name in the document.
- TU.TO.WORD.MERGE . Transfers data from host records into a Word mail merge document.
- TU.UPLOAD . Uploads a single PC file to a host record.

Data Transfer API

TU.ANSI.TO.OEM

TU.ANSI.TO.OEM(*oemfile*, *ansifile*, *status*)

Converts an ANSI file to an OEM file on the PC. You should use this function when transferring from a Windows-based application to an OEM host. The Windows operating system uses an ANSI character set, however most hosts use an OEM character set. The difference between and ANSI and OEM character set is the high bit characters. Languages such as English do not use these high bit characters and conversion is not normally required. However, languages like Spanish use the high bit characters for accented characters and require the conversion.

ansifile

[P] The DOS path and name of the ANSI file to convert to the OEM character set.

oemfile

[P] The name of the file to contain converted file in the OEM character set. If oemfile is the same as ansifile, the original file is replaced with the converted file. If oemfile is not specified, and ansifile is specified, oemfile is set to ansifile, and the original file is replaced with the converted file.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. You should call DDE.GET.ERROR to retrieve details about a failure.

See also

TU.OEM.TO.ANSI

TU.OEM.TO.ANSI

TU.OEM.TO.ANSI(*oemfile*, *ansifile*, *status*)

Converts an OEM file to an ANSI file on the PC. You should use this function when transferring from an OEM host to a Windows-based application. The Windows operating system uses an ANSI character set, however most hosts use an OEM character set. The difference between and ANSI

and OEM character set is the high bit characters. Languages such as English do not use these high bit characters and conversion is not normally required. However, languages like Spanish use the high bit characters for accented characters and require the conversion.

oemfile

[P] The DOS path and name of the OEM file to convert to the ANSI character set.

ansifile

[P] The name of the file to contain converted file in the ANSI character set. If ansifile is the same as oemfile, the original file is replaced with the converted file. If ansifile is not specified and oemfile is specified, ansifile is set to oemfile, and the original file is replaced with the converted file.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. You should call DDE.GET.ERROR to retrieve details about a failure.

See also

TU.ANSI.TO.OEM

TU.DOWNLOAD

TU.DOWNLOAD(*data*, *pcfilename*, *options*, *description*, *status*)

Transfers a host record to a PC file. To transfer multiple records, refer to TU.PC.DOWNLOAD and TU.PC.UPLOAD.

When transferring binary data (for example, bitmaps, help files, and executables) on some host platforms, the data may become corrupt when you read it from the database. In operating environments that support directory type files, such as uniVerse and UniData, you use the B option only to transfer the data and store in in directory type files. If you wish to store data in hash files or your system doesn't support directory type files the you must hex encode the data.

data

[P] The actual data to be transferred.

pcfilename

[P] The name of the PC file in which the data is to be stored.

options

[P] An array of characters indicating optional transfer details. Valid values are:

- A Append to existing file
- B data is in Binary format
- H Host initiated transfer
- L Local transfer, simple protocol
- O Overwrite the PC file, if it exists
- R non-Resilient line, acknowledge every packet, do not stream
- V strip Value-marks and subvalue-marks
- X hexadecimal output
- Z no status window
- 7 7 bit data link
- M Minimal status

description

[P] A string describing the data being transferred. description is displayed in the SBClient dialog box.

status

[R] The result of the transfer. Valid values are:

- 0 file has been transferred successfully
- 1 user abort
- 2 unable to open PC file
- 3 PC read error

4 communication error

5 retry limit exceeded

6 PC write error

7 file transfer not supported

10 unknown error

11 file exists and override not specified

See also

TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.PC.DOWNLOAD

TU.PC.DOWNLOAD(*hostfilename*, *fieldlist*, *selection*, *pcfilename*, *separators*, *options*, *description*, *status*)

Transfers multiple host records to a PC file. TU.PC.DOWNLOAD invokes TU.DOWNLOAD to perform the actual download for each record.

hostfilename

[P] The name of the host file from which records are to be downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded. The definitions can include the modifiers BREAK-ON and TOTAL, which mean the same as they do in the database query language. The TOTAL-SUPP modifier suppresses the output of a grand total at the end of the file. To transfer a complete record, set this parameter to null.

selection

[P] The selection criteria by which the records are screened. Records that meet the selection criteria are selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

pcfilename

[P] The name of the PC file in which the data is to be stored.

separators

[P] A flag indicating what field separators are to be used in the PC file. Valid values are:

- 0 no delimiters, fixed length based on dictionary
- 1 strings in quotes, commas between data
- 2 tabs between fields
- 3 spaces between fields
- 4 no field separators, single record transfer

options

[P] An array of characters indicating optional transfer details. Valid values are:

- A Append to existing file
- B data is in Binary format
- H Host initiated transfer
- L Local transfer, simple protocol
- O Overwrite the PC file, if it exists
- R non-Resilient line, acknowledge every packet, do not stream
- V strip Value-marks and subvalue-marks
- X hexadecimal output
- Z no status window
- 7 7 bit data link
- 0 add duplicate data for correlative

1 insert column headers into data

2 insert blank line before data

M Minimal status

description

[P] A string describing the data being transferred. description is displayed in the SBClient dialog box.

status

[R] The result of the transfer. Valid values are:

0 file has been transferred successfully

1 user abort

2 unable to open PC file

3 PC read error

4 communication error

5 retry limit exceeded

6 PC write error

7 file transfer not supported

10 unknown error

11 file exists and override not specified

See also

TU.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.PC.UPLOAD

TU.PC.UPLOAD(*hostfilename*, *fieldlist*, *pcfilename*, *recordtype*, *separators*, *options*, *description*, *status*)

Uploads data from a PC file into host records. TU.PC.UPLOAD invokes TU.UPLOAD to perform the actual upload for each host record.

hostfilename

[P] The name of the host file into which the data is to be uploaded.

fieldlist

[P] List of dictionary field definitions used to build the host record from incoming PC data. The field definitions determine the property positions that the PC field needs to be placed in and is also used for any internal conversions that need to be applied to the data. Field definitions that are correlatives (I-types) are invalid and will cause the subroutine to exit with an error status.

pcfilename

[P] The name of the PC file that contains the data being uploaded.

recordtype

[P] A flag indicating how the host record is to be constructed. Valid values are:

0 use pcfilename as record id (the default)

1 use sequential number as id

2 use first field of record as id

separators

[P] A flag indicating the field separators that are used in the PC file. Valid values are:

0 no delimiters, fixed length based on dictionary (the default)

1 strings in quotes, commas between data

2 tabs between fields

3 spaces between fields

4 no field separators, single record transfer

options

[P] An array of characters indicating optional transfer details. Valid values are:

- B data is in Binary format
- H Host initiated transfer
- L Local transfer, simple protocol
- O Overwrite record, if it exists
- R non-Resilient line, acknowledge every packet, do not stream
- S Spooler output (interchangeable with P)
- P spooler output (interchangeable with S)
- X hexadecimal output
- Z no status window
- 7 7 bit data link
- M Minimal status

description

[P] A string describing the data being transferred. description is displayed in the SBClient dialog box.

status

[R] The result of the transfer. Valid values are:

- 0 file has been transferred successfully
- 1 user abort
- 2 unable to open PC file
- 3 PC read error
- 4 communications error
- 5 retry limit exceeded

6 PC write error

7 file transfer not supported

9 invalid field name specified

10 unknown error

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.EXCEL

TU.TO.EXCEL(*hostfilename*, *fieldlist*, *selection*, *options*, *sheetname*, *status*)

Transfers data from host records to an Excel spreadsheet. TU.TO.EXCEL invokes when the F option is selected.

This subroutine is only supported by Microsoft Office 95 and Office 97.

hostfilename

[P] The name of the host file from which the data is downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded. The definitions can include the modifiers BREAK-ON and TOTAL, which mean the same as they do in database query language. The TOTAL-SUPP modifier suppresses the output of a grand total at the end of the file.

selection

[P] The selection criteria by which the records are screened. Records that meet the selection criteria are selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

options

[P] An array of characters indicating optional transfer details. Valid values are:

- C Clear the destination before downloading
- F use File transfer protocol to send data
- H transfer Heading
- I convert the file from the OEM character set to the ANSI character set
- M transfer field Masks
- O Overwrite the destination if it exists
- P Print the document on the default printer
- 0 add duplicate data for correlative
- 1 insert column headers into data
- 2 insert blank line before data

sheetname

[P] The name of the Excel worksheet.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

The following example downloads records into an Excel spreadsheet.

Program Example

```
INCLUDE TUBP TU.API.H
INCLUDE TUBP SPECIAL.H
*
HOSTFILE = 'CARS'
FLDLIST = "REG MANUF MODEL COLOR RADIO AVAILABLE MILES"
SELECTION = 'BY MANUF BY MODEL BY REG WITH COLOR = "RED" AND WITH AVAILABLE
   = "Y"'
OPTIONS = 'HOMF'; SHEET = "E:\TEMP\CARS.XLS"; STATUS = 0
CALL TU.TO.EXCEL(HOSTFILE,FLDLIST,SELECTION,OPTIONS,SHEET,STATUS)
```

```
IF STATUS THEN CRT "ERROR: ":STATUS ELSE CRT "ok"
END
```

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.EXCEL.GRAPH

TU.TO.EXCEL.GRAPH

TU.TO.EXCEL.GRAPH(*hostfilename*, *fieldlist*, *selection*, *options*, *sheetname*, *graphtype*, *status*)

Transfers data from a host record to an Excel graph. It invokes when the F option is selected.

This subroutine is only supported by Microsoft Office 95 and Office 97.

hostfilename

[P] The name of the host file from which the data is to be downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded. The definitions can include the modifiers BREAK-ON and TOTAL, which mean the same as they do in database query language. The TOTAL-SUPP modifier suppresses the output of a grand total at the end of the file.

selection

[P] The selection criteria by which the records are screened. Records that meet the selection criteria are selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

options

[P] An array of characters indicating optional transfer details. Valid values are:

C Clear the destination before downloading

F use File transfer protocol to send data

H transfer Heading

I convert the file from the OEM character set to the ANSI character set

M transfer field Masks

O Overwrite the destination if it exists

P Print the document on the default printer

sheetname

[P] The name of the Excel worksheet.

graphtype

[P] The chart type. Valid values are:

1 AREA

2 BAR

3 COLUMN

4 LINE

5 PIE

6 XYSCATTER

7 3D.AREA

8 3D.COLUMN

9 3D.LINE

10 3D.PIE

11 RADAR

12 3D.BAR

13 3D.SURFACE

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

The following example downloads records into an Excel chart.

Program Example

```
INCLUDE TUBP TU.API.H
INCLUDE TUBP SPECIAL.H
*
HOSTFILE = 'CARS'
FLDLIST = "MANUF TOTAL MILES"
SELECTION = 'BY MANUF BY MODEL BY REG WITH COLOR = "RED" AND WITH AVAIL-
ABLE          = "Y"'
OPTIONS = 'HOMF'; SHEET = "E:\TEMP\CARS.XLS"; STATUS = 0 ; GRAPH = 5
CALL TU.TO.EXCEL.GRAPH(HOSTFILE, FLDLIST, SELECTION, OPTIONS, SHEET, GRAPH,
STATUS)
IF STATUS THEN CRT "ERROR: ":STATUS ELSE CRT "ok"
END
```

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.MONOLOG

TU.TO.MONOLOG(*phrase*, *status*)

Transfers a phrase to Monolog, the Sound Blaster text-to-speech application. You must ensure Monolog has DDE enabled.

phrase

[P] The text to be spoken. You may need to experiment with phonetic spelling to achieve the most accurate pronunciation.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.WORD

TU.TO.WORD(*hostfilename*, *fieldlist*, *selection*, *options*, *documentname*, *status*)

Transfers data from a host record into a Word document in tab delimited format. This data will be converted into a table. This subroutine could be used with the TOTAL/DET-SUPP options to include a grand total or summary information in a document.

This subroutine is only supported by Microsoft Office 95 and Office 97.

hostfilename

[P] The name of the host file from which the data is to be downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded. The definitions can include the modifiers BREAK-ON and TOTAL, which mean the same as they do in database query language. The TOTAL-SUPP modifier suppresses the output of a grand total at the end of the file.

selection

[P] The selection criteria by which the records are screened. Records that meet the selection criteria are selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

options

[P] An array of characters indicating optional transfer details. Valid values are:

C Clear the destination before downloading

H transfer Heading

I convert the file from the OEM character set to the ANSI character set

P Print the document on the default printer

R Restore the application window at completion

S Save document

X Close the Microsoft Word application

0 add duplicate data for correlative

2 Insert blank line between header and data

documentname

[P] The name of the Word document.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

The following example places data into a Word document. The data is tab delimited to allow the data to be easily converted to a table format.

Program Example

```
INCLUDE TUBP TU.API.H
INCLUDE TUBP SPECIAL.H
*
HOST = "CARS"; FIELD = "REG MILES COLOR"
DOCUMENTNAME = 'C:\TMP\TU.DOC'
SELECT = 'WITH COLOR = "RED" AND WITH AVAILABLE = "N"
OPTIONS = "XSRH"; DOC = "D:\TEMP\CARS.DOC"; STATUS = 0
*
CALL TU.TO.WORD (HOST, FIELD, SELECT, OPTIONS, DOC, STATUS)
IF STATUS THEN CRT "ERROR: ": STATUS ELSE CRT "OK"

END
```

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.WORD.BOOKMARK

TU.TO.WORD.BOOKMARK(*hostfilename*, *fieldlist*, *selection*, *options*, *documentname*, *status*)

This function transfers data from a host record to pre-defined bookmarks inside a Microsoft Word document, and uses DDE (Dynamic Data Exchange) or OLE as the vehicle to do this. A typical scenario where you might want to use this function is where you generate invoices using Microsoft Word. You could, for example, store details about the client inside the database. These details would typically include; contact name, client name, client address, and so on. The TU.TO.BOOKMARK function could be used to transfer a copy of these details to the top of an invoice that has been created in Microsoft Word for Windows.

You would not use the TU.TO.WORD.BOOKMARK function if you want to retrieve "multiple" records from the database such as retrieving the details of several clients.

TU.TO.WORD.BOOKMARK is only supported for Microsoft Office 95 and Office 97.

An example illustrating how to use the TU.TO.WORD.BOOKMARK has been provided further below. A similar example has also been supplied with SBClient. If you want to run the example that comes with SBClient then enter RUN DEMOBP WORD.BOOKMARK.

hostfilename

[P] The name of the host file from which the data is to be downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded. The description of the dictionaries is used to correlate to the name of the bookmark in the word document. I.e. the description of the dictionary must be the same as the name of the bookmark. The definitions can include the modifiers BREAK-ON and TOTAL, which mean the same as they do in database query language. The TOTAL-SUPP modifier suppresses the output of a grand total at the end of the file.

selection

[P] The selection criteria by which the records are screened. The record that meet the selection criteria is selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

options

[P] An array of characters indicating optional transfer details. Valid values are:

I Do OEM to ANSI conversion

S Save document

X Close Microsoft Word

P Print the document on the default printer

R Restore the application window at completion.

documentname

[P] The name of the Word document. The document must be created and the bookmark inserted before the transfer is initiated.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Example

1. Create a new account for the purposes of this exercise and log onto it (any name will do).
2. Create a file e.g. REPOSIT. We will use this file to store the data (both the dictionaries and records).
3. Create a programming file e.g. PROGS. We will use this file to store the program that uses the TU.TO.WORD.BOOKMARK function.
4. Create dictionaries F0, F1, F2, and F3 in the REPOSIT file with the contents shown in the following table:

5. Now create an item in the REPOSIT file called UDT1. This will be used to store the records for this example. The data in this item should be three entries as shown below:
6. Now that we have the data, we need to create bookmarks called SBC1, SBC2, and SBC3. You should place these where you want the data to appear in the Word document. The following shows how the Word document might look before it is populated with data from the database:
7. Save the Word document with the name SBC3.doc.
8. We are now ready to create the program which will use the data in the REPOSIT file. In the file PROGS create a program and call it TEST.WORD.BOOKMARK. You will notice that the TU.TO.WORD.BOOKMARK function uses various parameters, namely; host, fieldlist, selection, options, documentname, status . These terms are explained in further detail at the top of this topic. Type in the code for the program TEST.WORD.BOOKMARK as shown in the following diagram, and provide the path to SBC3.doc on your network.
9. Enter RUN PROGS TEST.WORD.BOOKMARK to run the program.
10. If you now open the Microsoft Word document you will see that the program populated the Microsoft Word document with the details from the database:

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.MERGE, TU.UPLOAD

TU.TO.WORD.MERGE

TU.TO.WORD.MERGE(*hostfilename*, *fieldlist*, *selection*, *options*, *documentname*, *status*)

Transfers data from host records into a Word mail merge document.

This routine is supported for Microsoft Office 95 and Microsoft Office 97.

hostfilename

[P] The name of the host file from which the data is to be downloaded.

fieldlist

[P] A list of dictionary field definitions to be downloaded.

selection

[P] The selection criteria by which the records are screened. Records that meet the selection criteria are selected for downloading. If the record list has been previously selected, selection can be GET-LIST listname.

BY-EXP selections are not supported.

options

[P] An array of characters indicating optional transfer details. Valid values are:

I convert the file from the OEM character set to the ANSI character set.

O Add duplicate data for correlative.

P Print the document on the default printer.

R Restore the application window at completion.

S Save document.

X Exit Microsoft Word.

documentname

[P] The name of the Word document. The Word document must be created before the transfer is initiated. The document must be defined as a mail merge document, whose data source file is named “merge.doc”. This data source document must co-exist with the merge document.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

There is an example in the DEMOBP file called WORD.MERGE which demonstrates use of this routine. The steps that follow cover both the actions you would typically perform in Microsoft Word and with SBClient.

1. Create a Microsoft Word document with a table that contains the following details:
2. Save the document with the filename “merge.doc”.
3. Create a new Microsoft Word document.
4. Select the command Tools > Mail merge.

5. A dialog will appear as follows:
6. From the Mail Merge Helper dialog under 1 Main document > Create ”select the appropriate option. For the purpose of this example we will choose Form Letters…
7. A dialog will appear prompting you specify whether you want to use the current document or want to create a new main document. For this exercise click on the Active Window button.
8. From the Mail Merge Helper dialog under 2 Data source > Get Data navigate to the location of merge.doc and select it, then click on the Open button.
9. An information dialog will appear. Click on the Edit Main Document button.
10. Insert the merge fields into the document as shown in the following example:
11. Place as appropriate in the document. The following illustration shows all the options which have been set for a merge document:
12. If you have the Mail Merge Helper dialog on display press the Esc key or click on the Cancel button to close it.
13. Select **File > Save As** and specify a *filename* and save the file at your preferred location.

Note

Both the data source and the merged document must reside in the same directory

14. Launch SBClient
15. Create the Data BASIC code that will be used to run the mail merge. An example of how this code might look is provided below:

Program Example

```
INCLUDE TUBP TU.API.H
INCLUDE TUBP SPECIAL.H
*
HOSTFILE = "CARS"; FIELDLIST = "REG MANUF MODEL MILES"
SELECTION = 'WITH COLOR = "BLACK" AND WITH AVAILABLE = "Y" BY MANUF BY MODEL BY
REG'
OPTIONS = ''; DOCUMENT = 'D:\TEMP\TEST.DOC'; STATUS = 0
*
```

```
CALL TU.TO.WORD.MERGE(HOSTFILE, FIELDLIST, SELECTION, OPTIONS, DOCUMENT, STATUS)
IF STATUS THEN CRT 'ERROR:- ':STATUS ELSE CRT "OK, BYE"
END
```

16. Run the program, for example by typing:

    Run BP programname

17. SBClient will perform the merge operation which equates to the 3 Merge the data with the document > Merge step of the Mail Merge Helper dialog.

See also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.UPLOAD

TU.UPLOAD

TU.UPLOAD(*pcfilename*, *filehandle*, *id*, *options*, *description*, *status*)

Transfers data from a PC file into a host record. See TU.PC.UPLOAD on uploading multiple files.

When transferring binary data (for example, bitmaps, help files, and executables) on some host platforms, the data may become corrupt when you write it to the database. In operating environments that support directory type files, such as uniVerse and UniData, you use the B option only to transfer the data and store in in directory type files. If you wish to store data in hash files or your system doesn't support directory type files the you must hex encode the data.

pcfilename

[P] The name of the PC file containing the data to be transferred.

filehandle

[P] The handle to the host file.

id

[P] The host record id. If null, data is returned in this parameter.

options

[P] An array of characters indicating optional transfer details. Valid values are:

- B data is in Binary format
- H Host initiated transfer
- L Local connect, simple protocol
- R non-Resilient line, acknowledge every packet, do not stream
- S Spooler output (interchangeable with P)
- P spooler output (interchangeable with S)
- X hexadecimal output
- Z no status window
- 7 7 bit data link

In operating environments that support DIR type files, such as uniVerse and UniData, you use the B option only to transfer the data. You use the X option (hex encoding) when storing data in hash files.

description

[P] A string describing the data being transferred. description is displayed in the SBClient dialog box.

status

[R] The result of the transfer. Valid values are:

- 0 file has been transferred successfully
- 1 user abort
- 2 unable to open PC file
- 3 PC read error
- 4 communications error
- 5 retry limit exceeded

6 PC write error

7 file transfer not supported

10 unknown error

Sea also

TU.DOWNLOAD, TU.PC.DOWNLOAD, TU.PC.UPLOAD, TU.TO.EXCEL, TU.TO.EXCEL.GRAPH, TU.TO.MONOLOG, TU.TO.WORD, TU.TO.WORD.BOOKMARK, TU.TO.WORD.MERGE

Chapter 5 - MAPI Mail Integration

Note

The MAPI routines described below make use of the Microsoft MapiSess and MapiMess OCX's For these routines to work, the 32-bit MAPI DLLs need to be installed properly and Mail must be installed. For example, on Windows 95, you must install Mail during the operating system setup, or install it separately from the control panel to correctly use MAPI functions. SBClient installs and registers the OCX's, however the user must still setup a MAPI profile on the users PC.

These subroutines are used for sending and receiving email. The email program used is Microsoft Mail, which needs to be installed on your PC before you can use these subroutines. MAPI mail subroutines are:

- TU.MAPI.ADDRESSBOOK Displays the MAPI address-book.
- TU.MAPI.DELETE. Deletes the specified message from your inbox..
- TU.MAPI.FORWARD. Forwards mail to specified address and so on.
- TU.MAPI.GETMAIL. Gets mail or mail headers from the Inbox.
- TU.MAPI.GETMESSAGE Retrieves the specified message (text) from your inbox.
- TU.MAPI.LOAD. Instantiates a Mapi session.
- TU.MAPI.REPLY. Replies to mail represented by MSGNO and sends to specified address and so on.
- TU.MAPI.REPLYTOALL. Replies to all addresses on the mail represented by MSGNO.

- TU.MAPI.SENDMAIL. Sends mail via MAPI.
- TU.MAPI.TERMINATE. Terminates a Mapi session and should be called after completing MAPI usage

MAPI Mail API

TU.MAPI.ADDRESSBOOK

TU.MAPI.ADDRESSBOOK(*Mapihandle*, *Options*, *Error*)

Displays the MAPI address-book. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle of the MAPI session that was returned from TU.MAPI.LOAD.

Options

[P] Reserved for future use.

Error

[R] A non zero error codes indicates a failure.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.DELETE

TU.MAPI.DELETE(*Mapihandle*, *Msgno*, *Options*, *Error*)

Deletes the specified message from your inbox. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle of the MAPI session that was returned from TU.MAPI.LOAD.

Msgno

[P] The message number to be deleted. This corresponds to the list that is returned by TU.MAPI.GETMAIL and refers to the attribute in the list that contains the message to be deleted.

Options

[P] Reserved for future use.

Error

[R] A non zero error codes indicates a failure.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.FORWARD

TU.MAPI.FORWARD(*Mapihandle*, *Address*, *CC*, *Subject*, *Msgtext*, *Attachment*, *Msgno*, *Options*, *error*)

Forwards mail to specified address and so on. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle to the MAPI session that was returned by TU.MAPI.LOAD.

Address

[P] A SVM delimited list of primary email addresses.

CC

[P] A SVM delimited list of cc email addresses.

Subject

[P] The email subject heading.

Msgtext

[P] The actual email.

Attachment

[P] A SVM delimited list of attachments with a full path name.

Msgno

[P] The message number to be forwarded. This corresponds to the list that is returned by TU.MAPI.GETMAIL and refers to the attribute in the list that contains the message to be forwarded.

Options

[P] Options are:

R Receipt requested.

Error

[R] A non zero error codes indicates a failure.

Note

Any parameter that is specified here will take precedence over the original mails parameters. The letters ‘FWD' will be placed in the mail and the MSGTEXT specified in this function will be concatenated to the original message.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.GETMAIL

TU.MAPI.GETMAIL(*Mapihandle*, *Msgtext*, *Options*, *Error*)

Gets mail or mail headers from the Inbox. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle to the MAPI session that was returned by TU.MAPI.LOAD.

Options

[P] Options are

T Return the mail text as well as headers.

D Delete the message after returning.

U Return unread messages only.

Msgtext

[R] The email headers. To get all the mail messages as well as the headers, a 'T' option needs to be specified in the options. Options are:

1 Original display name

2 Message subject

3 Date that the message was received by the mail server

4 If the T option has been specified in the options then this value will be the actual text of the mail

5 E-mail address of the sender.

Error

[R] A non zero error codes indicates a failure.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.GETMESSAGE

TU.MAPI.GETMESSAGE(*Mapihandle*, *Msgno*, *Msgtext*, *Options*, *Error*)

Retrieves the specified message (text) from your inbox. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle of the MAPI session that was returned from TU.MAPI.LOAD.

Msgno

[P] The message number to be retrieved. This corresponds to the list that is returned by TU.MAPI.GETMAIL and refers to the attribute in the list that contains the message to be retrieved.

Options

[P] Options are

D Delete the message after retrieval.

Msgtext

[R] The text of the specified message.

Error

[R] A non zero error codes indicates a failure.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.TERMINATE

TU.MAPI.LOAD

TU.MAPI.LOAD(*Profilename*, *Mapihandle*, *Options*, *Error*)

Instantiates a Mapi session. This must be called before calling any Mapi functions.

Profilename

[P] The name of predefined Mapi profile. If set to 0, then the user is prompted for a profile.

Options

[P] Reserved for future use.

Mapihandle

[R] The handle of the MAPI session that should be passed to any subsequent call to a Mapi function.

Error

[R] A non zero error codes indicates a failure.

Note

The MAPI routines make use of the Microsoft MapiSess and MapiMess OCX's For these routines to work, the 32-bit MAPI DLLs need to be installed properly and Mail must be installed. For example, on Windows 95, you must install Mail during the operating system setup, or install it separately from the control panel to correctly use MAPI functions. SBClient installs and registers the OCX's, however the user must still setup a MAPI profile on the users PC.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.REPLY

TU.MAPI.REPLY(*Mapihandle*, *Address*, *CC*, *Subject*, *Msgtext*, *Attachment*, *Msgno*, *Options*, *Error*)

Replies to mail represented by MSGNO and sends to specified address and so on. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle to the MAPI session that was returned by TU.MAPI.LOAD.

Address

[P] A SVM delimited list of primary email addresses.

CC

[P] A SVM delimited list of cc email addresses.

Subject

[P] The email subject heading.

Msgtext

[P] The actual email.

Attachment

[P] A SVM delimited list of attachments with a full path name.

Msgno

[P] The message number to be replied to. This corresponds to the list that is returned by TU.MAPI.GETMAIL and refers to the attribute in the list that contains the message to be replied to.

Options

[P] Options are:

R Receipt requested.

Error

[R] A non zero error codes indicates a failure.

Note

Any parameter that is specified here will take precedence over the original mails parameters. The letters ‘RE' will be placed in the mail and the MSGTEXT specified in this function will be concatenated to the original message.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.REPLYTOALL

TU.MAPI.REPLYTOALL(*Mapihandle*, *Address*, *CC*, *Subject*, *Msgtext*, *Attachment*, *Msgno*, *options*, *error*)

Replies to all addresses on the mail represented by MSGNO. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle to the MAPI session that was returned by TU.MAPI.LOAD.

Address

[P] A SVM delimited list of primary email addresses.

CC

[P] A SVM delimited list of cc email addresses.

Subject

[P] The email subject heading.

Msgtext

[P] The actual email.

Attachment

[P] A SVM delimited list of attachments with a full path name.

Msgno

[P] The message number to be replied to. This corresponds to the list that is returned by TU.MAPI.GETMAIL and refers to the attribute in the list that contains the message to be replied to.

Options

[P] Options are:

R Receipt requested.

Error

[R] A non zero error codes indicates a failure.

Note

Any parameter that is specified here will take precedence over the original mails parameters. The letters 'RE will be placed in the mail and the MSGTEXT specified in this function will be concatenated to the original message.

See also

TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.SENDMAIL

TU.MAPI.SENDMAIL(*mapihandle*, *address*, *cc*, *subject*, *msgtext*, *attachment*, *options*, *error*)

TU.MAPI.SENDMAIL Sends mail via MAPI. Before this can be called, TU.MAPI.LOAD needs to be called.

Mapihandle

[P] The handle to the MAPI session that was returned by TU.MAPI.LOAD.

Address

[P] A SVM delimited list of primary email addresses.

CC

[P] A SVM delimited list of cc email addresses.

Subject

[P] The email subject heading.

Msgtext

[P] The actual email.

Attachment

[P] A SVM delimited list of attachments with a full path name

Options

[P] Options are:

R Receipt requested.

Error

[R] A non zero error codes indicates a failure.

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE, TU.MAPI.TERMINATE

TU.MAPI.TERMINATE

TU.MAPI.TERMINATE(*mapihandle*, *options*, *error*)

Terminates a Mapi session and should be called after completing MAPI usage.

Mapihandle

[P] The handle of the MAPI session that was returned from TU.MAPI.LOAD.

Options

[P] Reserved for future use.

Error

[R] A non zero error codes indicates a failure.

Note

After sending mail with TU.MAPI.SEND.MAIL, TU.MAPI.REPLY, TU.MAPI.REPLYTOALL or TU.MAPI.FORWARD, you should give the MAPI routines time to work before calling this sub-routine or your mail may not be sent successfully (3 to 5 seconds is usually safe)

See also

TU.MAPI.REPLYTOALL, TU.MAPI.FORWARD, TU.MAPI.REPLY, TU.MAPI.SENDMAIL, TU.MAPI.GETMAIL, TU.MAPI.LOAD, TU.MAPI.DELETE, TU.MAPI.ADDRESSBOOK, TU.MAPI.GETMESSAGE

Chapter 6 - ODBC Connectivity

These subroutines provide SQL connectivity to Microsoft ODBC32 drivers. Consult Microsoft ODBC32 help or the Microsoft ODBC Programmer's Reference Guide for further information regarding these drivers. To set up the drivers, go to the Windows Control Panel and click the ODBC32 icon.

- TU.SQL.CONNECT . Connects to a database.
- TU.SQL.DISCONNECT . Disconnects from a database.
- TU.SQL.EXEC . Executes an SQL statement on a database.
- TU.SQL.MAKEDICT . Creates host database dictionary records from the schema of an SQL table.
- TU.SQL.READ . Fetches the next record from a previous TU.SQL.EXEC call.

ODBC API

TU.SQL.Connection

TU.SQL.CONNECT(*connectparams*, *dbhandle*, *status*)

Connects you to a foreign database. You must connect to the database before you query it. You may connect to more than one database concurrently (by calling the subroutine again with the new parameters – a new dbhandle will be returned).

connectparams

[P] Parameters to connect to the database in the format:

token=value[;token=value…]

Valid tokens are:

- DRV. Name of datasource, defined in the ODBC32 configuration.
- UID. User id or name
- PWD. Password

dbhandle

[R] The handle returned for this database connection.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Program Example

```
CALL TU.SQL.CONNECT("DRV=QEDBF", DB.HANDLE, STATUS)
```

See also

TU.SQL.DISCONNECT, TU.SQL.EXEC, TU.SQL.MAKEDICT, TU.SQL.READ

TU.SQL.DISCONNECT

TU.SQL.DISCONNECT(*dbhandle*, *status*)

Disconnects from a database.

dbhandle

[P] The database handle returned from TU.SQL.CONNECT.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.SQL.CONNECT, TU.SQL.EXEC, TU.SQL.MAKEDICT, TU.SQL.READ

TU.SQL.EXEC

TU.SQL.EXEC(*dbhandle*, *sqlstat*, *sqlhandle*, *status*)

Executes any SQL statement supported by the target database.

dbhandle

[P] The database handle returned from TU.SQL.CONNECT.

sqlstat

[P] Valid SQL statement for the target database.

sqlhandle

[R] The handle returned from the SQL statement, uniquely identifying this specific statement.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

If the SQL statement is a SELECT statement, it does not return any data immediately. Instead, it returns a handle. This handle is used by the SQLREAD statement to fetch data, one record at a time.

Program Example

```
CALL TU.SQL.CONNECT('DRV=QEDBF', DB.HANDLE, STATUS)
IF STATUS # 0 PRINT 'ERROR';STOP
PRINT 'Enter SQL Table Name ':
INPUT TABLE.NAME
SQLSTAT = 'SELECT * FROM ':TABLE.NAME

CALL TU.SQL.EXEC(DB.HANDLE, SQLSTAT, SQL.HNDLE, STATUS)
```

See also

TU.SQL.CONNECT, TU.SQL.DISCONNECT, TU.SQL.MAKEDICT, TU.SQL.READ

TU.SQL.MAKEDICT

TU.SQL.MAKEDICT(*dbhandle*, *dbtablename*, *dictfile*, *status*)

Creates host database dictionary records from the schema of the SQL table.

dbhandle

[P] The database handle returned from TU.SQL.CONNECT.

dbtablename

[P] The name of the SQL table you wish to generate dictionary records from.

dictfile

[P] The name of the host dictionary file to contain the generated dictionary records.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

In the following example this statement retrieves the name, length and data type from each column in the target database table and creates dictionary records for each of these fields:

Program Example

```
CALL TU.SQL.CONNECT('DRV=QEDBF', DB.HANDLE, STATUS)
IF STATUS # 0 PRINT 'ERROR';STOP

CALL TU.SQL.MAKEDICT(DB.HANDLE, 'EMP.DBF', 'EMPLOYERS', STATUS)
```

See also

TU.SQL.CONNECT, TU.SQL.DISCONNECT, TU.SQL.EXEC, TU.SQL.READ

TU.SQL.READ

TU.SQL.READ(*sqlhandle*, *record*, *status*)

Fetches the next record from a previously executed SQL statement.

sqlhandle

[P] The handle returned from a previous TU.SQL.EXEC call.

record

[R] The variable to contain the contents of the read. The format of the data returned is as follows: Each field of the record being read is converted to a string. These strings are copied into a dynamically allocated array (separated by attribute marks), which is returned in record. No data conversion is performed. You decide how the data is processed.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Program Example

```
CALL TU.SQL.CONNECT('DRV=QEDBF', DB.HANDLE, STATUS)
IF STATUS # 0 PRINT 'ERROR';STOP
SQLSTAT = 'SELECT NAME BAL FROM CUSTOMER.DBF'
CALL TU.SQL.EXEC(DB.HANDLE, SQLSTAT, SQL.HNDLE,STATUS)
LOOP WHILE STATUS = 0 DO
       CALL TU.SQL.READ(SQL.HNDLE, RECORD, STATUS)
       * do something with RECORD
       .......

REPEAT
```

See also

TU.SQL.CONNECT, TU.SQL.DISCONNECT, TU.SQL.EXEC, TU.SQL.MAKEDICT

Chapter 7 - PC File Handling

These subroutines are used to manipulate PC files from SBClient.

PC file handling subroutines are:

- TU.CHECK.DIRECTORY . Checks whether a particular directory exists.
- TU.CHECK.FILE . Checks whether a particular file exists.
- TU.CREATE.DIRECTORY . Creates a directory.
- TU.CREATE.FILE . Creates a file.
- TU.DELETE.DIRECTORY . Deletes a directory.
- TU.DELETE.FILE . Deletes a file.

PC File Handling API

TU.CHECK.DIRECTORY

TU.CHECK.DIRECTORY(*directoryname*, *status*)

Returns a flag indicating the existence of the named directory.

directoryname

[P] The full pathname of the directory to be checked.

status

[R] Indicates whether the directory exists. Valid values are:

0 directory exists

1 directory does not exist

Example

The following example checks that the directory c:\tmp\tu exists, creates it if it does not, and then deletes it.

Program Example

```
DIRECTORYNAME = 'C:\TMP\TU'
CALL TU.CHECK.DIRECTORY(DIRECTORYNAME, STATUS)
IF STATUS = 0 THEN
       CALL TU.DELETE.DIRECTORY(DIRECTORYNAME, STATUS)
END
CALL TU.CREATE.DIRECTORY(DIRECTORYNAME, STATUS)
IF STATUS THEN
       PRINT 'CREATE FAILED - STATUS=':STATUS
END
CALL TU.DELETE.DIRECTORY(DIRECTORYNAME, STATUS)
IF STATUS THEN
       PRINT 'DELETE FAILED - STATUS=':STATUS
```

```
END
```

See also

TU.CHECK.FILE, TU.CREATE.DIRECTORY, TU.CREATE.FILE, TU.DELETE.DIRECTORY, TU.DELETE.FILE

TU.CHECK.FILE

TU.CHECK.FILE(*filename*, *status*)

Returns a flag indicating the existence of the file.

filename

[P] The full pathname of the file to be checked.

status

[R] Indicates whether the file exists. Valid values are:

0 file exists

1 file does not exist

See also

TU.CHECK.DIRECTORY, TU.CREATE.DIRECTORY, TU.CREATE.FILE, TU.DELETE.DIRECTORY, TU.DELETE.FILE

TU.CREATE.DIRECTORY

TU.CREATE.DIRECTORY(*directoryname*, *status*)

Creates the named directory and returns a flag indicating the success or failure of the operation.

directoryname

[P] The name of the directory to be created.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

This API will return success if you attempt to delete a non-existant directory.

See also

TU.CHECK.DIRECTORY, TU.CHECK.FILE, TU.CREATE.FILE, TU.DELETE.DIRECTORY, TU.DELETE.FILE

TU.CREATE.FILE

TU.CREATE.FILE(*filename*, *status*)

Creates the named file and returns a flag indicating the success or failure of the operation.

filename

[P] The full pathname of the file to be created.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.CHECK.DIRECTORY, TU.CHECK.FILE, TU.CREATE.DIRECTORY, TU.DELETE.DIRECTORY, TU.DELETE.FILE

TU.DELETE.DIRECTORY

TU.DELETE.DIRECTORY(*directoryname*, *status*)

Deletes the named directory and returns a flag indicating the success or failure of the operation. This subroutine will fail if the directory contains files.

directoryname

[P] The full pathname of the directory to be deleted.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.CHECK.DIRECTORY, TU.CHECK.FILE, TU.CREATE.DIRECTORY, TU.CREATE.FILE, TU.DELETE.FILE

TU.DELETE.FILE

TU.DELETE.FILE(*filename*, *status*)

Deletes the named file and returns a flag indicating the success or failure of the operation.

filename

[P] The name of the file to be deleted.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

This API will return success if you attempt to delete a non-existant file.

See also

TU.CHECK.DIRECTORY, TU.CHECK.FILE, TU.CREATE.DIRECTORY, TU.CREATE.FILE, TU.DELETE.DIRECTORY

Chapter 8 - PC Printer Control

These subroutines are used to query the status of Windows' printers.

PC printer control subroutines are:

- TU.GET.PRINTER.LIST. Lists the printers defined in the Windows environment.
- TU.GET.PRINTER.ROWS. Returns the number of rows that can printed on a printer.
- TU.QUERY.PRINT.OPTIONS. Is used to get parameters normally defined in the Print Options window, off of the File menu.
- TU.SELECT.PRINTER. Selects a Windows printer.
- TU.SEND.TO.PRINTER . Outputs data to the currently selected Windows printer.
- TU.SET.PRINT.OPTIONS. Is used to set parameters normally defined in the Print Options window, off of the File menu.

PC Printer Control API

TU.GET.PRINTER.LIST

TU.GET.PRINTER.LIST(*printerlist*, *status*)

Lists the installed Windows printers.

printerlist

[P] A tilde (~) delimited list of printer definitions. Right braces (}) separate fields in the definition.

An example of printerlist is:

HP LaserJet Series I}winspol}LPT1:~\\NP1ECEC4\HPLJ4-2}winspol}Ne02:

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.SELECT.PRINTER, TU.SEND.TO.PRINTER

TU.GET.PRINTER.ROWS

TU.GET.PRINTER.ROWS(*cols*, *rows*, *status*)

Returns the number of rows that can be printed on a page based on the number of columns, the selected printer and the chosen font.

cols

[P] The number of columns to be printed.

rows

[P] The number of rows that can be printed on a page.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.QUERY.PRINT.OPTIONS

TU.QUERY.PRINT.OPTIONS(*option*, *status*)

TU.QUERY.PRINTER.OPTIONS is used to get parameters normally defined in the Print Options window, off of the File menu.

Option

[R] This will contain the following characters if the corresponding option is set:

P Use Windows Print Drivers.

Q Use SBClient Print Font.

R Strip Box Characters.

T Use Condensed Mode.

U OEM to ANSI Conversion.

For example, if Use Windows Print Drivers and Use condensed Mode are set, Option will return 'PT'

Status

[R] A non zero error codes indicates a failure.

See also

TU.GET.PRINTER.LIST, TU.GET.PRINTER.ROWS, TU.SELECT.PRINTER, TU.SEND.TO.PRINTER, TU.SET.PRINT.OPTIONS

TU.SELECT.PRINTER

TU.SELECT.PRINTER(*printername*, *status*)

Selects a printer. Data can then be sent to this printer using TU.SEND.TO.PRINTER.

printername

[P] The printer, defined in the TU.GET.PRINTER.LIST option printerlist, to be used for future printing.

For example:

If printerlist is HP LaserJet Series I}winspol}LPT1:~\\NP1ECEC4\HPLJ4-2}winspol}Ne02:, printername is the first value from printerlist, that is, HP LaserJet Series I or \\NP1ECEC4\HPLJ4-2.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.GET.PRINTER.LIST, TU.SEND.TO.PRINTER

TU.SEND.TO.PRINTER

TU.SEND.TO.PRINTER(*data*, *options*, *status*)

Outputs data to the currently selected printer.

data

[P] The data (string) to be printed.

options

[P] Reserved for future use.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.GET.PRINTER.LIST, TU.SELECT.PRINTER

TU.SET.PRINT.OPTIONS

TU.SET.PRINT.OPTIONS(*Option*, *State*, *Status*)

TU.SET.PRINTER.OPTIONS is used to set parameters normally defined in the Print Options window, off of the File menu.

Option

[P] A character representing one of the following options to set:

P Use Windows Print Drivers.

Q Use SBClient Print Font.

R Strip Box Characters.

T Use Condensed Mode.

U OEM to ANSI Conversion.

State

[P] The value of the option. This may be one of the following:

1 On.

0 Off.

Status

[R] Zero denotes success, and non zero denotes failure.

See also

TU.GET.PRINTER.LIST, TU.GET.PRINTER.ROWS, TU.SELECT.PRINTER, TU.SEND.TO.PRINTER, TU.QUERY.PRINT.OPTIONS

TU.SELECT.PRINTER

TU.SELECT.PRINTER(*printername*, *status*)

Selects a printer. Data can then be sent to this printer using TU.SEND.TO.PRINTER.

printername

[P] The printer, defined in the TU.GET.PRINTER.LIST option printerlist, to be used for future printing.

For example:

If printerlist is HP LaserJet Series I}winspol}LPT1:~\\NP1ECEC4\HPLJ4-2}winspol}Ne02:, printername is the first value from printerlist, that is, HP LaserJet Series I or \\NP1ECEC4\HPLJ4-2.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.GET.PRINTER.LIST, TU.SEND.TO.PRINTER

TU.SEND.TO.PRINTER

TU.SEND.TO.PRINTER(*data*, *options*, *status*)

Outputs data to the currently selected printer.

data

[P] The data (string) to be printed.

options

[P] Reserved for future use.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.GET.PRINTER.LIST, TU.SELECT.PRINTER

Chapter 9 - Windows Process Control

These subroutines control the launching and closing of Windows applications.

Windows process control subroutines are:

- TU.CHECK.APP . Determines whether a Windows application is currently running.
- TU.CLOSE.APP . Closes a Windows application.
- TU.LAUNCH.APP . Launches a DOS or Windows application.

Due to the architecture of the Windows operating system it is not possible to find 16 bit applications in a 32 bit environment.

Windows Process Control API

TU.CHECK.APP

TU.CHECK.APP(*applicationname*, *appstatus*)

Determines whether an application is currently running.

SBClient only supports 32-bit applications.

applicationname

[P] The name of the application, e.g. winword.exe.

appstatus

[R] Indicates whether the application is running. Returns:

0 application running

12 application not running

See also

TU.CLOSE.APP, TU.LAUNCH.APP

TU.CLOSE.APP

TU.CLOSE.APP(*applicationname*, *status*)

Closes the named application.

SBClient only supports 32-bit applications.

applicationname

[P] The name of the application to be closed.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.CHECK.APP, TU.LAUNCH.APP

TU.LAUNCH.APP

TU.LAUNCH.APP(*applicationname*, *option*, *status*)

This routine launches 32 bit Windows applications. SBClient will first look in the registry to see if it can find the path to the executable. If SBClient cannot find the path in the registry SBClient then checks the environment PATH variable and will try to use this. Finally if SBClient still cannot launch the application it will prompt the user for this information via a dialog. You will be able to use the Browse button on this dialog to navigate to the application on the appropriate hard drive. SBClient will then store this information in the registry so that you won't have to do this again.

In the case of DOS commands and 16 bit application (which will all use the same DOS virtual machine) you should use TU.EXECUTE.SHELL instead of TU.LAUNCH.APP.

applicationname

[P] The name of the application e.g. "winword.exe".

option

[P] The state in which the application is to be launched, maximized, minimized or open covering a portion of the screen.

Use these predefined constants in TU.API.H:

```
EQU APP.HIDE TO 0
EQU APP.SHOWNORMAL TO 1
EQU APP.SHOWMINIMIZED TO 2
EQU APP.SHOWMAXIMIZED TO 3
EQU APP.SHOWNOACTIVATE TO 4
EQU APP.SHOW TO 5
EQU APP.MINIMIZE TO 6
EQU APP.SHOWINNOACTIVE TO 7
EQU APP.SHOWNA TO 8
EQU APP.RESTORE TO 9
```

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

The following example launches a Windows application. It first checks SBClient's version details, then launches Word. Once it has launched Word, it checks Word to see whether it is running. If it is running, it closes Word, then checks to see if Word is closed.

Program Example

```
CALL TU.GET.VERSION(VERSION, STATUS)
PRINT VERSION
APPLICATION = 'WINWORD.EXE'
OPTIONS = APP.SHOWMINNOACTIVE
CALL TU.LAUNCH.APP(APPLICATION, OPTIONS, STATUS)
SLEEP 2
CALL TU.CHECK.APP(APPLICATION, STATUS)
PRINT 'CHECK STATUS=':STATUS
IF NOT(STATUS) THEN
       CALL TU.CLOSE.APP(APPLICATION, STATUS)
       PRINT 'CLOSE STATUS=':STATUS
       CALL TU.CHECK.APP(APPLICATION, STATUS)
       PRINT 'CHECK STATUS=':STATUS
END
```

See also

TU.CHECK.APP, TU.CLOSE.APP, TU.GET.VERSION, TU.EXECUTE.SHELL

Chapter 10 - Misc Windows Integration

These subroutines provide additional Windows Integration functionality.

Miscellaneous Windows Integration subroutines are:

- TU.EXECUTE.SHELL. Is used to run a shell command on the client's operating system.
- TU.CLIENT.GETENV. Is used to return the contents of the specified environment variable.
- TU.CLIENT.SETENV . Is used to set the contents of the specified environment variable.
- TU.GET.VERSION . Gets the SBClient and SBClient host library version.
- TU.IMAGE . Displays a bitmap image.
- TU.MACRO. Invokes SBClient's macro facility to control Windows applications and SBClient scripts.
- TU.RUN.MULTIMEDIA. Is used to run a Multimedia Command Interface (MCI) string.
- TU.RUN.SBO.COMMAND . Runs an SBDesktop command.
- TCL.SBCVERSION. This is a TCL command that will display the version information of SBClient including the client version and the host platform and version.
- TU.SESSION.CLOSE . Closes a session without displaying a dialog.
- TU.VIDEO . Plays a video using the SBClient video player.

Miscellaneous Windows Integration APIs

TU.EXECUTE.SHELL

TU.EXECUTE.SHELL(*Shell.Command*, *Option*)

This is used to run a shell command on the client's operating system or a 16 bit application. When specifying the shell command it must be located in your PATH variable or alternatively you must place the full path to the command.

Therefore to execute Notepad you would specify a path similar to:

"C:\Windows\Notepad.exe"

Applications such as Microsoft Word should be launched using TU.LAUNCH.APP .

All shell commands are executed in the DOS virtual machine. It is therefore not possible to use routines like TU.CHECK.APP or TU.CLOSE.APP on commands/applications which run in the virtual machine. If this functionality is required then you must use TU.LAUNCH.APP. This routine will create processes for each application (must be 32 bit as any 16 bit application will run in the DOS virtual machine) which means that you can check to see if it is running or kill it from Basic.

Shell.Command

[P] A string containing the command and arguments to be executed.

The format for the command string is:

[+ | & | -] [n] command

where:

+ indicates that a Windows 16 bit application command is to be run asynchronously (that is the command is started, and SBClient waits for it to finish).

& indicates that a Windows 16 bit application command is to be run synchronously (that is the command is started , but control is returned to SBClient, without waiting for the command to complete).

- indicates that a DOS command is to be run (synchronously). This is the default.

n only applies when running a Windows 16 bit application command and has a default value of 3. This may be:

0 – runs program, but hides it.

1 – runs program in normal show state, and restores from maximised or minimised.

2 – runs minimised.

3 – runs maximised.

4 – runs in current state , but does not activate.

5 – runs in current state.

6 – runs minimised, and activates top window in system list.

7 – runs minimised, but does not activate.

8 – runs in current state, and leaves currently active window active.

9 – same as 1.

Examples

```
TU.EXECUTE.SHELL( "-DIR /P","P");
```
runs the DOS DIR command

```
TU.EXECUTE.SHELL("-OCOPY test.bmp tmp.bmp","");
```
runs the DOS COPY command invisibly.

Option

[P] A string containing additional arguments to be used during execution of the shell command

[P] option – do not parse the shell command. This option will disable the parsing and will not convert '/' to '\'.

See also

TU.LAUNCH.APP

TU.CLIENT.GETENV

TU.CLIENT.GETENV(*variable*, *returnstring*)

TU.CLIENT.GETENV is used to return the contents of the specified environment variable. If the call fails then an empty string is returned.

Variable

[P] The environment variable to query (for example, “TEMP”).

ReturnString

[R] String containing the value of the environment variable. An empty string is returned if the call is unsuccessful.

TU.CLIENT.SETENV

TU.CLIENT.SETENV(*Env*, *Set.str*)

This function will set the contents of the specified environment variable.

Env

[P] This is the variable’s name.

SetStr

[P] This is the string which the variable is set to.

Note that some of these variables are system specific to SBClient (i.e. SBOPATH, SBTUPATH, SBODB, SBOCLIENT, SBOPRINTPATH) and cannot be set by developers. Some user defined variables used by SBClient will not be enacted on until the client has been restarted. All environment variables are set and then written to the sbopen.ini file.

TU.GET.VERSION

TU.GET.VERSION(*version*, *error*)

Returns the version of SBClient and the SBClient host library in a dynamic array.

version

[R] The first attribute is the SBClient version. The second attribute is the host library version.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

The following example returns the SBClient version information.

Program Example

```
CALL TU.GET.VERSION(VERSION, STATUS)
PRINT VERSION
```

TU.IMAGE

TU.IMAGE(*bitmapfile*, *col*, *row*, *scale*)

Displays an image.

bitmapfile

[P] The name of the file in which the bitmap is stored.

col

[P] The horizontal position of the left side of the bitmap.

row

[P] The vertical position of the top of the bitmap.

scale

[P] The size at which the bitmap is displayed. A value of 100 displays the bitmap at its original size. A value of 0 makes the bitmap disappear.

Note

The underlying escape sequence sent to SBClient is described in Escape Sequences.

TU.MACRO

TU.MACRO(*type*, *macro*, *value*)

Invokes SBClient's macro facility to control Windows applications and SBClient scripts.

type

[P] Can be one of:

- APP.CHECK
- APP.CLOSE
- APP.FIND
- APP.ICON
- APP.LAUNCH
- APP.LOCATION
- APP.MACRO
- APP.SHOWWIND
- APP.ZOOM
- APP.SHELL

macro

[P] The macro itself.

value

[R] Any returned value (if applicable).

Note

Developers will need to include the TUBP record TU.API.H to access the above types.

See also

Macro Syntax for more information on macros.

TU.RUN.MULTIMEDIA

TU.RUN.MULTIMEDIA(*mci.string*, *returnstring*, *error*)

TU.RUN.MULTIMEDIA is used to run a Multimedia Command Interface (MCI) string. Please consult the MCI Command String reference for more details.

mcistring

[P] A MCI string but with the following option extension following a + token:

+C#	convert # from columns to x pixels.
+D#	convert # from character depth to pixel.
+HW	substitute hCeo window handle.
+HD	substitute hCeo device context handle.
+I {# # # #}	invalidate rectangle.
+I {# # # #}	invalidate rectangle.
+R#	convert # from rows to y pixels.
+W#	convert # from character width to pixel.

Return.String

[R] String containing error or status text.

Error

[R] A no zero error codes indicates a failure.

See also

The MCI help file.

TU.RUN.SBO.COMMAND

TU.RUN.SBO.COMMAND(*commandline*, *capture*)

Runs an SBDesktop command.

commandline

[P] The SBDesktop command to run. The commandline may consist of a command followed by parameters, as if it was typed from the SBDesktop command line. The command to be run must be cataloged in the VOC of the SBClient account.

capture

in/[R] Returns the captured output, but only if an input capture is set to true.

[P] (passed or in) values:

0 Don't capture the output.

1 Capture and return output.

2 Capture and return output, converts all AM, VM, SVM to tab and ^\.

3 Capture output and return nothing.

[R] (returned or out) values:

Represents the captured data if modes 1 and 2 are used.

TCL.SBCVERSION

TCL.SBCVERSION

TCL.SBCVERSION is a TCL command that will display the version information of SBClient including the client version and the host platform and version.

TU.SESSION.CLOSE

TU.SESSION.CLOSE(*sessionname*)

Closes an SBClient session without any dialog.

sessionname

[P] The name of the session to close. If null, the current active session is closed.

TU.VIDEO

TU.VIDEO(*videofile*)

Plays a video using the SBClient video player.

videofile

[P] The name of the Windows .AVI file to play. This may be left null, in which case a file may be selected from the video player Load button.

Note

You must have the Microsoft 'Video For Windows' driver loaded to be able to play .AVI files.

PART III: ADVANCED WINDOWS INTEGRATION

This section of the manual introduces you to overall concepts built into SBClient.

Chapter 11 - Generic Object Manipulation

Generic Object Manipulation APIs

ROC.CREATE

ROC.CREATE(*classname*, *attributes*, *values*, *handle*, *error*)

This is used to create any Data/C++ COM or ActiveX object and set any initial attributes for that instance. Upon successful creation the handle for the newly created object is returned in the handle parameter.

classname

[P] The class type of the object you wish to create. For example: formclass, textclass, pbclass, fredsclass and so on.

attributes

[P] Any attributes needed or desired to be set at create time. For example: dimensions, foreground, string and so on.

values

[P] The associated values for the attribute list specified above. For example: !00:VM:100, YELLOW, “Hello World” and so on.

handle

[R] This parameter is updated upon successful creation with the new object’s handle. This handle is used to reference subsequent object actions.

error

[R] A no zero error code indicates a failure.

See also

ROC.DESTROY

ROC.DESTROY

ROC.DESTROY(*objectname*, *error*)

This is used to destroy a Data/C++ COM or ActiveX object previously created with ROC.CREATE. Any children of this object will also be destroyed.

objectname

[P] The object name or object handle of the object you wish to destroy.

error

[R] A no zero error code indicates a failure.

See also

ROC.CREATE

ROC.GET

ROC.GET(*objectname*, *attributes*, *values*, *error*)

This is used to retrieve the current values of any attributes in a previously created Data/C++ COM or ActiveX object.

objectname

[P] The object name or object handle of the object you wish to interrogate.

attributes

[P] The attributes to be examine. For example: dimensions, foreground, string and so on. If multiple attributes are to be examined, then the ROC.US (ROC.US is defined in ROC.H) delimiter (usually CHAR(250)) must be used. In a multi-parameter attribute examination, for example, <Attrib, param1, param1>, then multi-value characters must be used to separate the attribute name and it's parameters.

When using third party OCXes an ANSI to OEM conversion may be required. Please consult the documentation for your OCX. If this conversion is required then you need to specify the attributes arguments as follows:

<attrib;X,param1,param2>

values

[R] The associated list of values returned after the call. For example: !00:VM:100 or YELLOW, “Hello World” and so on. Multiple get values use the ROC.US delimiter as described above.

error

[R] A no zero error code indicates a failure.

See also

ROC.SET, ROC.CREATE, ROC.DESTROY

ROC.GETHANDLE

ROC.GETHANDLE(*objectname*, *handle*, *error*)

This is used to return the numeric handle of a specified object name.

objectname

[P] The object name of the object you wish to get the handle of.

handle

[R] This parameter is updated with the object’s handle.

error

[R] A no zero error code indicates a failure.

See also

ROC.CREATE

ROC.SET

ROC.SET(*objectname*, *attributes*, *values*, *error*)

ROC.SET is used to set values of any attributes in any previously created Data/C++ COM or ActiveX object. Attributes can also be set at create time.

objectname

[P] The object name or object handle of the object you wish to examine.

attributes

[P] The attributes to be set. For example: dimensions, foreground, string and so on. If multiple attributes are to be set, then the ROC.US (ROC.US is defined in ROC.H) delimiter (usually CHAR(250)) must be used. In a multi-parameter attribute set, for example, <Attrib, param1, param1>, then multi-value characters must be used to separate the attribute name and it's parameters.

When using third party OCXes an OEM to ANSI conversion may be required. Please consult your OCX documentation. If this conversion is required then you need to specify the attributes arguments as follows:

<attrib,X,param1,param2>

values

[P] The associated values for the attribute list specified above. For example: 100:VM:100, YELLOW, "Hello World" and so on. To separate multiple set values use then ROC.US delimiter as described above.

error

[R] A no zero error code indicates a failure.

See also

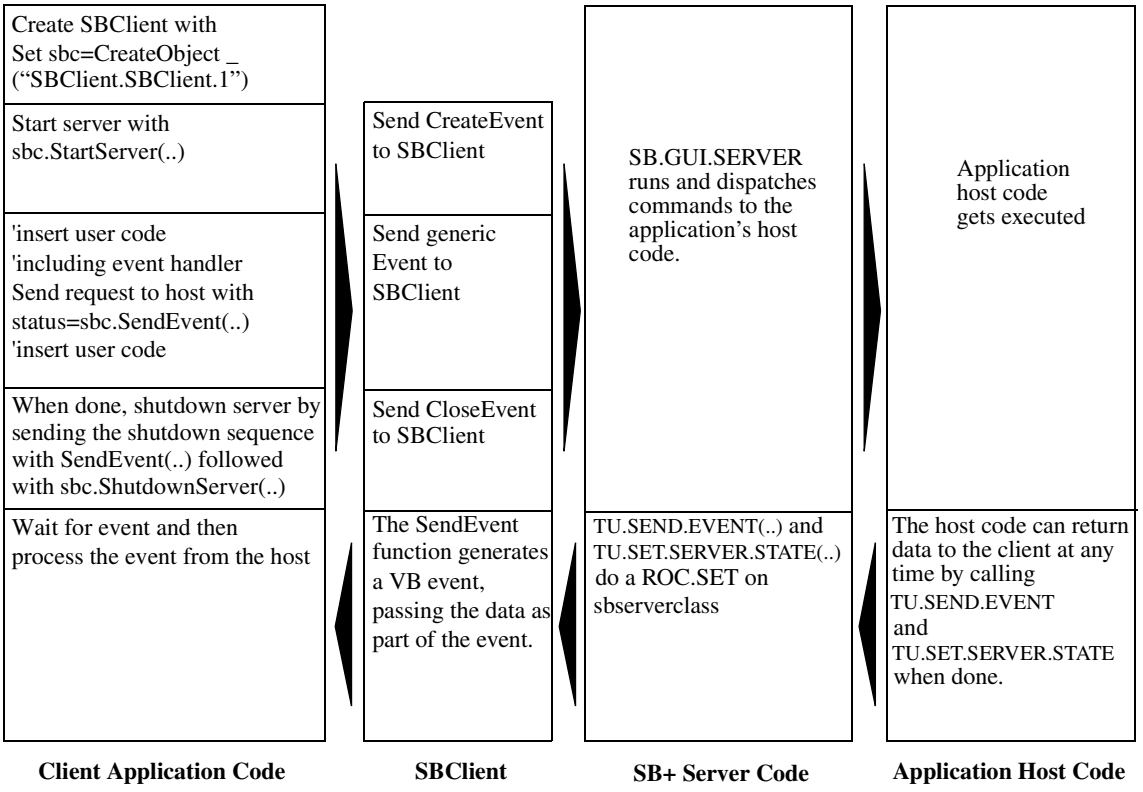
ROC.GET, ROC.CREATE, ROC.DESTROY

Chapter 12 - OLE Server Interface

It is now possible to control SB+ Server through SBClient while making the user interface of SBClient invisible. With the "Self-contained forms" functionality of SB+ Server activated, it is possible to activate SB+ Server commands and see the resultant SB+ Server processes without either the SB+ Server main background window (known as MainWin) or the SBClient user interface being visible. This provides developers with a great deal of flexibility in customising applications. There is a comprehensive overview of how to use the OLE Server Interface located in the samples directory of SBClient.

The code examples shown are, unless otherwise stated, Microsoft Visual Basic (VB) 5 based routines which represent a client application interacting with the SB OLE Server. The client application supplied is based on a Windows Explorer user interface which demonstrates how to access the various SB+ Server commands (via an "invisible" SBClient). The application uses a TreeView control. The VB example can be found on the SBClient product cd-rom in the samples area.

The following diagram illustrates the overall process of SB OLE Server communication.



SB OLE Server Components

In order to use the SB OLE Server functionality, interaction takes place through four components. These are:

- Client application

- SBClient
- SB+ Server
- Custom application (created using SB+ Server)

SB OLE Server - Host Interface

The following routines have been added to the standard SBClient HOST LIBRARY and are available to enable the host code (i.e. custom application created using SB+ Server) to communicate with the client code if required. TU.SEND.EVENT is to be used for generic user defined events while TU.SET.SERVER.STATE is to be used to inform the client code (and SBClient) the current state of the host.

SB OLE Server - SB+ Server Interface

SB+ Server provides an interface for internal processes, as well as a server which can dispatch events to user defined host code.

This enables client code to run any SB+ Server process on the host which can then return values to the client code. SB+ Server will handle most of the server communications, however it will still be possible for any user defined host code to send events to the client code.

Note

SB+ Server runs in GUI mode only.

To start the session the client code will run StartServer passing the name of a valid session configuration document (and its path) and Unix/NT, SB+ user login details (see ActiveX interface above). The session should have an appropriate script attached that will use the login info to log the user into Unix and SB+. A terminal type of TU.xxx.SERVER (TU.VT220.SERVER for example) will be used to tell SB+ that it must run in SERVER mode. Internally, the SB.LOGIN program will replace this terminal type with TU.xxx.GUI and invoke the SB.GUI.SERVER program.

SB.GUI.SERVER will establish the correct environment (Common block, Sysid), call TU.SET.SERVER.STATE with a READY status and then wait for a client generated event.

There are five possible events the SB+ Server can receive:

1. Execute Process (interactive)

The SB+ Server can be told to run an interactive process, for example input/output process, menu processes, help window displays etc. After the form etc is displayed, the SB+ Server will call TU.SET.SERVER.STATE to let the client code know that the server is ready for more interactive commands. At this point the program control will be in SB.GUI.INP (standard input routine), SB.MENU.GUI (menu selection) or SB.DISP.BOXTEXT (help display) i.e. any SB+ program capable of receiving and processing events.

2. Execute Process (non-interactive)

   This type of event will cause the process to be executed until completion before setting the Server state to READY. Some examples of processes to be invoked in this manner are CEO's, Access Reports, Select processes etc.

3. Get Menu Level (non-interactive)

   The client code will use this type of call to get the menu tree information. The server will return the menu options and process types to the client code (input, output, selection, shell process, etc).

4. Exit server (non-interactive)

   This event causes the SB+ Server to shut down

5. Modify Process (interactive)

   This is equivalent to typing /mp to modify a process

   The client code will need to specify one of the 5 types when calling a menu/process etc. The mode will be passed as part of the event_string in the SendEvent method. The following are syntax examples of the event_string parameter in the SendEvent() method when using Microsoft Visual Basic:

sbcom_process	Run a process in interactive mode: event_string = *session_handle* & ";sbcom_process;"& *process-name[,parameter][;processdata]*
sbcom_modal	Run a process in non interactive mode: event_string = *session_handle* & "; sbcom_modal;"&*process-name[,parameter][;processdata]*
sbcom_menu	Get a menu level in non interactive mode: event_string = *session_handle* & ";sbcom_menu;"& *menuname*
sbcom_exit	Shut down SB+ server: event_string = *session_handle* & ";sbcom_exit"
tree_modify	Call modify process: event_string = *session_handle* & ";tree_modify;" & *processname*

   The non-interactive processes and 'get menu level' type processes will be capable of passing the data back to the client.

To run user defined host code, the client code specifies a process name (and optional parameter) and how it is to be executed (interactively or non-interactively). The process could be one of 21 different process types including screens, reports, call to a Basic subroutines etc.

SBClient Specific

SBClient Events

EventServer()

EventServer(*event_type*, *event_src*, *event_data*)

event_type

Output This may be either SERVER_STATE, SERVER_EVENT, PROGRESS, SCRIPT_DATA.

SERVER_STATE

The SBClient object generates this type of event when the server identified by a session handle changes state. The new server state value is delivered with the event in the event_data parameter.

The state value can be one of the following:

`1` - READY

A READY state is returned when the host server program becomes ready to receive events. The host program can generate this event using the TU.SET.SERVER.STATE API with the state parameter set to 1 (see Host interface section).

`2` - BUSY

A BUSY state is returned when the host server program becomes busy and can not receive events. The host program can generate this event using the TU.SET.SERVER.STATE API with the state parameter set to 2 (see Host interface section).

`3` - SHUTDOWN

A SHUTDOWN state is returned when a session handle becomes invalid (e.g. after the CloseSession() call). The host program can generate this event using the TU.SET.SERVER.STATE API with the state parameter set to 3 (see Host interface section).

`4, 5, 6, 7`

As an approximate guide 4=25%, 5=50% etc, of logging in or logging out progress.

SERVER_EVENT

The host program can generate this type of event using the TU.SEND.EVENT API specifying an event string appropriate for the implemented protocol.

PROGRESS

A percentage value is returned which represents the percentage completion of an executing file transfer from the host to a client PC or vice-versa.

SCRIPT_DATA

This is used if the script contains a trigger response of [com(data)] where data is user specified. It can be used, for example, to respond to an invalid password string. The VB code can then put up a dialog to ask the user to reenter his password and then send the password with the following:

' Status = sbc.SendEvent(*session_handle*, "STRING", *newpassword*, val)

This [com_data(data)] macro actually enables the VB program to pick up every trigger specified in the script and to respond to them with sendEvent. In other words the entire script can be driven from the client program.

event_src

Output All events return the session handle in the parameter that identifies the session that this event was generated from.

event_data

Output This contains protocol-dependent event information.

SBClient Methods

The following methods are exposed by SBCom.dll to a calling application (i.e. an application that references the object containing the method). These examples all assume that you have previously created an instance of SBCom called sbc using the following line of Microsoft Visual Basic code:

```
Set sbc = CreateObject("SBClient.SBClient.1")
```

StartServer()

This method must be called to initialize all the communication channels and start the server before calling any other methods.

This method uses the following syntax:

status = *objectname*.StartServer(*session_id*, *parameters*, *timeout*, *session_handle*)

Program Example

```
Public WithEvents sbc As SBCOMLib.SBClient
Public SessionHandle As Variant
.
.
Status = sbc.StartServer(File1.Path & "\" & File1.filename, "un~" & uname _
& ";pw~" & pw, 10, SessionHandle)
```

status

Overview — Once a call to StartServer() has been made, the client application code should wait for the EventServer event with an event type of SERVER_STATE to determine when the server is ready. It should not rely on the status returned by StartServer() to determine whether the server is ready (see Return value description).

If status does not equal zero (0) then the number represents the error code.

Note

The client can display the progress of the login as the client will receive various server states that track the progess. (See the section titled “Events” that follows shortly).

Return — A zero (0) is returned if StartServer() is executed by SBClient. StartServer() returns immediately and does not wait for the server to be ready. A zero does not indicate that the server has started, rather it indicates that the command to start the server has been sent to the host successfully.

session_id

Input — This is the file name of a ".sbc" file which includes a script to login to SB+.

parameters

Overview — This Contains script replacement values for user ids and passwords (e.g. unix user_id and password, SB+ Server user_id and password). The parameters will be semicolon delimited.

Input — Each pair of parameters will be tilde (i.e. ~) delimited for example:

“un~” *usernamevariable*;”pw~” & *passwordvariable*.

In this case "un~" & *usernamevariable*;"pw~"& *passwordvariable* are the replacement variables. There are thus 2 replacement variables passed. In other words when un is encountered in a response to a trigger in a script, username-variable will be substituted. Similarly passwordvariable will be substituted for pw. There can be any number of replacement variables passed separated by semicolons. This means that if the script has [user_data("un")] as a response to a trigger, the script will substitute the value that was passed as username-variable.

If the variables are setup via the **Setup > Script** command as user defined script replacements,

Example — usernamevariable=mark

passwordvariable=mypassword

then the auto script learn will insert the [user_data("un")]' whenever the user types in the word "mark".

timeout

Overview — This specifies a default timeout value for this and calls to all other methods. Where a timeout value is specified in a call to a different method that timeout takes precedence for the call to that method.

Input — The timeout in this method refers only to the time taken for the client code method call to return from SBClient. The method will return as soon as SBClient has sent the request/data to the host, that is it will not wait until the host receives (or processes) the request/data.

session_handle

Output — If the server is started successfully (`status = 0`) a valid session handle is returned. This is to be used in all subsequent calls to the interface.

Note

A valid session handle is denoted by a positive number. The positive number represents the “internal identity” of an SB OLE Server object that has been created.

If the session handle is returned as a zero (0) then an error has taken place.

ShutdownServer()

This method should be called before the client application shuts down.

Note

Before calling this method, the client code should have sent the shut down sequence to the host program with sbc.SendEvent( ) that is "sbcom_exit" otherwise the host program will be left running.

This method uses the following syntax:

sbc.ShutdownServer(*session_handle, timeout*)

Program Example

```
Public SessionHandle As Variant
Public WithEvents sbc As SBCOMLib.SBClient
.
.
sbc.ShutdownServer (SessionHandle):
```

session_handle

Input This is the valid session handle received from a call to StartServer().

timeout

Input Optional. If specified, this timeout will take precedence over the timeout specified in StartServer().

The timeout in this method refers only to the time taken for the client code method call to return from SBClient. The method will return as soon as SBClient has sent the request/data to the host, that is it will not wait until the host receives (or processes) the request/data.

SendEvent()

This is the method that the client application uses to communicate with the host (or server machine).

This method takes the following syntax:

status = *objectname*.SendEvent(*session_handle*, *event_type*, *event_string*, *retStr, timeout*)

Program Example

```
Public WithEvents sbc As SBCOMLib.SBClient
Public SessionHandle As Variant
.
.
Status = sbc.SendEvent(SessionHandle, "DEBUG", "1;" & Picture1.hwnd, val)
```

This method can only be called if the server is in the READY state. The client program can determine the server state by calling the CheckServerState() method or waiting for a SERVER_STATE event to be generated (see the Events section that follows shortly).

status

Output Zero if the event was successfully sent to the server program. Note that no acknowledgment from the server is required. If a number between 2 and 7 is returned, this indicates that the server is not ready to handle events and the event was not sent to the host. The numbers indicate the actual state of the server. See the list of valid server states in the GetServerState method below.

session_handle

Input Valid session handle received from a call to StartServer( )

event_type

Type of event to generate to host. The table shown below identifies the values for each event_type. Valid event types are:

i. SB+

   This type is used when talking to an SB+ server. The data is packaged in a manner that SB+ understands.

ii. STRING

   This is used when sending any string directly to the host without any packaging.

iii. SB+SYNCHRONOUS

   This is used when sending data to the host, which requires a return value from the host before the client code can continue.

Warning

Careful consideration should be given before using SB+SYNCHRONOUS because client code will not continue running until the server has completed its actions. In effect this will “freeze” the client code unless the client code has other threads running. It is therefore recommended that SB+

is used instead, and to wait for a SERVER READY event from the host to achieve similar functionality.

iv. DEBUG

If this is set then the SBClient terminal will display the session while logging in.

The following table shows the values for each event_type/event_string combination.

	event_type			
	SB+	STRING	SB+ SYNCHRONOUS	DEBUG
event_string	session_handle & ";sbcom_process;" & processname session_handle & ";sbcom_modal;" & processname session_handle & ";sbcom_menu;" & menuname session_handle & ";sbcom_exit;"	"Any string"	session_handle & ";sbcom_process;" & processname session_handle & ";sbcom_modal;" & processname session_handle & ";sbcom_menu;" & menuname session_handle & ";sbcom_exit;"	"1" & window handle True 1 False 0

Possible Values

event_string

Overview User-defined parameter to be passed to the server program running on the host. The required value of the parameter depends on the event_type parameter shown in the above table.

Input When using SB+ or SB+ SYNCHRONOUS as the event type, the event_string must be made up of the following 3 parameters concatenated together and delimited with semicolons.

1. Object_Details - The Session handle (which is the same as the value passed in the session handle parameter shown above).
2. Event - One of the following events - sbcom_process, sbcom_modal, sbcom_menu , sbcom_exit and tree_modify. See the section on SB+ for a description of these events.
3. Any additional data required for the event.

Program Example

```
session_handle & ";sbcom_process;" & processname
```

When using STRING as the event_type, the event_string can be any literal string that you wish to send to the host.

When using DEBUG as the event_type, the event_string can be 1 or 0 to turn debug ON or OFF. If you also pass a valid Windows handle, the session will be displayed in your window e.g. "1;" & picture1.hwnd.

retStr

Output The value returned by the function when used in synchronous mode, i.e. event_type = SB+SYNCHRONOUS.

timeout

OverviewThis determines the time to allocate for a timeout.

Input Optional. If specified, this timeout will take precedence over the timeout specified in StartServer().

If using SB+, STRING, or DEBUG then the timeout in this method refers only to the time taken for the client code method call to return from SBClient. The method will return as soon as SBClient has sent the request/data to the host, that is it will not wait until the host receives (or processes) the request/data.

Otherwise, if using SB+ SYNCHRONOUS then this represents the entire time for the host to respond.

CheckServerState()

state = *objectname*.CheckServerState(*session_handle, timeout*)

state

Output Returns the current state value for the session handle. State can take one of the following values:

1 = READY

2 = BUSY

3 = SHUTDOWN

4,5,6,7 = Various sequential progress indicators during the login and connection to the host

for example 4 is 25% done.

Note

Any call to SendEvent() will fail immediately unless the server is in the READY state.

session_handle

Input Valid session handle received from a call to StartServer( ).

timeout

Input Optional. If specified this timeout will take precedence over the timeout specified in StartServer().

The timeout in this method refers only to the time taken for the client code method call to return from SBClient. The method will return as soon as SBClient has sent the request/data to the host, that is it will not wait until the host receives (or processes) the request/data.

SetOptions()

This method is used to set the various SBClient options and takes the following syntax:

status = *objectname*.SetOptions(*session_handle*, *options, timeout*)

Program Example

```
Public WithEvents sbc As SBCOMLib.SBClient
Public SessionHandle As Variant
.
.
sbc.SetOptions(SessionHandle, "server_state~1")
```

status

Output Returns zero (0) if the options were set successfully.

session_handle

Input This is the valid session handle received from a call to StartServer( )

options

Input SBClient client options. The only options currently supported are SERVER_STATE and LAST_EVENT.

SERVER_STATE can take one of the following values

1 - READY

2 - BUSY

3 - SHUTDOWN

The options are semicolon delimited and the option-value pairs are tilde i.e. ~ delimited.

objectname.SetOptions(*session_handle*, "SERVER_STATE~1")

Last event can be set to null to clear the value contained in the internal last event parameter. This is useful in determining whether you have missed an event in the case where, for example, Microsoft Visual Basic discards events. An example of this would be when you display a modal message box and Visual Basic discards events sent to it. In order to ensure that you don't miss an event when displaying such a dialog, you can set the LAST_EVENT to null and then when you remove the dialog you can check (using GetOptions) whether the value is still null or whether you need to process a "missed event".

timeout

Input Optional. If specified, this timeout will take precedence over the timeout specified in StartServer().

The timeout in this method refers only to the time taken for the client code method call to return from SBClient. The method will return as soon as SBClient has sent the request/data to the host, that is it will not wait until the host receives (or processes) the request/data.

GetOptions()

This is used to get the various SBClient options.

value = *objectname*.GetOptions(*session_handle*, *options, timeout*)

Program Example

```
Public WithEvents sbc As SBCOMLib.SBClient
Public SessionHandle As Variant
.
.
```

```
instpath = sbc.GetOptions(SessionHandle, "installpath")
```

value

Output The value of the requested options.

session_handle

Input Valid session handle received from a call to StartServer

options

Input These are the various SBClient client options. Currently the only supported options are INSTALLPATH and LASTEVENT.

INSTALLPATH - This returns the path of the working directory for SBClient.

LASTEVENT - See SetOptions().

timeout

Input Optional. If specified, this timeout will take precedence over the timeout specified in StartServer().

Chapter 13 - VBScript Interface

Support for Microsoft's VBScript language has been built into SBClient. This enables a UniBasic programmer to pass and execute Visual Basic code from a host system (UniVerse or UniData RDBMS). Using VB Script programmers may use standard features of Visual Basic.

Below is a list of the new VBScripting API's - There is an example of how to use this functionality in DEMOBP called SCRIPT.DEMO.

The VBScript APIs are:

TU.SCRIPT.ADD.CODE()

TU.SCRIPT.ADDOBJECT()

TU.SCRIPT.CREATE()

TU.SCRIPT.CREATE.MODULE()

TU.SCRIPT.EVAL()

TU.SCRIPT.EXECUTE()

TU.SCRIPT.LAST.ERROR()

TU.SCRIPT.LIST.FUNCTIONS()

TU.SCRIPT.LIST.MODULES()

TU.SCRIPT.RESET()

TU.SCRIPT.RUN()

VBScript API

TU.SCRIPT.ADD.CODE()

TU.SCRIPT.ADD.CODE(*Handle*, *Module*, *Code*, *Reserved*, *Options*, *Status*)

This function will add functions or procedures to a specific module.

Handle

[P] Handle to the SBClientScriptControl object.

Module

[P] Name of the module to add code to.

VBSCode

[P] Visual Basic Script code to add to the module.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.ADDOBJECT()

TU.SCRIPT.ADDOBJECT(*Handle*, *ObjectName*, *Expose*, *Reserved*, *Options*, *Status*)

This function will add an object to the name space of the scripting engine. For example, you may wish to add the Scripting.FileSystemObject to the scripting engine to gain access to its member functions.

Handle

[P] Handle to the SBClientScriptControl object.

ObjectName

[P] Name or Class Id of object to add.

Expose

[P] Boolean value to let script engine know if the objects members are globally accessible. Default value is True.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.CREATE()

TU.SCRIPT.CREATE(*Handle*, *Timeout*, *UseSafeSubSet*, *Reserved*, *Options*, *Status*)

This function will create an instance of the SBClientScriptControl object. If the object has previously been created then the handle to the previous instance is returned, otherwise the handle to the created instance is returned.

Handle

[R] Handle to the SBClientScriptControl object.

Timeout

[P] Length of time in milliseconds that a Visual Basic script can execute before being considered hung. Default value is 10000.

UseSafeSubSet

[P] Boolean value which tells the scripting engine to execute in safe mode. If set this will disallow potential harmful operations. Default value is True.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of further options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.CREATE.MODULE()

TU.SCRIPT.CREATE.MODULE(*Handle*, *Module*, *Reserved*, *Options*, *Status*)

This function will add a new module to the scripting engine.

Handle

[P] Handle to the SBClientScriptControl object.

Module

[P] Name of the module to create.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.EVAL()

TU.SCRIPT.EVAL(*Handle*, *Expression*, *Results*, *Reserved*, *Options*, *Status*)

This function will evaluate an expression in the scripting engine. It will return the results of the evaluation in the Results argument. The TU.SCRIPT.LAST.ERROR API should be called to determine if there are any run time errors.

Handle

[P] Handle to the SBClientScriptControl object.

Expression

[P] The expression to be evaluated.

Results

[R] Results of the evaluation of the expression.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.EXECUTE()

TU.SCRIPT.EXECUTE(*Handle*, *Statements*, *Reserved*, *Options*, *Status*)

This function will execute statements in the scripting engine. No results are returned by this function. The function TU.SCRIPT.LAST.ERROR should be called to determine if there are any run time errors.

Handle

[P] Handle to the SBClientScriptControl object.

Statements

[P] Statements to be executed.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.LAST.ERROR()

TU.SCRIPT.LAST.ERROR(*Handle*, *Error*, *Reserved*, *Options*, *Status*)

This function will interrogate the script engine as to what the last error was that occurred. If no error has occurred then an empty string is returned.

Handle

[P] Handle to the SBClientScriptControl object.

Error

[R] Error string.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.LIST.FUNCTIONS()

TU.SCRIPT.LIST.FUNCTIONS(*Handle*, *Module*, *Functions*, *Reserved*, *Options*, *Status*)

This function will return a VM delimited list of functions and procedures in the specified module.

Handle

[P] Handle to the SBClientScriptControl object.

Module

[P] Name of the module to query

Functions

[R] A VM delimited list of functions and procedures located in module.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.LIST.MODULES()

TU.SCRIPT.LIST.MODULES(*Handle*, *List*, *Reserved*, *Options*, *Status*)

This function will interrogate the script engine as to what modules can be found in its name space.

Handle

[P] Handle to the SBClientScriptControl object.

List

[R] A string containing a VM delimited list of module names

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.RESET()

TU.SCRIPT.RESET(*Handle*, *Reserved*, *Options*, *Status*)

This function will reset the scripting engine to its default state.

Handle

[P] Handle to the SBClientScriptControl object.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

TU.SCRIPT.RUN()

TU.SCRIPT.RUN(*Handle*, *Module*, *Function*, *Arguments*, *Results*, *Reserved*, *Options*, *Status*)

This function will run a specific function or procedure in the specified module. Any return values will be contained in the Results argument.

Handle

[P] Handle to the SBClientScriptControl object.

Module

[P] Name of the module to execute function or procedure in.

Function

[P] Name of the function or procedure to run.

Arguments

[P] A SVM delimited list of arguments.

Results

[R] string containing any results returned from the procedure call.

Reserved

[P] Argument reserved for future use.

Options

[P] A string of options. Reserved for future use.

Status

[R] A zero status indicates a successful call. A non zero status indicates a failure.

Chapter 14 - Dynamic Data Exchange

These APIs allow the host to connect to and control Windows applications (for example, downloading sales figures from the host into an Excel spreadsheet on the PC). They define the host as the DDE client, and the Windows application being communicated with as the DDE server. SBClient can also be a DDE server, where a Windows application can be a DDE client communicating with the host through SBClient using similar DDE commands. See DDE Server Interface.

The DDE Client API comprises:

- TU.DDE.CONNECT . Initiates a DDE session with a Windows DDE server application.
- TU.DDE.DISCONNECT . Terminates a DDE session.
- TU.DDE.EXEC.MACRO . Sends a macro to a DDE server and asks the application to execute the macro.
- TU.DDE.GET.ERROR . Returns a detailed reason for the failure of a DDE subroutine.
- TU.DDE.READ . Performs a read from a DDE server.
- TU.DDE.WRITE . Performs a write to a DDE server.

DDE Client API

TU.DDE.CONNECT

TU.DDE.CONNECT(*application*, *topic*, *handle*, *status*)

Initiates a DDE session with a Windows DDE server application. To ensure SBClient connects with the application, launch the application before calling this subroutine. See TU.LAUNCH.APP on how to launch the application.

You can use TU.DDE.CONNECT multiple times with the same application – in which case you will receive a different handle for each conversation. Once you have a conversation handle, you can use TU.DDE.READ, TU.DDE.WRITE or TU.DDE.EXE.MACRO to communicate with the application.

When the DDE conversation is complete, use TU.DDE.DISCONNECT to end it.

application

[P] The name of the Windows application in the DDE session. The application must be the DDE server (not all applications can be DDE servers). The DDE application name is not necessarily the name of the executable application. To identify the appropriate application name, consult your Windows or application documentation.

topic

[P] The topic of the DDE conversation as defined by the application. By convention, you can usually use the System topic to return the valid topics for the given application.

handle

[R] The unique value that identifies the DDE conversation. The handle value is used in subsequent DDE calls from the host.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. To identify the specific details of any error, call TU.DDE.GET.ERROR.

If your conversation was connected with the System topic, you may be able to use the following strings for itemname depending on the DDE server application:

String	Purpose
"SysItems”	Provides a list of all strings you can use with itemname.
"Topics"	Provides a list of all open projects. For example, a list of open Word documents.
“Status”	The current status of the application. For example, READY in Excel or EDIT in Quattro Pro when a cell is being edited.
“Formats”	A list of all clipboard formats supported by the application.
"Selection"	A list of all items currently selected in the application. For example, in Excel cells A3..A47 could be selected.

itemname strings for System topic connections

See also

TU.DDE.DISCONNECT, TU.DDE.EXEC.MACRO, TU.DDE.GET.ERROR, TU.DDE.READ, TU.DDE.WRITE, TU.LAUNCH.APP

TU.DDE.DISCONNECT

TU.DDE.DISCONNECT(*handle*, *status*)

Terminates a DDE session.

handle

[P] The unique value identifying the DDE conversation. handle is the value returned by the TU.DDE.CONNECT subroutine.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. To identify the specific details of any error, call TU.DDE.GET.ERROR.

See also

TU.DDE.CONNECTTU.DDE.CONNECT, TU.DDE.EXEC.MACRO, TU.DDE.GET.ERROR, TU.DDE.READ, TU.DDE.WRITE

TU.DDE.EXEC.MACRO

TU.DDE.EXEC.MACRO(*handle*, *timeout*, *data*, *status*)

Sends a macro to a Windows application and causes the application to execute the macro. Before calling this subroutine, a DDE session must be initiated. Initiate a DDE session by calling TU.DDE.CONNECT.

handle

[P] The unique value identifying the DDE conversation. handle is the value returned by the TU.DDE.CONNECT subroutine.

timeout

[P] The number of seconds SBClient waits for the application to respond before returning an error status.

data

[P] The macro being sent to the Windows application. data is defined by the application and it must conform to the syntax used by the application. For example, {PGDN} works as a macro command for Quattro Pro, but the equivalent macro command in Excel is [VPAGE(1)].

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. To identify the specific details of any error, call TU.DDE.GET.ERROR.

Note

Most DDE servers now use a relatively standard syntax for sending keystroke macro commands. See Macro Syntax.

See also

TU.DDE.CONNECT, TU.DDE.DISCONNECT, TU.DDE.GET.ERROR, TU.DDE.READ, TU.DDE.WRITE

TU.DDE.GET.ERROR

TU.DDE.GET.ERROR(*handle*, *data*, *status*)

Returns an explanation for the failure of an SBClient host library DDE subroutine.

handle

[P] The unique value identifying the DDE conversation. handle is the value returned by the TU.DDE.CONNECT subroutine.

data

[R] Contains the explanation for the failure of the subroutine. If the previous call to a DDE subroutine was successful, then data is undefined (that is, it must be called after the subroutine that gave the error and before any other DDE subroutine is called).

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.DDE.CONNECT, TU.DDE.DISCONNECT, TU.DDE.EXEC.MACRO, TU.DDE.READ, TU.DDE.WRITE

TU.DDE.READ

TU.DDE.READ(*handle*, *itemname*, *timeout*, *data*, *status*)

Requests data from a DDE server. Before calling this subroutine, a DDE session must be initiated, by calling TU.DDE.CONNECT.

handle

[P] The unique value identifying the DDE conversation. handle is the value returned by the TU.DDE.CONNECT subroutine.

itemname

[P] The name of the data item being requested from the application. itemname is defined by the application. In spreadsheets, this could be a block reference such as A2..A7. In ObjectVision, this could be the name of a field or record.

timeout

[P] The number of seconds SBClient waits for the application to respond before returning an error status.

data

[R] If TU.DDE.READ is successful, data contains the value returned by the Windows application.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. To identify the specific details of any error, call TU.DDE.GET.ERROR.

Note

It is more efficient to read blocks of data rather than small strings.

See also

TU.DDE.CONNECT, TU.DDE.DISCONNECT, TU.DDE.EXEC.MACRO, TU.DDE.GET.ERROR, TU.DDE.WRITE

TU.DDE.WRITE

TU.DDE.WRITE(*handle*, *itemname*, *timeout*, *data*, *status*)

Initiates data upload to a DDE server. Before calling this subroutine, a DDE session must be initiated, by calling TU.DDE.CONNECT.

handle

[P] The unique value identifying the DDE conversation. handle is the value returned by the TU.DDE.CONNECT subroutine.

itemname

[P] The name of the data item into which the data is to be written. itemname is defined by the application. For example, it could be a block of cells in Excel or a bookmark in Word.

timeout

[P] The number of seconds SBClient waits for the application to respond before returning an error status.

data

[P] The value being sent to the Windows application.

status

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. To identify the specific details of any error, call TU.DDE.GET.ERROR.

Note

It is more efficient to write blocks of data than small strings.

See also

TU.DDE.CONNECT, TU.DDE.DISCONNECT, TU.DDE.EXEC.MACRO, TU.DDE.GET.ERROR, TU.DDE.READ

Chapter 15 - DDE Server Interface

The DDE protocol is a fundamental means of inter-process communication built on top of the standard Windows messaging system, allowing Windows applications to exchange data on a real time basis.

Introduction

To exchange data, the two participating applications first have to engage in a DDE conversation. The application that initiates the conversation is known as the client application, and the application responding to the client is known as the server application. SBClient can be engaged in several conversations at the same time, and can act as a client application in some of them and as a server application in others.

The Word BASIC macro examples provided in this appendix are also available in the SBClient online help, from where you may paste them into a Word macro window (using the Tools:Macro option in Word) and run them.

DDE Message Components

DDE uses the three-level hierarchy application, topic, item to uniquely identify a unit of data.

The application name is the same name as the application's Windows executable file name without its .EXE extension. For SBClient, the application name is SBCLIENT (or SBOPEN if you are connecting via the SBDesktop environment). This naming convention facilitates automatic execution of the application if it is not running when a DDE Initiate request is made.

The data source that a DDE conversation connects with in the server application is called a topic. SBClient's topic names are not case sensitive. The application name, topic name pair uniquely identifies a conversation channel or conversation handle akin to a communications link instance (like a serial port connection or one network connection via a specific protocol). The client that opens or creates the connection must use the returned connection reference number for all subsequent reads and writes on the topic and is also responsible for closing or destroying the connection when the client is finished with it. SBClient always supports the topic "System", and session topics which are identified by the title of opened sessions.

The unit of data that may be read or written to is identified as an item of a specific topic. To discover what topics are available for connection, first connect to the "System" topic, then read the item "Topics". This will return a tabbed list of available topics (where individual elements are delimited with the Tab character "char(9)" – a DDE delimiting convention).

For example, the Microsoft Word macros in this appendix use Microsoft Word as the DDE client application and call SBClient as their server. To do so, they use the server name "SBClient" and specify either "System" or the name of an open session as the topic. The available data items vary depending upon whether you are talking to the "System" topic or to one of the opened sessions.

When "System" is the topic, you can request the item "Sysitems" which will return a list of all available DDE "System" items.

Starting SBClient Via DDE

SBClient allows you to create logical connections between your PC and one or more host machines. A connection is called a session. SBClient sessions appear in their own windows with their own menus and are available via DDE as individual topics. Advanced Scripting/SBDesktop note: SBClient coordinates multiple sessions using a session manager called termulatorappclass. This is the class that is accessed via the "System" topic.

When making your initial DDE connection to SBClient, and SBClient is not currently running, your DDE client software may automatically start SBClient by trying to launch the Windows application SBCLIENT.EXE. Even if this succeeds in starting SBClient, your connection may fail if you are trying to connect to a specific session that may not be open in time. The safest way to start SBClient and connect to a session is by initially connecting to the "SBClient", "System" topic. If this connection fails, you can start SBClient from your client DDE application via a Win-Execute or equivalent command, then try to connect to the "SBClient", "System" topic again. Once connected to the "System" topic, you can then read the "Topics" item to see if your session has started successfully with a predictable name or you can start your session via the DDE system item "create_session".

The following shows an example of a Microsoft Word BASIC macro:

Program Example

```
Sub MAIN
  DDETerminateAll
  ChanNum = DDEInitiate("SBClient", "System")
  MsgBox DDERequest$(ChanNum, "Topics")
  DDETerminate ChanNum
End Sub
```

Connecting to the Session Manager

To connect to the SBClient session manager via DDE, the client application should specify the application name without the extension (that is, "SBClient", with a topic of "SBClient"). The topic may also be "System", unless SBClient is invoked from the SBDesktop environment. Consult your application's DDE documentation for the appropriate DDE connection syntax.

Once the client application is connected to the "SBClient" or "System" topic, the following items may be DDE READ from the SBClient server:

Item	Returns
SysItems	A list of the items you can request from the "System" topic (this list).
Formats	A list of clipboard format types (as numbers) supported by SBClient. This is a read only item. (CF_TEXT format).
Topicfunctionlist	A list of the functions that may be executed via DDE EXECUTE or via scripting macros. This list may vary as scripts and external DDE may register more external or internal functions to this list.
Topicfunctionlist	A list of the functions that may be executed via DDE EXECUTE or via scripting
Topicitemlist	A list of the items available under the current topic. Some may be read only while others may be write only. This list can vary from moment to moment if an external DDE application grants itself access to more or less items from the "Attributes" item list.
Topics	A list of the topics supported by the application at the current time. This list may vary from moment to moment as sessions are created and destroyed.
create_session	A write only item used to create a new session. You must pass it a stored configuration name via DDE WRITE.
destroy_session	A write only item, as it needs information on how and what session to destroy or close. It is passed an exit-method number and an optional name of an open session title after a Tab character or value mark.

Items that can be DDE Read via SBClient

Item	Returns
Attributes	Returns a list of possible attributes supported by the topic server. The items not appearing in the items or "SysItems" lists are non-public items, but as these item lists are writeable you can give yourself access to an attribute as required. However, you may not be able to DDE READ from or DDE WRITE to some of these attributes as they may be create-only or read-only.
Items	A list of the items you can request via the current topic. As this is the "System" topic, this list is identical to the "SysItems" list.

Items that can be DDE Read via SBClient

The following Word BASIC macro demonstrates the SBClient DDE Server.

Program Example

```
Sub MAIN
       ' SBClient macro to demonstrate SBClient DDE Server connectivity
       ' This macro behaves differently if it doesn't find SBClient and
       ' has to launch it.
       DDETerminateAll
       ' First test to see if SBClient is running and if not, launch it..
       If AppIsRunning("SBClient") = 0 Then
              'Rem 0 (zero)=Minimized window (icon), 1=Normal window,
              'Rem 2=Minimized window, 3=Maximized window, 4=Deactivated window
              Shell Chr$(34)+ "SBClient"+ Chr$(34), 0
              ' Wait a few seconds to give SBClient to fire up

RightNow = Now()
              While (Now() - RightNow < 0.0001)
                     'MsgBox Str$(Now() - RightNow), "", - 1
              Wend
       End If
       ' Let's talk DDE !
       ChanNum = DDEInitiate("sbclient", "System")
       SysItems$ = DDERequest$(ChanNum, "SysItems")
       Topics$ = DDERequest$(ChanNum, "Topics")
       Formats$ = DDERequest$(ChanNum, "Formats")
       ' Lets display what we have captured via DDE...
       DisplayTabbedList(SysItems$, "List of System Items")
```

```
        DisplayTabbedList(Topics$, "List of System Topics")

DisplayTabbedList(Formats$, "List of System Formats")
        ' If there are no sessions open then this will close SBClient!
        DDETerminate ChanNum
        ' We'll have a look for an open SBClient Session...
        Tab$ = Chr$(9)
        List$ = Topics$
        LastItem$ = ""
        ' Count the number tabs contained in string
        ' which is equal to the number of items
        cTabs = 0
        FoundTab = InStr(1, List$, Tab$)
        While FoundTab
                cTabs = cTabs+ 1
                LastItem$ = Right$(List$, Len(List$) - FoundTab)

FoundTab = InStr(FoundTab+ 1, List$, Tab$)
        Wend
        If cTabs > 1 Then
                ' FOUND an open Session !@#!
                ChanNum = DDEInitiate("sbclient", LastItem$)
                Items$ = DDERequest$(ChanNum, "Items")
                Topics$ = DDERequest$(ChanNum, "Topics")
                Functions$ = DDERequest$(ChanNum, "TopicFunctionList")
                ' Finished with session topic...
                DDETerminate ChanNum
                ' Lets display what we have found..
                DisplayTabbedList(Items$, "List of Session Items")
                DisplayTabbedList(Topics$, "List of Session Topics")

DisplayTabbedList(Functions$, "List of Session Functions")
                ' Ok, now close the Last Session - optional extra!
                'ChanNum = DDEInitiate("sbclient", "System")
                'DDEPoke ChanNum, "destroy_session", "0" + Chr$(9) + LastItem$
                'DDETerminate ChanNum
                End If
        End Sub
        ' This is prettier than using MsgBox ...
        Sub DisplayTabbedList(srcList$, Description$)
                Tab$ = Chr$(9)
                AM$ = Chr$(254)
                List$ = srcList$
                ' Count the number tabs contained in string
```

```
' which is equal to the number of items
            cTabs = 0
            FoundTab = InStr(1, List$, Tab$)
            FoundAM = InStr(1, List$, AM$)
            If FoundAM = 0 Then FoundAM = Len(List$)+ 1
            While FoundTab And FoundTab < FoundAM
                  cTabs = cTabs+ 1
                  FoundTab = InStr(FoundTab+ 1, List$, Tab$)
            Wend
            ' Create the array to hold the information
            Dim ListBox$(cTabs)
            ' Extract tab delimited items and store in ListBox$ array
            FoundTab = InStr(1, List$, Tab$)

For i = 1 To cTabs
                  ListBox$(i - 1) = Left$(List$, FoundTab - 1)
                  List$ = Right$(List$, Len(List$) - FoundTab)
                  FoundTab = InStr(1, List$, Tab$)
            Next i
            ' Snag last item, if it exists!
            If Len(List$) > 0 Then
                  FoundAM = InStr(1, List$, AM$)
                  If FoundAM Then
                        ListBox$(cTabs) = Left$(List$, FoundAM - 1)
                  Else
                        ListBox$(cTabs) = List$
                  End If
            End If
            ' Create and display dialog box with items in listbox

Begin Dialog UserDialog 320, 144
                  ListBox 11, 23, 296, 84, ListBox$(), .ListBox
                  OKButton 11, 113, 296, 21
                  Text 11, 4, 296, 14, Description$
            End Dialog
            Dim dlg As UserDialog
            GetCurValues dlg
            Dialog dlg

End Sub
```

Connecting to a Session Topic

You can establish a connection to a session by connecting to SBClient and its session topic name. To find, or test for the existence of a session server, DDE READ the "SBClient, "Topics" item. This is necessary as you may have multiple sessions of the same type of connection uniquely identified by numeric suffixes.

The session topic is the session name (that is, Unidata on Minotaur: 5).

Having found the name of the session you wish to converse with, initiate a new conversation with the session name as the topic:

```
ChanNum = DDEInitiate("sbclient", "Unidata on Minotaur: 5")
```

Once connected to a session, the same DDE items "Topicfunctionlist" and "Topicitemlist" can be requested to reveal the functions and items now available.

The following shows an example of a Microsoft Word BASIC macro:

Program Example

```
Sub MAIN
DDETerminateAll
  ChanNum = DDEInitiate("sbclient", "default serial: 1")
  Items$ = DDERequest$(ChanNum, "Topicitemlist")
Functions$ = DDERequest$(ChanNum, "Topicfunctionlist")
  DDETerminate ChanNum
       DETerminate ChanNum
DisplayTabbedList(Items$, "List of Session Items")
DisplayTabbedList(Functions$, "List of Functions")
End Sub
Sub DisplayTabbedList(srcList$, Description$)
       Tab$ = Chr$(9)
       List$ = srcList$
       ' Count the number tabs contained in string

' which is equal to the number of items
       cTabs = 0
       FoundTab = InStr(1, List$, Tab$)
       While FoundTab
              cTabs = cTabs+ 1
              FoundTab = InStr(FoundTab+ 1, List$, Tab$)
       Wend
```

```
        ' Create the array to hold the information
        Dim ListBox$(cTabs)
        ' Extract tab delimited items and store in ListBox$ array
        FoundTab = InStr(1, List$, Tab$)
        For i = 1 To cTabs
                ListBox$(i - 1) = Left$(List$, FoundTab - 1)
                List$ = Right$(List$, Len(List$) - FoundTab)

 FoundTab = InStr(1, List$, Tab$)
        Next i
        ' Snag last item, if it exists!
        If Len(List$) > 0 Then
                ListBox$(cTabs) = List$
        End If
        ' Create and display dialog box with items in listbox
        Begin Dialog UserDialog 320, 144
                ListBox 11, 23, 296, 84, ListBox$(), .ListBox
OKButton 11, 113, 296, 21
                Text 11, 4, 296, 14, Description$
        End Dialog
        Dim dlg As UserDialog
        Dialog dlg

End Sub
```

Item Parameters

This topic documents some of the items available via DDE READ, DDE WRITE or DDE POKE. The overall availability of the items can be gleaned from the "Items" item under the current topic. See Macro Syntax for a description of functions that can be DDE EXECUTED and found under the "Topicfunctionlist" item.

Items are:

- attributes
- create_session
- destroy_session
- formats

- items
- keep_session_minimized
- suppress_session_mode
- topicfunctionlist
- topicitemlist
- topics

attributes

Read only.

Returns a tab delimited list of items supported by this topic. Each item is followed by a code: C (Create only), R (Read only) or W (Write only).

For example:

Attributes$ = DDERequest$(ChanNum, "Attributes")

Some attributes are described below, others are found in the SBDesktop documentation. Unidata cannot guarantee the existence or behavior (from one release to another) of attributes not documented below. The behavior of the attributes described here may change.

create_session

Write only.

This is write only as it needs to be passed the name of a stored configuration to use to create a session instance.

For example:

DDEPoke ChanNum, "create_session", "Unidata on Minotaur"

destroy_session

Write only.

This will destroy or close the current or nominated session. If you use this to destroy the current session and are using the current session topic then you will invalidate your DDE channel or handle as the session closes. If you wish to destroy or close the current session, re-initiate the DDE conversation to the "System" or "SBClient" topic and issue the DDE Poke from there. By using this method, SBClient will remain alive until you terminate your DDE conversation with the "System" topic and you have not created another session.

The write data has two components. The first is the exitmethod code and it may be optionally followed by a tab character or a value mark and the name of the session to close. The name of the session to close must be precise; it usually has a colon and an instance number appended to its stored configuration name.

exitmethod may be:

0	full confirmation required
1	say NO to save configuration changes
2	say YES to save configuration changes
4	say NO to exit session dialog or reconnect
8	say YES to exit session dialog
16	say NEW SESSION to exit session dialog
32	say NO to exit SBClient dialog
64	say YES to exit SBClient dialog
128	will close all sessions

Program Example

```
Destroy$ = "1"+ Chr$(9)+ "Unidata on Minotaur: 1"
DDEPoke ChanNum, "destroy_session", Destroy$
```

formats

Read only.

Returns a list of clipboard format types (as numbers) supported by SBClient (for example, TEXT format.). More formats may be supported in future releases of SBClient.

items

Read and write.

This is a list of available items that can be DDE READ or DDE WRITE / POKE or both. If you try to DDE READ or DDE WRITE or DDE POKE to an item not in this list, it will fail. As this list can be written back, access to various items can be removed or granted as required.

keep_session_minimized

Write only.

This is equivalent to the command line option -M except it can turn this mode off (by setting the value to 0). If set via the "System" topic, sessions subsequently created will initially be and remain minimized (or not). If set via a session topic, the effect is immediate to either minimize the session or to restore the session.

suppress_session_mode

Write only.

This is equivalent to the command line option -S except it can turn this mode off (by setting the value to 0). If set via the "System" topic, sessions subsequently created will initially be minimized until a login script has run its course (or not). If set via a session topic, the session is immediately minimized or restored.

topicfunctionlist

Read and write.

A list of the functions that may be executed via DDE EXECUTE or via scripting macros. This list may vary as scripts and external DDE may register more external or internal functions to this list.

topicitemlist

Read and write.

A list of the items available under the current topic. Some may be read only, others may be write only. This list can vary from moment to moment if an external DDE application grants itself access to more or less items from the "Attributes" item list.

topics

Read and write.

A list of the topics supported by the application at the current time. This list may vary from moment to moment as sessions are created and destroyed.

Base System Topic Definition Item

The base definition item specifying SBClient's manager DDE and macro capabilities can be found in the SBDesktop file TUDEFN, item SBCLIENT.DDE. This is the "System" topic base capability list. Consult the SBDesktop manual on how to access this item.

The layout of the item is:

Attribute 1	'DDE'
Attribute 2	multivalued list of "System" topics
Attribute 3	multivalued list of accessible items
Attribute 4…	local macro function definitions, one per attribute. The format of these function definitions is:

Value 1:	function name
Value 2:	number of arguments expected
Value 3:	0 : internal, 1 : local, 2 : external function type

Base Session Topic Definition Item

The base definition item specifying SBClient's session DDE and macro capabilities can be found in the SBDesktop file TUDEFN, item SBCLIENT.SESS.DDE. Consult the SBDesktop manual for information on how to access this item.

The layout of this item is:

Attribute 1	'DDE'
Attribute 2	multivalued list of "System" topics
Attribute 3	multivalued list of accessible items
Attribute 4 …	local macro function definitions one per attribute. The format of these function definitions is:

Value 1:	function name
Value 2:	number of arguments expected
Value 3:	0 : internal, 1 : local, 2 : external function type

Note

There are more macro functions at this level that can be DDE-Executed.

Chapter 16 - GUItization

SBClient provides an integrated development environment (IDE) for creating a graphical user interface (GUI). This IDE comprises two main components:

- Form Painter. Allows you to create objects and modify their properties. It may be used by HostGUI developers.

GUItization Concepts

SBClient's approach to GUItization is based on Windows GUI objects. These are generated locally on the PC, with the host-based application controlling and manipulating them as needed. The original character-based screen is rendered as a genuine Windows form. The SBClient form painter allows you to edit and visually enhance the objects on the form.

SBClient's GUItization design goals are to:

- make it easy for both programmers and non-programmers to create a GUI to a character-based application
- allow them to visually enhance their forms
- allow them to easily maintain their forms
- provide them with a powerful and extensible toolset
- provide them with an efficient and reliable toolset
- ensure a smooth migration path

The following is a brief description of some of the concepts used in relation to GUI objects. A more thorough discussion of these is provided in the Data/C++ Reference Manual and The Data/C++ Programming Language book.

- Object. In the context of this document, an object is a graphical element that can be drawn on a form. Each type of GUI object is defined by the properties and behavior of its class.
- Class. An object belongs to a class of object. This class defines the default properties and behavior of an object created from that class. An object instantiated (created) from a class is referred to as an instance of that class. For example, when you initially create a label object, the properties of the object are based on the default class definition for label objects. You can then modify the properties of the object as required.
- Property. Objects have properties, the values of which affect their appearance and behavior. For example, the value of the background property of a text object defines the background color of the object.

- Attribute. GUI object classes and their properties are based on, but add to, the standard Data/C++ object classes and their attributes. The terms attribute and property are equivalent in the context of this reference manual.
- Event. Objects can be programmed to respond to events that happen to them. The occurrence of an event generates an (attribute-mark delimited) event string that describes the event. The event string generated depends upon the type of event. For example, pressing a special key (such as Enter or Esc) while in a text object will generate a special event on that object.
- Callback. Callbacks are user-defined subroutines that are called in response to a particular event. They are used as an effective way of dealing with events.

See GUI Classes and Their Attributes for a description of GUI object properties. See Events for a description of events and callbacks. See Properties & Events Window for details on how to modify the values for an object's properties.

GUI Classes and Their Attributes

An object class defines a type of object (for example, a label), by specifying what its attributes are (for example, justification, coordinates and so on) and how they behave. An attribute's behavior is determined by the set of values (left, right and so on) that may be assigned to it when an instance of the object is created.

This appendix describes the underlying attributes that correspond to the properties used to define GUI objects in the form painter (that is, within the Properties & Events Window). You can also modify and set these attributes once the object has been created. The terms attribute and property are equivalent in the context of this reference manual.

Attribute delimiters vary according to where they are used:

- In the form painter the attribute delimiter is a semicolon. This is what is used in this appendix.
- In code it is a VM.
- In a form definition it is an SM.

The standard text, edit, label, toggle, radio and optionlist classes have been replaced in the form painter by new object classes (with the prefix host). These new classes inherit from the standard classes, described below, but add additional attributes required by AutoGUI and HostGUI. See Attributes and New GUI Attributes.

GUI Class Attributes

form

background, border_color, border_style, border_width, coordinates, dimensions, drawable, icon, maximize, maximizable, minimize, minimizable, title, title_bar, tile

label

background, border_color, border_style, border_width, coordinates, dimensions, drawable, emphasized, font, tile, graphic, justification, string, foreground

text

background, border_color, border_style, border_width, coordinates, dimensions, drawable, editable, font, foreground, special_key_set, justification

bpbutton

background, border_width, coordinates, dimensions, down_graphic, down_string, drawable, emphasized, font, foreground, graphic, justification, string, tile

rectangle

background, border_color, border_style, border_width, coordinates, dimensions, drawable, tile

separator

order_style, coordinates, dimensions, direction, drawable, foreground

toggle

background, border_color, border_style, border_width, coordinates, dimensions, drawable, emphasized, false_graphic, font, foreground, special_key_set, string, state, style, tile, toggle_ border_width, toggle_size, true_graphic

radio

radio_set, background, border_color, border_style, border_width, dimensions, drawable, emphasized, tile

combo box

background, border_color, border_style, border_width, coordinates, dimensions, drawable, font, foreground, record_set, num_lines, scroll, special_key_set

Standard Attributes

Standard attributes used by the properties in the form painter are as follows:

back

A Boolean value forcing the object to be displayed at the back of existing objects. Possible values are TRUE and FALSE. Unlike other attributes, setting this attribute to TRUE performs an immediate action (sending the object to the back of other objects), and does not hold its value. Setting this attribute to FALSE therefore has no affect.

background

The background color of the object. This attribute is ignored if a bitmap or tile is being used.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

border_color

The color of the border to be drawn around the object.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

border_style

The type of border to be drawn around the object. Possible values are:

raised The border is shaded so that the object appears to be elevated from the background surface.

lowered The border is shaded so that the object appears to be sunk into the background surface.

flat The border is flat; neither raised or lowered.

msstyle The border is set two pixels wide and shaded to appear to be sunk into the background surface. This style emulates Microsoft's standard three-dimensional text input field border.

border_width

The width of the border to be drawn around the object. The value is given in pixels. Borders encroach upon the dimensions of the object, and their width should be considered when sizing an object and positioning children.

coordinates

The pixel address, relative to the object's parent, of the top-left corner of the object. Its format is x;y.

cursor

Defines a bitmap (as xxx.CUR) for the cursor used by an object. This is a system dependent attribute, and is normally an arrow, but may change from one object to another.

dimensions

The pixel width and pixel height of the object. Its format is width;height.

direction

The direction of the separator line. Possible values are HORIZONTAL and VERTICAL.

down_graphic

The graphic to be displayed when a button is displaying a bitmap, and the push button is in the pressed state. If this is not set, the bitmap defined in the graphic attribute is used but offset two pixels right and two pixels down.

down_string

The string to be displayed when a button is displaying a text string, and the push button is in the pressed state. This defaults to the value of string.

drawable

A Boolean value determining whether the object is rendered visible. An object is rendered provided it, and all ancestors, are renderable.

editable

A Boolean value determining whether the text in the object can be edited. Possible values are TRUE and FALSE.

emphasized

A Boolean value determining whether the object is capable of receiving input events (for example, mouse, keyboard and so on). Possible values are TRUE and FALSE. If FALSE, the object is displayed in the system-defined de-emphasized state. For example, a menu option might be shown grayed out if it is currently of no relevance to the user.

false_graphic

The bitmap to place on a toggle when in the FALSE state.

font

The font used when an object is displaying a string. Its format is:

*fontfamily*pointsize*style*effect*pitch*charset*

Program Example

```
Arial*8*bold*normal*pitch_variable*charset_ansi
```

foreground

The color of text in the object or the on-bit value if the object is displaying a bitmap.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

front

A Boolean value which forces the object to be displayed at the front of existing objects. Possible values are TRUE and FALSE. Unlike other attributes (apart from the back attribute), setting this to TRUE performs an immediate action (sending the object to the front of other objects), and does not hold its value. Setting this attribute to FALSE therefore has no affect.

graphic

The name of a file containing bitmap information. The format of this file is window-system specific.

icon

The icon file (xxx.ICO) used to represent the minimized form.

item_set

A multi-value list of records to appear in the list. This attribute accepts the standard set notation modifiers (APPEND, DELETE, index and value).

justification

The justification used when displaying either a string or graphic. Possible values are: left, center, right, top_left, top_center, top_right, bottom_left, bottom_center, bottom_right.

maximize

A Boolean value determining whether the form is maximized. To restore a maximized form to its previous size, set this attribute to FALSE.

maximizable

A Boolean value determining whether a maximize control is available for the form.

minimize

A Boolean value determining whether the form is minimized. To restore a minimized form to its previous size, set this attribute to FALSE.

minimizable

A Boolean value determining whether a minimize control is available for the form.

num_lines

The number of lines that will be displayed in the optionlist dropdown box.

scroll

A Boolean value determining whether a vertical scrollbar is available.

special_key_set

A value-mark delimited list of the special keys that generate a special event.

state

A Boolean value determining whether the object is toggled.

string

The string of characters to appear on the object. The type of label (character or bitmap) is determined by whether the string or graphic attribute has been most recently set.

style

Defines the style of the toggleclass object. The following predefined styles are available:

- CHECK. Displays a check mark when selected.
- CROSS. Displays a cross when selected.
- BUTTON. Is displayed recessed when selected.
- DIAMOND. Displays a filled-in diamond when selected.
- CIRCLE. Displays a filled-in circle when selected.
- BITMAP. Displays true and false graphic bitmaps on a button.

- MSCROSS. Displays a Windows style three dimensional cross.
- MSCIRCLE. Displays a Windows style three dimensional circle. Only toggle sizes of 13, 15 and 17 are supported for this style.

The first three toggles are generally used for checkboxes; where more than one checkbox can be toggled on at a time.

tile

The name of a file containing a bitmap to be used as the background for the object.

title

The text to appear as in the title bar for the window.

title_bar

A Boolean value determining whether a form has a title bar.

toggle_border_width

The width of the borders in CHECK, BUTTON, DIAMOND and BITMAP style toggles.

toggle_size

The width and height of the toggle button in pixels. Since the toggle button is a perfect square, only one dimension is specified.

true_graphic

The bitmap to place on the button and check toggle when in the TRUE state. If the style attribute is set to CHECK, the default for the true_graphic attribute becomes CHECK_BMP.

Additional Form Painter Attributes

Additional attributes used by the properties in the form painter are as follows:

char_col & char_row

These specify the position of an input or label on the character screen. It is these coordinates that link an object on a form to its corresponding input field or label on a character screen. This allows AutoGUI to set focus to a text object or update a label object's string value. These coordinates have no bearing on the position of the object on the GUI form.

char_length

This is the length of the character field or label associated with the object. It is initially derived when you enter a value into the character field, but may be changed.

dynamic

This determines whether the value of an object is redisplayed whenever data is sent from the host. The default for all object classes, except hostlabelclass, is TRUE (the Properties & Events Window only shows this attribute for hostlabelclass). If the labels on the form do not change, it is more efficient (and avoids flickering whenever a clear screen escape sequence is sent from the host) to set this to FALSE for hostlabelclass. This ensures its value will not be updated or cleared by AutoGUI. However, if the form has labels that do change then set the dynamic property for hostlabelclass to TRUE.

hide_until_inp

This only applies to text objects. Setting it to TRUE ensures that the object will only become drawable when focus is passed to it and will disappear when it loses focus. This is not standard Windows GUI behavior but may be useful where the appearance of input fields clutters the form or creates confusion.

true_value & false_value

Both these attributes apply to hosttoggleclass and determine what key value should be sent to the host when their toggle state is TRUE or FALSE. The false_value attribute also applies to hostcomboclass.

transparent

There is a new attribute called 'transparent' on HOSTPBCLASS and HOSTLABELCLASS.

If the 'transparent' attribute is set to a particular RGB color, then the value it is set to is used as a transparent mask for a bitmap. e.g. if the form definition has the value set to 192:SVM:192:SVM:192, where SVM is char (252) then any part of a bitmap that has that color set, will display the color of the object underneath it.. If, for example, it is required to have a button with a bitmap on it that picks up the default windows button colors, then the button itself should not have its background set and the 'transparent' attribute should be set to the desired color.

remove_tab

There is a new attribute called 'remove_tab' in HOSTEDITCLASS that when set, will not return the tab character to the host when the user tabs off the field.

reset_on_esc

There is a new attribute called RESET_ON_ESC that is available in hosttextclass and hostedit-class. If this attribute is set then when the user hits ESC on a field, the original value will be restored without the host having to reset the field.

SPLIT

This property is for use with hosteditclass. If this attribute is set then the hosteditclass will insert VM's at the places where the string has wrapped.

The above new attributes are not available in the Form Painter yet and need to be manually added to the form definitions.

GUI Objects

A GUI is comprised of various types of objects:

- form

  Forms provide the fundamental framework for an application's user interface. When GUItizing a host-based application with SBClient, each character screen is converted into an equivalent form.

  A form is a standard window that can be used to display application data and to initiate actions using objects and menus.

  SBClient supports the following menu types:

    - Type 1. Typical character-based menu. They pop-up and stay on the screen until explicitly removed. They may be either attached to a form or created stand-alone.
    - Type 2. Standard Windows-style menu. They are always attached to a form.

      The form's x,y coordinate system has (0,0) in the top left of the form with x increasing to the right across the form and y increasing downwards. The normal unit of measurement is pixels.
- label (used for screen and input field captions).
- text (used for input fields).
- push button (used for initiating an action).
- combo box (used for displaying a selection list of options).
- toggle (used for a checkbox that can be selected or deselected).
- radio (contains buttons used for the selection of mutually exclusive options).
- separator (used for adding lines to a form for aesthetic reasons).
- rectangle (used for adding rectangles to a form for aesthetic reasons).

GUI objects are created using the form painter IDE.

GUI Strategies

Most business applications may be structured into the following logical layers:

- the database layer
- the application layer (business rules)
- the presentation layer (user interface and screen handling)

Note that these are logical, not physical layers. In a traditional host-based application, all three layers reside on the host.

A clean logical separation of these three layers makes it easier to move one or more layers from the host to the client. For example, a clean presentation layer might feature a screen definition data structure (typically stored in a screens database) which is passed to a screen-drawing subroutine. This screen-drawing subroutine, in turn, might work in concert with a companion input-from-screen subroutine and perhaps, a front-end menu system.

GUItization is concerned with moving the presentation layer to the client.

GUI Models

"How do you generate a genuine GUI form that is equivalent to each screen in the host-based application?"

SBClient provides the following GUItization models as a solution to this question:

- AutoGUI. See AutoGUI.
- HostGUI. See HostGUI.

SBClient allows you to start with the AutoGUI model, then advance to the HostGUI model without having to throw away earlier work. A single system may consist of applications that use different models.

AutoGUI is a toolset (available within the SBClient IDE), while HostGUI is a methodology. AutoGUI and HostGUI are applicable only to nested relational database environments. Both AutoGUI and HostGUI support character applications on ASCII terminals.

With HostGUI you are able to create your own custom GUI objects on the PC using the Data/C++ programming language, VBX standard and custom controls or other visual tools. These objects can be created and manipulated from the host using the host library.

Note

AutoGUI is no longer supported. It is only provided for sites which currently use it.

Events

A user action on the form generates an event to which the application responds as best it can. The event-driven application waits for a user event, responds to it, then waits for the next event and so on.

Most user actions on a form will generate an event. Each type of event is identified by a unique name. For example, clicking the mouse on a push button will generate an activate event for that button. Similarly, pressing a special key (such as Esc or Enter or one of the function keys) will generate a special event.

Associated with each event is a corresponding attribute-mark (AM) delimited event string that fully describes the event. The first two mandatory attributes of the event string are:

- eventname. The name of the event (e.g. activate).
- formhandle:objectname. The form handle uniquely identifies a form and is returned by the TU.FORM.LOAD subroutine. The object name is the name given to the object when the form is created. Together, these two uniquely identify the object that has generated the event. In HostGUI mode, this property has the format formhandle:objectname:col,row where col and row are the starting column and row of the field in the character screen.

Note

Details of events and their associated event strings can be found in the Data/C++ Reference Manual. Note, however, that Data/C++ event strings use objecthandle as the second property instead of formhandle:objectname.

An event string may also contain additional attributes, depending on the type of event. A special event, for instance, contains a third attribute indicating which special key was pressed.

For example, suppose the form handle is 14 and the object name is obj. If this object is a push button, an activate event occurring on it would generate the following event string:

"activate":AM:"14:obj"

If, instead, the object was a text object (used for input fields), the user pressing Esc while in this field would generate the following event string:

"special":AM:"14:obj":AM:27

27 being the decimal representation of the Esc key.

Chapter 17 - Form Painter

The form painter environment consists of the following components:

- **Menu Bar**. The form painter menu.
- **Toolbar**. Provides buttons for certain form painter menu options.
- **Object Bar**. Displays buttons for the different classes of objects that may be placed on the form.
- **Color Palette**. Allows you to specify the foreground and background colors for a form or selected objects.
- **Properties & Events Window**. Allows you to view and/or change the values of an object's properties and the subroutines associated with the object's events.

Overview

Manipulating Objects on a Form

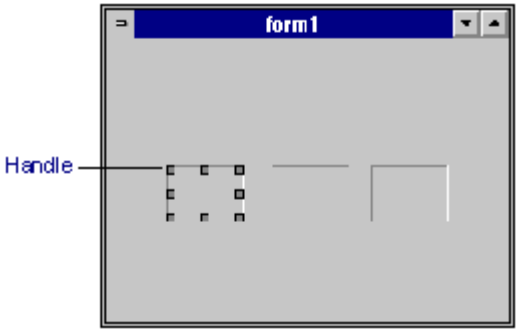
This topic describes the ways you can manipulate objects on a form.

Placing objects on a form

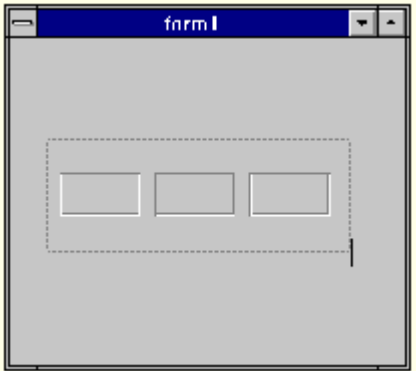
To place an object on the form, click the appropriate object button on the Object Bar and draw the object to the appropriate scale on the form (see Object Bar).

Selecting objects

To select a single object, click on the object. Selected objects are shown with handles:



To select multiple contiguous objects, lasso the objects:



To select multiple non-contiguous objects, hold down Shift (or Ctrl) between selections.

To deselect one or more objects, press Shift and click on each object. To deselect all objects click on the background form.

Resizing objects

To resize a single object, click the left mouse button on a handle of the selected object, and drag the handle. To resize multiple objects, press Shift, click the left mouse button on a handle of one of the objects and drag the handle. All selected objects are then resized.

To resize objects using the keyboard, press Shift and use the arrow keys to resize the objects. Objects are resized in single pixel increments or, if Options:Snap Grid is selected, in increments set using Options:Grid Dimensions. Press Ctrl during resizing to resize the objects in increments of 10 pixels or a grid dimension of 3 (if Options:Snap Grid is selected).

Moving objects

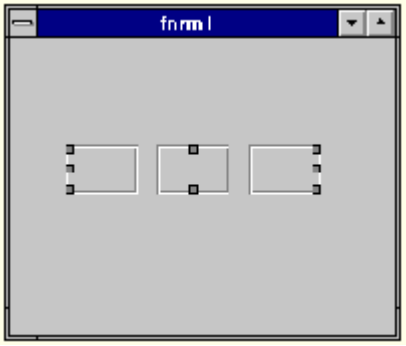
To move a signal object using the mouse, point at the object. This changes the cursor to a hairline. Drag the object to its new position. As you drag the object, a ghost outline will indicate its position.

To move multiple objects using the mouse, select the objects you wish to move, point at one of the objects, press Shift (or Ctrl) and drag the object to its new position. All selected objects are then moved to the new position.

To move objects using the keyboard, select the objects and use the arrow keys to move them. Objects are moved in single pixel increments or, if Options:Snap Grid is selected, in increments set using Grid Dimensions. Press Ctrl during moving to move the objects in increments of 10 pixels or a grid dimension of 3 (if Options:Snap Grid is selected).

Grouping objects

To construct a single object from selected objects, choose Edit:Group:



This allows you to move the group around the screen as you would a single object. To deconstruct the group, choose **Edit > Ungroup**.

Deleting objects

To delete selected objects, select the appropriate objects and choose Edit:Delete (or click Delete).

Changing an objects properties

Once an object is selected its properties are displayed in the Properties & Events Window (see Properties & Events Window), allowing you to modify them.

To change an object's foreground or background color use the Color Palette (see Color Palette).

You can manipulate objects using either the mouse or the keyboard, in conjunction with Shift and Ctrl.

You may further manipulate objects on the form using the options on the Edit menu (see Edit).

Properties and Events Window

The Properties & Events Window allows you to view and modify the properties and events of an object (you can switch between the Properties and Events dialogs). You can toggle it on and off from the Window menu, or close it directly.

See Events for details of events. See GUI Classes and Their Attributes for details of the attributes on which the properties in the Properties dialog are based.

Click on an object to display its properties in the Properties & Events dialog. You may need to click on the Properties tab if the Properties dialog is not displayed.

If objects of the same class are selected, the Properties dialog does not show any values. However, you may enter common values for a property.

If objects of different classes are selected, the Properties dialog shows only those properties that are common to all the classes selected.

If no object is selected, the Properties dialog shows the properties of the form.

To modify the value of a property for a selected object, select the appropriate property. The value of the property is displayed in the value edit field. Modify this and click Enter. You can display an Intuitive Help Window for help in entering the correct value for certain properties.

Note

The Intuitive Help button on the Properties & Events Window displays a dialog showing the range of valid entries for the selected property.

The button is enabled only for properties for which intuitive help is available.

Color Palette

The Color Palette allows you to specify the foreground (text) and background colors for a form or selected objects. You can toggle it on and off from the Window menu or close it directly.

To change a color, click on the form or objects whose color you wish to change, select the foreground or background radio button, and choose the appropriate color. The foreground radio button is disabled if not applicable for an object.

The Color Palette only shows 16 colors. To select another color, enter its RGB value directly into the property field or use intuitive help for the property.

Object Bar

The Object Bar displays buttons for the different classes of objects that may be placed on the form. You can toggle it on and off from the Window menu, or close it directly. Click the right mouse on a button to display balloon help for that object.

Object Bar buttons are as follows:

Selector

Allows you to select objects on a form.

Label

Label objects are used primarily for creating field titles. They may also be used for displaying a bitmap, by setting the graphic property of the label to the name of the BMP file.

Text

Text objects allow users to enter a single line of text.

Edit

Edit objects allow users to enter multiple lines of text. Edit objects are similar to text objects, except that they have additional functionality to cater for multiple lines of text.

Push Button

Push button objects allow users to generate an event. See Events.

Combo Box

Combo box objects display an option list of items, with the currently selected option being displayed in a text field. The user can only select an item that is in the list. Whether the user can edit the contents of the selection field depends on its style. If the list box is visible, typing characters into the selection box will cause the first list box entry that matches the characters typed to be highlighted. Conversely, selecting an item in the list box displays the selected text in the selection field.

Toggle

The toggle object is more commonly known as a checkbox. It is used for data that only has two values (yes and no), and can be toggled on or off when selected. The appearance of the button changes when selected.

Separator

Separator objects are simple horizontal or vertical lines.

Rectangle

Rectangles objects are simple geometric rectangles. You need to create objects in the proper order if the illusion of objects being contained in the rectangle is intended. Creating such objects before creating the rectangle results in the rectangle obscuring what is behind it (unless the back property is set to TRUE).

Radio

A radio object is a rectangular area on the form to which you add toggle objects. These toggle objects then become children of the radio object. A radio object must contain at least two radio buttons (toggle objects), only one of which may be selected (be in the TRUE state) at a time.

Toolbar

The toolbar provides buttons for the more commonly used form painter options. Click the right mouse button on a toolbar button to display balloon help for that button.

To do This	Use This Option	Use This Button
Open a form	File > Open Form	
Save a form	File > Save Form	
Clear a form	File > Close Form	
Cut selected objects to clip-board	Edit > Cut	

Toolbar Buttons

To do This	Use This Option	Use This Button
Copy selected objects to clip-board	Edit > Copy	
Paste from clip-board	Edit > Paste	
Duplicate selected objects	Edit > Duplicate	
Display a back-ground grid	Options > Show Grid	
Snap selected objects to grid	Options > Snap Grid	

Toolbar Buttons

The toolbar provides buttons for the more commonly used form painter options button on a toolbar button to display balloon help for that button.

To do this	Use this option	Use this button
Open a form	File:Open Form	
Save a form	File:Save Form	
Clear a form	File:Close Form	
Cut selected objects to clipboard	Edit:Cut	
Copy selected objects to clipboard	Edit:Copy	
Paste from clipboard	Edit:Paste	
Duplicate selected objects	Edit:Duplicate	
Display a background grid	Options:Show Grid	
Snap selected objects to grid	Options:Snap Grid	

Menu Bar

Options on the menu bar are as follows:

Use This Menu	To do This
File	Open and close forms and sessions, and exit the form painter.
Edit	Manipulate objects on a form.
Options	Use the form grid and set default properties for a class.
Objects	Select an object class (which may also be selected from the object bar), and change the class of an object.
Build	Use AutoGUI options, if available.
Window	Toggle the display of the form painter components on the desktop.
Help	Display on-line help.

Menu Bar Options

File

Allows you to manage your forms. Options on the menu relevant to the form painter are:

New Form

Creates a new form.

Close Form

Closes the current form. You are prompted to save any changes.

Open Form

Opens an existing form. This displays the File Name And Form Id Selection dialog.

If the Host Files checkbox is checked, the dialog will display files from the host, otherwise it will display files from the PC.

Enter or select the file name and record id for the form.

The Refresh button will regenerate the file list; you will have to select the file name again to display its records.

Save Form

Saves the current form.

Save Form As

Allows you to save the current form under a different name or to a different file. This displays the File Name And Form Id Selection dialog where you may enter the new form details.

Edit

Allows you to manage the objects on the form. It contains the following options (which may be applied to one or more selected objects):

Undo

Undoes the previous edit, move or size event. An undo stack maintains the history of the previous 50 events.

Redo

Redoes the last undo action. This is not recorded in the undo stack.

Cut

Deletes the selected objects from the form and places them on the clipboard. This is not the standard Windows clipboard, but rather a clipboard specific to the form painter.

Copy

Places the selected objects on the clipboard but does not delete them from the form.

Paste

Places the objects currently on the clipboard into the form.

Duplicate

Creates another copy of the selected objects, placing them on top of the original objects (and displacing them diagonally).

Delete

Deletes the selected objects from the form.

Align

Allows you to position two or more selected objects relative to each other.

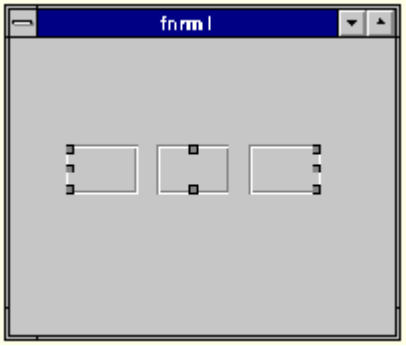
When you have selected the objects and chosen Align, the Alignment dialog is displayed:

This has the following radio buttons:

- Top. Aligns the tops of the objects with the highest one.
- Bottom. Aligns the bottoms of the objects with the lowest one.
- Left. Aligns the left sides of the objects with the left-most one.
- Right. Aligns the right sides of the objects with the right-most one.

Group

Allows you to construct a single object from two or more selected objects:



This allows you to act upon the objects as if they were a single object; for example, moving a group of objects to another position while preserving their relative spacing.

Ungroup

Deconstructs a grouped set of objects into individual objects.

Send Back

Moves the selected object to the bottom layer.

Each object on a form occupies a particular layer. When you create a new object, it occupies the layer nearest to the viewer and is placed on top of (and thus obscuring) any objects occupying the same space. The Send Back and Bring Front options allow you to correctly position objects (including grouped objects) on top of or behind other objects.

Bring Front

Moves the selected objects to the top layer.

Select All

Selects all objects on the form. This allows you to manipulate objects as a group.

Options

Provides the following additional options for manipulating a form:

Show Grid

Toggles the display of the grid on and off. The grid is a matrix of horizontal and vertical grid lines, which you may use to align objects on the form:

Snap Grid

Ensures that when you reposition or resize an object on the form it is automatically aligned to the nearest grid lines.

Grid Dimensions

Determines the spacing between dots on the grid. The larger the X and Y coordinates, the greater the spacing.

Defaults

Displays the default property values for a class, which may be the form or an object:

Initially this dialog displays the defaults for the formclass. You can select an object class from the selection list.

You may modify the displayed defaults and save them. The Save button will be shown in red if defaults have changed. Modified properties are also shown in red. To return a property to its original setting, select the property and click the Use Defaults button.

The defaults you specify are used whenever you create a new form or object of the modified class, whether manually or using AutoGUI.

Windows On Top

Forces the Color Palette, Object Bar and Properties & Events Window, if displayed, to be always placed on top of any forms on the screen.

Save Desktop on Exit

Saves the current configuration of the form painter. This includes details of grid settings and the display and positioning of the Color Palette, Object Bar and Properties & Events Window.

Objects

Allows you to select an object class and to change the class of an object. Object classes are:

- hostlabelclass
- hosttextclass
- hosteditclass
- hostpbclass
- hostcomboclass
- hosttoggleclass
- separatorclass
- rectangleclass
- hostradioclass

Object classes can also be selected from the Object Bar. See Object Bar.

Convert Object(s)

Allows you to change the class of certain types of objects. You need to first select the objects you wish to translate to another class of object, then choose the appropriate option:

- Convert To hosttextclass
- Convert To hostcomboclass
- Convert To hosttoggleclass
- Convert To hosteditclass

Window

Allows you to toggle the display of the various components of the form painter on the desktop:

- Color Palette
- Object Bar
- Properties & Events Window
- Toolbar

Help

Provides this on-line help on using the form painter.

Chapter 18 - Concepts of HostGUI

This model is ideal when you wish to GUItize a character application built with screen-based tools, that needs to run on both GUI workstations and character terminals, without having to re-structure the character application. HostGUI allows you to make GUI enhancements to the application that the character-based version cannot support. Like AutoGUI, this model assumes that only one form is active at a time.

HostGUI gives you full access to the host library, which includes subroutines for:

- storing forms on the host; TU.FORM.LOAD will automatically download the form to the PC (if the one stored in the PC disk cache is out of date)
- creating and manipulating new GUI objects that are not available in the character application
- creating a main window
- creating Windows style menus
- creating toolbars
- allowing the user to move between fields using the mouse rather than being restricted by any sequential ordering imposed by the character application
- setting the hourglass cursor (TU.FORM.HOURGLASSS)
- creating warning and error dialog boxes (TU.FORM.DIALOG)

The following host library header items are available for inclusion in HostGUI applications:

- ROC.H. Contains fundamental equates.

- SPECIAL.H. Contains equates relating to special keys, such as Esc and Enter.
- OBJECT.H. Contains equates for the pre-defined classes and their attributes.
- USER.INCLUDE.H. A header item for use by developers. This is initially empty, and allows developers to include their own subroutines.

For user input and output, use:

- TU.FORM.INPUT. This is essentially a more powerful replacement for the BASIC INPUT statement. It also supports input from textclass and editclass. See TU.FORM.INPUT.
- TU.FORM.UPDATEFIELD (or embedded cursor addressing). This is a replacement for the character mode CRT @(X,Y):NEWVALUE: method of updating fields. See TU_FORM_UPDATEFIELD.

Embedded Cursor Addressing

Embedded cursor addressing makes it easier to GUItize character applications. It also simplifies the maintenance of applications that must run on both character terminals and GUI workstations.

To illustrate how it works, consider the following character code which uses embedded cursor addressing to update the value of a field:

```
CRT @(X,Y):NEWVALUE
```

You can replace this with:

```
CRT GUI.LABEL:@(X,Y):NEWVALUE:GUI.END
```

The HostGUI run-time engine will check the current form for a field that begins at coordinates X,Y (properties char_col and char_row) and update that field.

Without embedded cursor addressing, you would have to update the string property of the field (whether label or text object) associated with coordinates X,Y.

Note that GUI.LABEL and GUI.END are defined in ROC.H as:

```
GUI.LABEL = CHAR(27):"_U3"
GUI.END = CHAR(27):"_X"
```

So, to ease maintenance of applications required to run on both character terminals and GUI workstations, you might try something like this (where GUI is a variable assigned a value by the host application):

Program Example

```
IF NOT(GUI) THEN
GUI.LABEL = NULL
   GUI.END = NULL
END
* ...
CRT GUI.LABEL:@(X,Y):NEWVALUE:GUI.END
```

An alternative to embedded cursor addressing is to call the TU.FORM.UPDATEFIELD subroutine. For example:

```
TU.FORM.UPDATEFIELD(X, Y, NEWVALUE, ERROR)
```

is equivalent to the preceding CRT statement. In both cases, the HostGUI run-time engine looks up the X,Y coordinates in its internal table and updates the corresponding field.

Comparison of HostGUI and Character Applications

This topic compares HostGUI and character applications in performing the common tasks identified below.

Initialization

Character

```
GUI.LABEL = ''
GUI.END = ''
```

HostGUI

```
CALL TU.INIT(ERROR)
CALL TU.FORM.LOAD("TUFORMS", FORMNAME, NULL, NULL, HFORM, ERROR)
```

Setting GUI.LABEL and GUI.END to NULL, while not strictly necessary, simplifies output for applications that must run on both GUI workstations and character terminals.

The TU.INIT subroutine must be called once (at application start-up) before calling any of the host library GUItization subroutines.

The TU.FORM.LOAD call loads a form.

Clear Screen

Character

* clear the screen prior to bringing up next record

CRT @(-1)

HostGUI

CALL TU.FORM.CLEAR(HFORM, ERROR)

Termination

Character

* clear the screen on exit

```
CRT @(-1)
```

HostGUI

```
CALL TU.FORM.KILLL(HFORM, ERROR)
CALL TU.TERMINATE(ERROR)
```

The CRT @(-1) statement clears the screen.

The TU.FORM.KILLL call removes a form.

The TU.TERMINATE must be called once when the application exits.

Program Input

Character

```
INPUT ANS:
```

HostGUI

```
CALL TU.FORM.INPUT(FIELDNAME, ANS, USERDATA, EVENT, ERROR)
```

The INPUT ANS: statement accepts user input. The INPUT statement is terminated by the user pressing Enter. The TU.FORM.INPUT subroutine is the HostGUI equivalent.

FIELDNAME is the name given to the field in the form painter. Instead of the field name, you may enter "X,Y" where X and Y are the character starting positions of the field (properties char_col and char_row).

Note that TU.FORM.INPUT gives more information than INPUT - along with ANS (the value of the field), the event string of the event terminating the input is returned in EVENT. This allows you to detect, for instance, that the user pressed the Esc key to terminate field entry. See Events for more information on event strings.

Program Output

Character

```
CRT @(X,Y):STR
```

Alternatively, when maintaining an application that must run in two environments, you could use:

Note

GUI.LABEL and GUI.END have been set to NULL at program start-up

```
CRT GUI.LABEL:@(X,Y):STR:GUI.END:
```

HostGUI

For GUI version the default (GUI) values given in ROC.H, are used

```
CRT GUI.LABEL:@(X,Y):STR:GUI.END:
```

-or-

As above, X and Y are the starting char. positions of field to be output

```
TU.FORM.UPDATEFIELD(X, Y, STR, ERROR)
```

The CRT statement simply writes STR to the screen.

The TU.FORM.UPDATEFIELD subroutine is an equivalent alternative that does not depend on embedded cursor addressing.

Chapter 19 - GUI Form Handling

These subroutines are used to control GUI forms.

GUI form handling subroutines are:

- TU.FORM.ADDOBJ. Adds an object to the current form at run time.
- TU.FORM.CLEAR. Clears a form of data.
- TU.FORM.DELOBJ. Removes an object from the current form.
- TU.FORM.FOCUS. Reveals a hidden form or moves a form in the form stack.
- TU.FORM.GETACTIVEFORM. Returns the handle to the active form.
- TU.FORM.GETATTR. Gets current field data.
- TU.FORM.GETDATA. Gets current form values for specified fields.
- TU.FORM.GETHANDLES. Returns the current stack of loaded form handles.
- TU.FORM.HELP. Displays help associated with the field.
- TU.FORM.INPUT. Gets input from an object on a form.
- TU.FORM.KILL. Destroys a form.
- TU.FORM.LOAD. Creates a GUI form and all its child objects.
- TU.FORM.SELECTLIST. Displays a list of options in a select list and allows one to be selected.
- TU.FORM.SETATTR. Sets field data.

- TU.FORM.SETDATA. Sets field values.
- TU.FORM.SETDEFAULTS. Controls the appearance of the default objects.
- TU.FORM.STATUSLINE. Displays information on the status line.
- TU.FORM.UPDATEFIELD. Updates the value of a cursor-address based field.
- TU.INIT. Initializes the GUI form and menu subroutines.
- TU.RESET. Resets all the TU GUI subroutines.
- TU.SERVER. Used at the host command line to re-start the host server after a fail has occurred.
- TU.SESSION. Sets a property of the session window.
- TU.TERMINATE. Terminates the GUI form subroutines.
- TU.TOOLBAR.EFFECT. Sets effects on toolbar buttons.
- TU.TOOLBAR.KILL. Used to remove a toolbar from a form.
- TU.TOOLBAR.LOAD. Attaches a toolbar to an existing form.

GUI Form Handling API

TU.FORM.ADDOBJ

TU.FORM.ADDOBJ(*fieldname*, *classname*, *attribs*, *values*, *error*)

Adds an object to the current form at run time.

fieldname

[P] The name to assign to the object.

classname

[P] The class of object to create.

attribs

[P] AM delimited list of attributes to set.

values

[P] Corresponding list of values (AM delimited).

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.CLEAR

TU.FORM.CLEAR(*formhandle*, *error*)

Clears a form of data.

formhandle

[P] The handle of the form returned by TU.FORM.LOAD. If null, the current form is cleared.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

TU.FORM.CLEAR removes any data in any text fields, toggles any toggles to their default values, unselects any selected sets, and clears labels with the dynamic property set to TRUE.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
CALL TU.FORM.LOAD("TUFORMS", "ACCOUNTS", NULL, "CL", FORMHANDLE, ERROR)

CALL TU.FORM.CLEAR(FORMHANDLE, ERROR)
```

See also

TU.FORM.LOAD

TU.FORM.DELOBJ

TU.FORM.DELOBJ(*fieldname*, *error*)

Removes an object from the current form.

fieldname

[P] The name of the field.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.FOCUS

TU.FORM.FOCUS(*handle*, *options*, *error*)

Reveals a hidden form (see the H option in TU.FORM.LOAD) or moves a form in the form stack.

handle

[P] The handle of the loaded form.

options

[P] This is usually null, but valid options are:

H keep Hidden (that is, move the form in the stack only)

L leave in stack (that is, reveal form only)

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.GETACTIVEFORM

TU.FORM.GETACTIVEFORM(*handle*, *error*)

Returns the handle to the active form.

handle

[R] The active form handle

If the active form is an external application, that is, the active form is not an SBClient form, the handle returned is 0 (zero).

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.GETATTR

TU.FORM.GETATTR(*fieldnames*, *attr*, *values*, *error*)

Gets current field data.

fieldnames

[P] The AM delimited object names whose values you wish to get.

attr

[P] The property you wish to get for each object.

values

[R] The AM delimited values requested in the order corresponding to the fieldnames.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

Unlike the companion TU.FORM.SETATTR, this subroutine does not allow multiple attributes for each object.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
INCLUDE TUBP USER.INCLUDE.H
* get the value of BACKGROUND color for fields "fred" and "bill".
* The values are returned in an AM delimited list namely RETURNCOLORS.
CALL TU.FORM.GETATTR("fred":AM:"bill", BACKGROUND, RETURNCOLORS, ERROR)
* eg RETURNCOLORS might be set to "red":AM:"blue" indicating that the

* background color of field "fred" is "red" and field "bill" is blue.
```

See also

TU.FORM.SETATTR

TU.FORM.GETDATA

TU.FORM.GETDATA(*fieldnames*, *values*, *error*)

Gets values from any object that has value attributes.

fieldnames

[P] The AM delimited object names whose values you wish to get.

values

[R] The corresponding AM delimited returned values.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

This subroutine is equivalent to TU.FORM.GETATTR(fieldnames, attr, values, error).

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* get values of fields "fred" and "bill" and return them in an
* AM delimited string RETVAL

CALL TU.FORM.GETDATA("fred":AM:"bill", RETVAL, ERROR)
```

See also

TU.FORM.SETDATA, TU.FORM.GETATTR

TU.FORM.GETHANDLES

TU.FORM.GETHANDLES(*handles*, *error*)

Returns the current stack of loaded form handles.

handles

[R] A multi-valued list of form handles, the most recently loaded form first.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.HELP

TU.FORM.HELP(*fieldname*, *error*)

Displays help associated with the field. The field, referenced in fieldname, must have its help_topic property set.

fieldname

[R] The name of the object can be specified as x,y, where x is the column and y is the row.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.INPUT

TU.FORM.INPUT(*fieldname*, *returndata*, *userdata*, *event*, *error*)

Gets input from an object on a form.

fieldname

[P] The object name or xcoord,ycoord where xcoord and ycoord are the starting character positions of the field.

returndata

[R] The current value of the field on which the terminating event occurred.

userdata

[R] The user_data value (if any) of the object the event took place in.

event

[R] The event string describing the event that terminated the input:

Event<1>The event type.

Event<2>fieldname the event happened to.

Event<3> …The rest of the event string.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

This subroutine is akin to BASIC's INPUT subroutine and can be used in its place. Don't use this subroutine in a multi-active form environment; use TU.FORM.EVENTLOOP instead.

The subroutine returns the current contents of the field and the event that occurred which terminated the subroutine.

Many properties of fields, such as length, autoreturn and so on are specified in the form painter itself, rather than here.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
CALL TU.FORM.LOAD("TUFORMS", "ANDREW2", NULL, "L", FORMHANDLE, ERROR)
* input the field with logical name "fred" returning its value in
* RETURNDATA and the description of the event that terminated the
* input subroutine in EVENTSTR
CALL TU.FORM.INPUT("fred", RETURNDATA, USERDATA, EVENTSTR, ERROR)
* or using cursor addresses to identify field

CALL TU.FORM.INPUT("10,12", RETURNDATA, USERDATA, EVENTSTR, ERROR)
```

See also

TU.FORM.EVENTLOOP, Embedded Cursor Addressing and Events.

TU.FORM.KILL

TU.FORM.KILL(*formhandle*, *error*)

Destroys a form.

formhandle

[P] The handle to the form returned by TU.FORM.LOAD. If null, the current form is destroyed.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

Forms created with TU.FORM.LOAD should be removed with TU.FORM.KILL.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
CALL TU.FORM.LOAD("TUFORMS", "FRED", NULL, "CL", FORMHANDLE, ERROR)

CALL TU.FORM.KILL(FORMHANDLE, ERROR)
```

See also

TU.FORM.LOAD

TU.FORM.LOAD

TU.FORM.LOAD(*filename*, *formname*, *parenthandle*, *options*, *formhandle*, *error*)

Creates a GUI form and all its child objects.

filename

[P] The name of the file containing the form definition.

formname

[P] The name of the definition.

parenthandle

[P] The handle of the form's parent. Null indicates the default parent. Zero indicates no parent.

options

[P] An array of characters indicating optional transfer details. Valid values are:

- H Hide. Create the form, but don't display it.
- L Local. The filename and formname reside locally on the PC.
- S Single. Any previous form will be destroyed before this form is created. That is, only one form is displayed at a time.
- M Main Window. This form will become the main window of the application. The main window will contain a status line at the bottom for displaying messages, and all forms/menus that do not specify a parent will be parented to the main window. Forms specified as M should not contain any objects – any objects in the form definition will be ignored.
- F removes the Forced input method.
- O default Objects. This allows the automatic creation of objects if none are present at the desired coordinates.
- B default Button. Indicates that a default button is present and that Tab is used to move through the form.
- I Use a true Windows MDI form. This must be used with the M option. This will restrict any children forms to within the MDI frame.

formhandle

[R] The handle of the newly-loaded form.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

When loading a host-based form, this subroutine negotiates time/date stamps between the host and the client. If the client version is out of date, the definition is loaded from the host, stored on the

client and form creation continues as normal. This form caching is transparent to the application developer.

TU.FORM.KILL should be used to remove the form.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* load a local form ("L"), created by AutoGUI residing in SBDesktop file
* TUFORMS, record BILL, and return its handle in FORMHANDLE
CALL TU.FORM.LOAD("TUFORMS", "BILL", NULL, "L", FORMHANDLE, ERROR)
* load a form residing in a file on the host
* FORMS, record JOE, and return its handle in FORMHANDLE
* the "C" means that this form will be used in an event-driven BASIC

* application
CALL TU.FORM.LOAD("FORMS", "JOE", NULL, "C", FORMHANDLE, ERROR)
* load a form residing on the host file
* FORMS, record BILL, and return its handle in FORMHANDLE
* this is a HostGUI-style application

CALL TU.FORM.LOAD("FORMS", "BILL", NULL, NULL, FORMHANDLE, ERROR)
```

See also

TU.FORM.KILL

TU.FORM.SELECTLIST

TU.FORM.SELECTLIST(*title*, *recordlist*, *width*, *depth*, *coords*, *background*, *foreground*, *options*, *rtnval*, *error*)

Displays a list of options in a select list and allows one to be selected.

title

[P] Title to appear on the form containing the list. title can be followed by a VM delimited list of column headings.

recordlist

[P] Multivalued list of records to display in the list. For multiple records each column value is separated by a subvalue mark.

width

[P] Width per column (in characters). If differing widths are required for the columns, specify a width for each column, separated by value-marks.

depth

[P] Depth of list (in characters).

coords

[P] Co-ordinates of where list is to appear (col:VM:row).

background

[P] Background color of list. See OBJECT.H for a list of colors.

foreground

[P] Foreground color of list. See OBJECT.H for a list of colors.

options

[P] Options are:

M Multi-column. This will create separate list objects for each column of data. You can have multi-columns without specifying the M option. In this case one list will be created with tab stops being used between columns. However, if some column data will exceed the tab stop in length, the M option should be used.

F use Fixed pitch font rather than proportional.

rtnval

[R] The selected option. This is an AM delimited list. Property 1 is the record number in the list and property 2 is the record string.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.SETATTR

TU.FORM.SETATTR(*fieldnames*, *attrs*, *values*, *error*)

Sets field data.

fieldnames

[P] The AM delimited object names whose values are to be set.

attrs

[P] The AM delimited attributes to be set. All of these attributes are set for each object specified in fieldnames.

values

[P] The corresponding AM delimited values.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* set BACKGROUND color to green for fields "fred" and "bill"
CALL TU.FORM.SETATTR("fred":AM:"bill", BACKGROUND, GREEN, ERROR)
* set BORDER.WIDTH to 0 and BACKGROUND color to green
* for fields "fred", "bill" and "jack"

CALL TU.FORM.SETATTR("fred":AM:"bill":AM:"jack", BORDER.WIDTH:AM:BACKGROUND,
0:AM:GREEN, ERROR)
```

See also

TU.FORM.GETATTR

TU.FORM.SETDATA

TU.FORM.SETDATA(*fieldnames*, *values*, *error*)

Sets form field values.

fieldnames

[P] The AM delimited object names whose values are to be set.

values

[P] The corresponding AM delimited values.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* set value to "hello" for field "fred" and "world" for field "bill"

CALL TU.FORM.SETDATA("fred":AM:"bill", "hello":AM:"world", ERROR)
```

See also

TU.FORM.GETDATA, TU.FORM.SETATTR

TU.FORM.SETDEFAULTS

TU.FORM.SETDEFAULTS(*type*, *special*, *attr*, *vals*, *error*)

Controls the appearance of the default objects (see the O option in TU.FORM.LOAD).

type

[P] Options are:

I Input field

O Output label

special

[P] Value in pixels for character position. For example: 8:VM:18.

attr

[P] AM delimited list of attributes, such as FOREGROUND, BORDER_STYLE and so on.

vals

[P] AM delimited list of corresponding values, such as LIGHT.GREY, LOWERED and so on.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.STATUSLINE

TU.FORM.STATUSLINE(*string*, *error*)

Displays information on the status line. The status line is created when loading a "main window" form – the M option is specified in TU.FORM.LOAD.

string

[P] The string to display in the status line. Null will clear the status line.

TU.FORM.UPDATEFIELD

TU.FORM.UPDATEFIELD(*col*, *row*, *value*, *error*)

Updates the value of a cursor-address based field.

col

[P] The column of the field to be updated.

row

[P] The row of the field to be updated.

value

[P] The new value of the field to be updated.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

This subroutine is only used for HostGUI applications. It is a way of directly updating the value of cursor-address based fields. This method may be used instead of the more common GUI.LABEL:@(X,Y):value:GUI.END embedded cursor addressing method.

This method is useful when value is a large string, as it will employ file transfer protocols if necessary.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
CALL TU.FORM.LOAD("TUFORMS", "FRED", NULL, "L", FORMHANDLE, ERROR)
* update the field at x=20,y=40 (col=20, row=40) with the value "hello"
CALL TU.FORM.UPDATEFIELD(20, 40, "hello", ERROR)
* do the same using embedded cursor addressing

CRT GUI.LABEL:@(20,40):"hello":GUI.END
```

See also

Embedded Cursor Addressing

TU.INIT

TU.INIT(*error*)

Initializes the GUI form and menu subroutines.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

TU.INIT should be called once at the beginning of any application that calls the form or menu subroutines (that is, subroutines in the host library that start with TU.FORM or TU.MENU).

Program Example

```
CALL TU.INIT(ERROR)
```

See also

TU.TERMINATE

TU.RESET

TU.RESET

This is a host command that resets all the TU GUI subroutines. It is equivalent to calling TU.TERMINATE.

TU.SERVER

TU.SERVER

Used at the host command line to re-start the host server after a fail has occurred.

TU.SESSION

TU.SESSION(*attrib*, *value*, *error*)

Sets an attribute of the session window.

attrib

[P] Attribute to set. Useful attributes to set are FRONT and BACK.

value

[P] The value of the new attribute.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine. You need to have called TU.INIT prior to calling this subroutine.

TU.TERMINATE

TU.TERMINATE(*error*)

Terminates the GUI form subroutines.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

TU.TERMINATE should be called once at the end of any application that uses the form or menu subroutines (that is, subroutines in the host library that start with TU.FORM or TU.MENU).

Any forms, objects and so on created with the form subroutines will also be destroyed.

Program Example

```
CALL TU.TERMINATE(ERROR)
```

See also

TU.INIT

TU.TOOLBAR.EFFECT

TU.TOOLBAR.EFFECT(*handle*, *buttons*, *effect*, *error*)

Sets effects on toolbar buttons.

handle

[P] The toolbar handle.

buttons

[P] A VM delimited list of button numbers, from left to right (1 is the first button).

effect

[P] This can be ENABLED or DISABLED.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.TOOLBAR.KILL

TU.TOOLBAR.KILL(*handle*, *error*)

Used to remove a toolbar from a form. Killing the form itself will also result in the toolbar being removed.

handle

[P] The toolbar handle (returned by TU.TOOLBAR.LOAD).

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.TOOLBAR.LOAD

TU.TOOLBAR.LOAD

TU.TOOLBAR.LOAD(*hostfile*, *toolbarname*, *parent*, *options*, *handle*, *error*)

Attaches a toolbar to an existing form.

hostfile

[P] The name of the host file containing the toolbar definition. Instead of supplying a file name here, you may supply a toolbar definition specification (see Note below).

toolbarname

[P] The id of the host record containing the definition.

parent

[P] The form handle (returned via TU.FORM.LOAD) of the form to which you wish to attach the toolbar.

Note

If the parent handle passed is NULL and a main window is present, the toolbar will attach to the main window. If there is no main window, the toolbar will attach itself to the last form loaded.

options

[P] Options are:

L Local. The toolbar definition resides locally on the PC.

handle

[R] The returned toolbar handle (used by TU.TOOLBAR.KILL).

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

A toolbar definition has the following format:

Attribute	Description
1	Toolbar (time/date stamp).
2	Buttons: multi-value list, each multi-value containing five subvalues: subvalue 1:type 1 for a Windows application. type 2 for a host value. subvalue 2:action: type 1 for an application path (for example, WINFILE.EXE). type 2 for the string to return or macro. macro: {F#} where # is the function key number. {ESC} for escape key press. subvalue 3:bitmap or string for the button. subvalue 4:the tooltip help text to display when the mouse pointer is positioned over the button. subvalue 5:set this subvalue to 1 to display the button in the 97 style (no 3D border around the button).

Toolbar Definition Format

See also

TU.TOOLBAR.KILL

Chapter 20 - GUI Menu Handling

These subroutines are used to control GUI menus.

GUI menu handling subroutines are:

- TU.MENU.EDIT. This is a host command that allows you to create and edit menu definition records.
- TU.MENU.EFFECT. Changes a menu's characteristics.
- TU.MENU.INPUT. Inputs from a menu.
- TU.MENU.KILLL. Destroys a previously created menu.
- TU.MENU.LOAD. Loads a menu.

GUI Menu Handling

TU.MENU.EDIT

TU.MENU.EDIT

This is a host command that allows you to create and edit menu definition records.

TU.MENU.EFFECT

TU.MENU.EFFECT(*menuhandle*, *ids*, *effects*, *error*)

Changes a menu's characteristics.

menuhandle

[P] The handle of the menu being updated. It is returned when loading a menu with TU.MENU.LOAD.

ids

[P] A VM delimited list of menu ids to set.

effects

[P] A corresponding VM delimited list of effects to set. Each effect may be one of: "disabled", "enabled", "checked", "unchecked".

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* load a type 2 (windows) menu
CALL TU.MENU.EFFECT(MHANDLE, 'REP1':VM:'REP2', "disabled":VM:"enabled", ERROR)
```

See also

SUBROUTINE, TU.MENU.KILL

TU.MENU.INPUT

TU.MENU.INPUT(*returnval*, *userdata*, *event*, *error*)

Inputs from a menu.

returnval

[R] The unique menu id identifying which menu option was selected.

userdata

[R] The USER.DATA value of the menu (if any).

event

[R] The event string: event:AM:handle:AM:option:AM:params

where:

- event is the event that happened: menu, select or special
- handle is the object handle.

For menu events:

option is the menu option name.

params are any additional parameters.

For select events:

option is the menu option number.

params are any additional parameters.

For special events:

option is the special key number.

params are the Shift, Ctrl and Esc states (AM list).

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

If the user presses Esc while on a type 2 menu option, the menu id will be correctly returned in returnval, and the event parameter will indicate a special key event with the third attribute of the event string set to the escape key.

The BASIC program can respond to a close event in TU.MENU.INPUT. The form definition can specify callbacks for example, to trap for a close event on a form specify close~hCb~{ESC} as the callback attribute in the form definition. If the user clicks on the close button of the form, EXIT will be returned in the retdata of TU.MENU.INPUT and the EVENT variable will contain close:SVM:handle. If no close callback is specified in the form definition, the close event will still be returned in EVENT but retdata will be null.

See also

TU.MENU.LOAD, TU.MENU.KILL

TU.MENU.KILL

TU.MENU.KILL(*menuhandle*, *error*)

Destroys a previously created menu.

menuhandle

[P] The handle of a previously created menu, returned by TU.MENU.LOAD.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

See also

TU.MENU.LOAD, TU.MENU.INPUT

TU.MENU.LOAD

TU.MENU.LOAD(*hostfile*, *menuname*, *formhandle*, *options*, *menuhandle*, *error*)

Loads a menu.

hostfile

[P] The name of the file containing the menu definition. Instead of supplying a file name here, you may supply a menu definition specification (see Note below).

menuname

[P] The id of the host record containing the definition.

formhandle

[P] The handle of the menu's parent form. A null value will set SBClient as the menu's parent. For type 2 menus, formhandle is the form to which you wish to attach the menu (this is mandatory). In HostGUI, if there is a "main window" and a null value is supplied, the parent will be the "main window".

options

[P] Options are:

H Hide. Create the menu, but don't display it

L Local. The menu definition resides locally on the PC

S Stop Flicker. Stops flicker when changing menus in MDI mode. If the S option is specified, only one turunmenuclass is created. Each time a menu is loaded this class is updated with the new menu specification. It is not necessary to do a TU.MENU.KILLL before loading a new menu. TU.MENU.KILLL does not restore the previous menu automatically.

menuhandle

[R] The newly created menu's handle.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

A menu definition specification has the following format:

Attribute	Description
1	Menu (time/date stamp).
2	Menu type: Type 1. Windows-style menu. Type 2. These behave like a typical character-based menu; they pop-up and stay on the screen until explicitly removed.
3	The menu's character coordinates: col:VM:row. Used for type 1 menus only.
4	Title. Used for type 1 menus only.
5	A VM delimited list of menu options. Each option may have up to four subvalues as follows: • menuid. Mandatory. • description. Mandatory. • parentid. Optional. Used for type 2 menus only. The menuid of the sub-menu's parent.
6	A VM delimited list of the initial disabled menu items. Used in type 2 menus only.
7	A VM delimited list of the initial checked menu items. Used in type 2 menus only.

menu definition format

Menus are not destroyed until a TU.MENU.KILL is executed.

Program Example

```
INCLUDE TUBP SPECIAL.H
INCLUDE TUBP ROC.H
INCLUDE TUBP OBJECT.H
* load a type 1 (windows) menu
MENUDEFN = 'MENU':AM:1
MENUDEFN<4> = 'FILE':SVM:'&File':VM:'EDIT':SVM:'&Edit':VM:'REP':SVM:'&Reports'
MENUDEFN<4, -1> = 'OPEN':SVM:'&Open':SVM:'FILE'
MENUDEFN<4, -1> = 'SAVE':SVM:'&Save':SVM:'FILE'
MENUDEFN<4, -1> = 'REP1':SVM:'&Report 1':SVM:'REP'
MENUDEFN<4, -1> = 'REP2':SVM:'&Report 2':SVM:'REP'
CALL TU.MENU.LOAD(MENUDEFN, '', HFORM, '', MHANDLE0, ERROR)
* load a type 2 (pop-up) menu
MENUDEFN = 'MENU':AM:2:AM:2:VM:5
MENUDEFN<4> = 'O1':SVM:'Option 1':VM:'O2':SVM:'Option 2':VM:'O3':SVM:'Option 3'
MENUDEFN<4,-1> = 'O4':SVM:'Option 4':VM:'O5':SVM:'Option 5':VM:'O6':SVM:'Option
6
CALL TU.MENU.LOAD(MENUDEFN, 'XX', NULL, NULL, MHANDLE1, ERROR)
```

See also

TU.MENU.INPUT, TU.MENU.KILL

Chapter 21 - Misc GUItization

These subroutines provide additional GUItization functionality.

Miscellaneous GUItization subroutines are:

- TU.FORM.DIALOG. Displays a dialog box.
- TU.FORM.HOURGLASS. Toggles on or off the Windows hourglass cursor.
- TU.FORM.OPENDOS. Displays a Windows "Open DOS File" dialog box.
- TU.FORM.SAVEDOS. Displays a Windows "Save DOS File" dialog box.
- TU.FORM.SMARTHOURGLASS. Toggles on and off the Smart Cursor functionality.
- TU.QUERY.TERMINAL.WINDOW. Gets the mode as to what to do if an unexpected character is received while in GUI mode.
- TU.SHOW.TERMINAL.WINDOW. Sets the mode as to what to do if an unexpected character is received while in GUI mode.

TU.FORM.DIALOG

TU.FORM.DIALOG(*returnvalue*, *dialogtype*, *message*, *buttons*, *returnvalues*, *dialogtitle*, *error*)

Displays a dialog box.

returnvalue

[R] The value selected by the user.

dialogtype

[P] The type of dialog. This specifies the type of graphic to be displayed in the dialog. Valid values are: DLG.ERROR, DLG.QUESTION, DLG.INFO or DLG.WARNING.

message

[P] The textual message to display inside the dialog box.

buttons

[P] A VM delimited list of button strings. Null indicates an OK button only.

returnvalues

[P] A VM delimited list of values each button will return when clicked.

dialogtitle

[P] The dialog form title.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

TU.MENU.DIALOG is a pre-defined subroutine to pop up a message dialog containing buttons with which the user may select an option. Possible uses are for information dialogs, error messages, prompting for OK and Cancel type actions or for selecting from a list of options such as Create, Amend, View and Quit.

TU.FORM.HOURGLASS

TU.FORM.HOURGLASSS(*hourglassstate*, *error*)

Toggles on or off the Windows hour glass cursor.

hourglassstate

[P] A TRUE/FALSE value to toggle the hour glass on or off. You should set this to TRUE when you know an operation will take a while and you want to prevent users from clicking the mouse or using the key board.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.FORM.OPENDOS

TU.FORM.OPENDOS(*initialfilter*, *startpath*, *filetypes*, *returnpath*, *error*)

Displays a Windows "Open DOS File" dialog box.

initialfilter

[P] The starting filter for the dialog window. This can be any DOS filter (for example, *.BMP, NEWFILE.DOC, *.*).

startpath

[P] The initial DOS path for the dialog window. If null, the dialog will use either the current directory, or, if the dialog has previously been invoked, the last path used by the dialog.

filetypes

[P] An SM delimited list of file types to be shown in the Types combo box. The format is:

description:SM:filter:SM:description:SM:filter …

For example, "Bitmaps":SM:"*.BMP":SM:"All Files":SM:"*.*"

returnpath

[R] The selected DOS file. This will be null if no file was selected.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

This subroutine will display a Windows "Open DOS File" dialog box, as found using File:Open in most Windows applications. You can supply the filters, initial directory and any starting filter or file name to be used. The user can then browse the directory structures and select a DOS file or cancel the operation. If the user cancels then a null file name will be returned; otherwise the full DOS path and file name are returned to the application.

TU.FORM.SAVEDOS

TU.FORM.SAVEDOS(*initialfilter*, *startpath*, *filetypes*, *returnpath*, *error*)

Displays a Windows "Save DOS File" dialog box.

initialfilter

[P] The starting filter for the dialog window. This can be any DOS file name or filter. For example, *.BMP, NEWFILE.DOC, *.*.

startpath

[P] The initial DOS path for the dialog window. If null, the dialog will use either the current directory, or, if the dialog has previously been invoked, the last path used by the dialog.

filetypes

[P] An SM delimited list of file types to be shown in the Types combo box. The format is:

description:SM:filter:SM:description:SM:filter …

For example, "Bitmaps":SM:"*.BMP":SM:"All Files":SM:"*.*"

returnpath

[R] The selected DOS file. This will be null if no file was selected.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

Note

This subroutine will display a Windows "Save DOS File" dialog box, as found using File:SaveAs in most Windows applications. You may supply the filters, initial directory and any starting filter or file name to be used. The user can then browse the directory structures and select or enter a DOS file or cancel the operation. If the user cancels then a null file name will be returned; otherwise the full DOS path and file name are returned to the application.

TU.FORM.SMARTHOURGLASS

TU.FORM.SMARTHOURGLASSS(*flag*, *error*)

Used to turn on/off the Smart Cursor functionality. This means that the Windows hour glass will be displayed automatically during the times when no user activity is required. For example, when loading a form, the hour glass will be displayed. When an input is required, the hour glass will be removed. After an input, the hour glass will re-appear.

flag

[P] A TRUE/FALSE value to toggle this method on or off. A value of -1 is similar to FALSE but throws away any keyboard buffering.

error

[R] Indicates the success (zero) or failure (non-zero) of the subroutine.

TU.QUERY.TERMINAL.WINDOW

TU.QUERY.TERMINAL.WINDOW(*val*, *status*)

TU.QUERY.TERMINAL WINDOW gets the mode as to what to do if an unexpected character is received while in GUI mode. (that is, whether to bring up the character screen or not)

Val

[R] The returned value can be

0 Never show character window

1 Always show character window

2 Show character window on predefined strings only

Status

[R] A non zero error codes indicates a failure.

Note

This subroutine corresponds to the Setup GUI Parameters Screen in SBClient

See also

TU.SHOW.TERMINAL.WINDOW

TU.SHOW.TERMINAL.WINDOW

TU.SHOW.TERMINAL.WINDOW(*option*, *status*)

TU.SHOW.TERMINAL WINDOW sets the mode as to what to do if an unexpected character is received while in GUI mode. (that is, whether to bring up the character screen or not)

Option

[P] The value to be passed can be:

0 Never show character window

1 Always show character window

2 Show character window on predefined strings only

Status

[R] A non zero error codes indicates a failure.

Note

This subroutine corresponds to the Setup GUI Parameters Screen in SBClient.

See also

TU.QUERY.TERMINAL.WINDOW

PART 5: APPENDICES

Appendix A - Demonstration Programs

This appendix describes the demonstration programs that are supplied with SBClient.

Demonstration Programs

SBClient's Host Library includes a number of programs demonstrating various aspects of the library's functions. The programs are copied to the file DEMOBP during host installation.

The programs can be run by typing RUN DEMOBP PROGRAMNAME at the command line prompt. Many of the programs can be accessed from the Application demonstration program. Type RUN DEMOBP APPLICATION.DEMO (or APPLICATION.DEMO.MDI) at the command line.

Before running the demonstration programs, you should run the program BUILD.CARS. Type RUN DEMOBP BUILD.CARS at the command line.

The following briefly describes the programs:

APPLICATION.DEMO & APPLICATION.DEMO.MDI

This demonstrates a small built application, including a MAINWIN form, a toolbar, a menu specification and a status bar. The .MDI variant includes audio and video demonstrations if you have the appropriate software on your system.

BUILD.CARS

This program builds a small database used by many programs in the demonstration suite. You should run this before any other programs.

CARS.HOSTGUI & CARS.HOSTGUI.PLUS

These programs demonstrates a HostGUI screen created using TU.FORM.PAINTER and run from a program designed to suit legacy applications built using a linear programming methodology. CARS.HOSTGUI.PLUS demonstrates a more complex screen. See HostGUI.

DDE.DEMO

This program demonstrates the ability to launch and control a Windows application from a host program running in an SBClient session.

DIRECTORY.DEMO

This program demonstrates the DOS directory control functions in the host library. A directory is checked for on the local hard disk, created and deleted.

EXCEL

This program demonstrates the ability to select data from the host, format it and send it to Microsoft Excel.

EXCEL.FORMULA

This program demonstrates the ability to send data to Microsoft Excel, select a range and apply a formula, using DDE macro functionality in the library.

EXCEL.GRAPH

This program demonstrates the ability to select data from the host, format it, send it to Microsoft Excel and create a graph based on the data transferred.

FILE.DEMO

This program demonstrates the DOS file control functions in the host library. A file is checked for on the local hard disk, created and deleted.

IMAGE.DEMO

This program demonstrates the ability to display a bitmap image in a window. The program prompts for a path to a bitmap image.

L123

This program demonstrates the ability to select data from the host, format it and send it to Lotus 123.

L123.GRAPH

This program demonstrates the ability to select data from the host, format it, send it to Lotus 123 and create a graph based on the data transferred.

LAUNCH

This program demonstrates the library functions that can launch a Windows application, check for its existence, and close the application.

MACRO

This program demonstrates the library function that allows launch of applications and full DDE control. The program loads Notepad, enters text with spelling errors, and corrects the errors, using macro strings.

PRINTER.DEMO

This program demonstrates the library functions which allow the current Windows printer setup to be retrieved, and new default to be set.

SELECTLIST1.DEMO, SELECTLIST2.DEMO, SELECTLIST3.DEMO & SELECTLIST4.DEMO

These programs demonstrate different ways of building and displaying a selectlist box.

VBXDEMO

This program demonstrates the ability to add VBX controls to an SBClient form. A grid is created, and values in the grid may be modified.

VER

This program demonstrates the host library function that returns the current version details of the installed SBClient Host Library. This should be the same as the SBClient version number.

VIDEO.DEMO

This program demonstrates the ability to display and run a video file from a host program. A video player is displayed. You will need to have appropriate software/drivers installed on your system to run this program.

WINDOW.DEMO

This program demonstrates the character window drawing library functions. A number of character Windows are drawn with a save of the screen occurring after each draw. The process is then reversed.

WORD

This program demonstrates the ability to select data from the host, format it, send it to Word, then have word format the data in a table.

Appendix B - GUI Classes and Their Attributes

This appendix describes the GUI classes and their attributes.

Introduction

An object class defines a type of object (for example, a label), by specifying what its attributes are (for example, justification, coordinates and so on) and how they behave. An attribute's behavior is determined by the set of values (left, right and so on) that may be assigned to it when an instance of the object is created.

Attribute delimiters vary according to where they are used:

- In the form painter the attribute delimiter is a semicolon. This is what is used in this appendix.
- In code it is a VM.
- In a form definition it is an SM.

The standard text, edit, label, toggle, radio and optionlist classes have been replaced in the form painter by new object classes (with the prefix host). These new classes inherit from the standard classes, described below, but add additional attributes required by AutoGUI and HostGUI. See Attributes and New GUI Attributes.

GUI Class	Attributes
form	background, border_color, border_style, border_width, coordinates, dimensions, drawable, icon, maximize, maximizable, minimize, minimizable, title, title_bar, tile
label	background, border_color, border_style, border_width, coordinates, dimensions, drawable, emphasized, font, tile, graphic, justification, string, foreground
text	background, border_color, border_style, border_width, coordinates, dimensions, drawable, editable, font, foreground, special_key_set, justification
bpbutton	background, border_width, coordinates, dimensions, down_graphic, down_string, drawable, emphasized, font, foreground, graphic, justification, string, tile

GUI Class Attributes

GUI Class	Attributes
rectangle	background, border_color, border_style, border_width, coordinates, dimensions, drawable, tile
separator	border_style, coordinates, dimensions, direction, drawable, foreground
toggle	background, border_color, border_style, border_width, coordinates, dimensions, drawable, emphasized, false_graphic, font, foreground, special_key_set, string, state, style, tile, toggle_ border_width, toggle_size, true_graphic
radio	radio_set, background, border_color, border_style, border_width, dimensions, drawable, emphasized, tile
combobox	background, border_color, border_style, border_width, coordinates, dimensions, drawable, font, foreground, record_set, num_lines, scroll, special_key_se

GUI Class Attributes

Standard Attributes

Standard attributes used by the properties in the form painter are as follows:

back

A Boolean value forcing the object to be displayed at the back of existing objects. Possible values are TRUE and FALSE. Unlike other attributes, setting this attribute to TRUE performs an immediate action (sending the object to the back of other objects), and does not hold its value. Setting this attribute to FALSE therefore has no affect.

background

The background color of the object. This attribute is ignored if a bitmap or tile is being used.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

border_color

The color of the border to be drawn around the object.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

border_style

The type of border to be drawn around the object. Possible values are:

raised	The border is shaded so that the object appears to be elevated from the background surface.
lowered	The border is shaded so that the object appears to be sunk into the background surface.
flat	The border is flat; neither raised or lowered.

msstyle — The border is set two pixels wide and shaded to appear to be sunk into the background surface. This style emulates Microsoft's standard three-dimensional text input field border.

border_width

The width of the border to be drawn around the object. The value is given in pixels. Borders encroach upon the dimensions of the object, and their width should be considered when sizing an object and positioning children.

coordinates

The pixel address, relative to the object's parent, of the top-left corner of the object. Its format is x;y.

cursor

Defines a bitmap (as xxx.CUR) for the cursor used by an object. This is a system dependent attribute, and is normally an arrow, but may change from one object to another.

dimensions

The pixel width and pixel height of the object. Its format is width;height.

direction

The direction of the separator line. Possible values are HORIZONTAL and VERTICAL.

down_graphic

The graphic to be displayed when a button is displaying a bitmap, and the push button is in the pressed state. If this is not set, the bitmap defined in the graphic attribute is used but offset two pixels right and two pixels down.

down_string

The string to be displayed when a button is displaying a text string, and the push button is in the pressed state. This defaults to the value of string.

drawable

A Boolean value determining whether the object is rendered visible. An object is rendered provided it, and all ancestors, are renderable.

editable

A Boolean value determining whether the text in the object can be edited. Possible values are TRUE and FALSE.

emphasized

A Boolean value determining whether the object is capable of receiving input events (for example, mouse, keyboard and so on). Possible values are TRUE and FALSE. If FALSE, the object is displayed in the system-defined de-emphasized state. For example, a menu option might be shown grayed out if it is currently of no relevance to the user.

false_graphic

The bitmap to place on a toggle when in the FALSE state.

font

The font used when an object is displaying a string. Its format is:

fontfamily*pointsize*style*effect*pitch*charset

For example:

Arial*8*bold*normal*pitch_variable*charset_ansi

foreground

The color of text in the object or the on-bit value if the object is displaying a bitmap.

Specify a particular color by entering the appropriate values for the three primary colors: red,green,blue. For example: 3;20;210 creates a color made up of the specified hues of red (3), green (120) and blue (210). Each number must be in the range 0-255.

front

A Boolean value which forces the object to be displayed at the front of existing objects. Possible values are TRUE and FALSE. Unlike other attributes (apart from the back attribute), setting this to TRUE performs an immediate action (sending the object to the front of other objects), and does not hold its value. Setting this attribute to FALSE therefore has no affect.

graphic

The name of a file containing bitmap information. The format of this file is window-system specific.

icon

The icon file (xxx.ICO) used to represent the minimized form.

item_set

A multi-value list of records to appear in the list. This attribute accepts the standard set notation modifiers (APPEND, DELETE, index and value).

justification

The justification used when displaying either a string or graphic. Possible values are: left, center, right, top_left, top_center, top_right, bottom_left, bottom_center, bottom_right.

maximize

A Boolean value determining whether the form is maximized. To restore a maximized form to its previous size, set this attribute to FALSE.

maximizable

A Boolean value determining whether a maximize control is available for the form.

minimize

A Boolean value determining whether the form is minimized. To restore a minimized form to its previous size, set this attribute to FALSE.

minimizable

A Boolean value determining whether a minimize control is available for the form.

num_lines

The number of lines that will be displayed in the optionlist dropdown box.

scroll

A Boolean value determining whether a vertical scrollbar is available.

special_key_set

A value-mark delimited list of the special keys that generate a special event.

state

A Boolean value determining whether the object is toggled.

string

The string of characters to appear on the object. The type of label (character or bitmap) is determined by whether the string or graphic attribute has been most recently set.

style

Defines the style of the toggleclass object. The following predefined styles are available:

- CHECK. Displays a check mark when selected.
- CROSS. Displays a cross when selected.
- BUTTON. Is displayed recessed when selected.
- DIAMOND. Displays a filled-in diamond when selected.
- CIRCLE. Displays a filled-in circle when selected.
- BITMAP. Displays true and false graphic bitmaps on a button.
- MSCROSS. Displays a Windows style three dimensional cross.
- MSCIRCLE. Displays a Windows style three dimensional circle. Only toggle sizes of 13, 15 and 17 are supported for this style.

The first three toggles are generally used for checkboxes; where more than one checkbox can be toggled on at a time.

tile

The name of a file containing a bitmap to be used as the background for the object.

title

The text to appear as in the title bar for the window.

title_bar

A Boolean value determining whether a form has a title bar.

toggle_border_width

The width of the borders in CHECK, BUTTON, DIAMOND and BITMAP style toggles.

toggle_size

The width and height of the toggle button in pixels. Since the toggle button is a perfect square, only one dimension is specified.

true_graphic

The bitmap to place on the button and check toggle when in the TRUE state. If the style attribute is set to CHECK, the default for the true_graphic attribute becomes CHECK_BMP.

Additional GUI Attributes

Additional attributes used by the properties in the form painter are as follows:

char_col & char_row

These specify the position of an input or label on the character screen. It is these coordinates that link an object on a form to its corresponding input field or label on a character screen. This allows AutoGUI to set focus to a text object or update a label object's string value. These coordinates have no bearing on the position of the object on the GUI form.

char_length

This is the length of the character field or label associated with the object. It is initially derived when you enter a value into the character field, but may be changed.

dynamic

This determines whether the value of an object is redisplayed whenever data is sent from the host. The default for all object classes, except hostlabelclass, is TRUE (the Properties & Events Window only shows this attribute for hostlabelclass). If the labels on the form do not change, it is more efficient (and avoids flickering whenever a clear screen escape sequence is sent from the host) to set this to FALSE for hostlabelclass. This ensures its value will not be updated or cleared by AutoGUI. However, if the form has labels that do change then set the dynamic property for hostlabelclass to TRUE.

hide_until_inp

This only applies to text objects. Setting it to TRUE ensures that the object will only become drawable when focus is passed to it and will disappear when it loses focus. This is not standard Windows GUI behavior but may be useful where the appearance of input fields clutters the form or creates confusion.

true_value & false_value

Both these attributes apply to hosttoggleclass and determine what key value should be sent to the host when their toggle state is TRUE or FALSE. The false_value attribute also applies to hostcomboclass.

transparent

There is an attribute called 'transparent' on HOSTPBCLASS and HOSTLABELCLASS.

If the 'transparent' attribute is set to a particular RGB color, then the value it is set to is used as a transparent mask for a bitmap. e.g. if the form definition has the value set to 192:SVM:192:SVM:192, where SVM is char (252) then any part of a bitmap that has that color set, will display the color of the object underneath it.. If, for example, it is required to have a button with a bitmap on it that picks up the default windows button colors, then the button itself should not have its background set and the 'transparent' attribute should be set to the desired color.

remove_tab

There is an attribute called 'remove_tab' in HOSTEDITCLASS that when set, will not return the tab character to the host when the user tabs off the field.

reset_on_esc

There is an attribute called RESET_ON_ESC that is available in hosttextclass and hosteditclass. If this attribute is set then when the user hits ESC on a field, the original value will be restored without the host having to reset the field.

SPLIT

There is a property called "split" for use with hosteditclass. If this attribute is set then the hostedit-class will insert VM's at the places where the string has wrapped.

The above new attributes are not available in the Form Painter yet and need to be manually added to the form definitions.

Appendix C - Escape Sequences

This appendix describes the extended escape sequences recognized by SBClient when received from the host. You can use these functions from within any user written host program to enhance functionality when they are used with SBClient. We recommend that you use the host library subroutines to perform these tasks where practical.

In all examples, ESC refers to the ASCII character 27, and spaces between characters are shown for clarity only - they should not be entered.

This appendix assumes considerable knowledge of SBDesktop and its GUI functionality. Consult the SBDesktop Reference Manual, the Data/C++ Reference Manual and The Data/C++ Programming Language for more information.

Escape sequences are:

ESC _ 0 *roccommand* ESC \

Remote Object Call command actions to be passed to the RemoteObjectCallClass.

This is uniquely GUI by nature and if GUI children have been minimized then any sequence of this family will restore GUI mode again.

A response is always sent to the host from this sequence.

ESC _ A ESC \

Returns the SBClient session title followed by CHAR(2), the GUI leadin character.

ESC _ B *col* ; *row* ; *length* ; *depth* ; *frame* ; *style* ESC \

Draws a cleared box in the character emulation window where:

col top left column position of window.

row top row of window.

length total length of window (including vertical lines).

depth total depth of window (including horizontal lines).

frame 0 : no frame, 1 : single line, 2 : double line.

style 0 : no style, 1 : shadow, 2 : exploding (not supported), 3 : shadow and exploding.

ESC _ C *capturemode* ; *runcommand* ESC \

Runs a Data/C++ command from the local VOC, then sends a 0:CR back to the host upon completion. If capturemode is 1 rather than 0 the captured result is then sent back to the host.

Warning

Non-event driven aware commands will freeze all SBClient sessions, GUI screens and so on until completion. Do not use for time-consuming operations, or watch-dog timers may time-out and put you 'out-of-order'.

ESC _ D *appname* ESC \

Launches a DOS application or OS command. SBClient is suspended while the application or command is running. This is not suitable for running time consuming commands.

ESC _ D + *appname* ESC \

Launches a Windows application. SBClient is suspended while the application is running - this is not recommended; use & instead.

ESC _ D & *appname* ESC \

Launches a Windows application asynchronously; the application can be closed at anytime. SBClient will continue to run concurrently in the background.

ESC _ D & n *appname* ESC \

Launches a Windows application as above but n sets the display style of the windows application:

0 run program but hide it.

1 run program in normal restore state mode.

2 run minimized.

3 run maximized.

4 run in current state but don't activate.

5 run in current state.

6 run minimized and activate top window in system list.

7 run minimized and don't activate.

8 run in current state and leave currently active window still active.

9 same as 1.

ESC _ D ? *appname* ESC \

Returns to the host either: 1 (if the application is running) or 0 (if it is not running). appname can contain a full or partial path to the executable.

ESC _ D ? ? *appname* ESC \

Returns to the host either: the application module handle (if the application is running) or 0 (if it is not running). appname can contain a full or partial path to the executable.

ESC _ D @ *appname* ESC \

Returns to the host either: the full pathname of the application (if the application is running) or NUL (if it is not running). appname can contain a full or partial full path to the executable.

ESC _ D # *appname* ESC \

Posts a "Close" message to the application if it is found running. If the message is accepted, 0 is returned to the host otherwise 1 is returned. appname can contain a full or partial path to the executable.

ESC _ D F *title* ESC \

Finds a window by given title and returns the window handle to the host if found or 0 if not found. title may optionally be specified in the form of classname=title to be more specific. The classname must be a real windows reference name and is best found by using a third party window's message SPY utility.

If the window is found, it is made active so that macro keys may be sent to it subsequently.

The returned windows handle may be used for the following tests and actions.

ESC _ D I *windowhandle* ESC \

Returns 1 to the host if a window with the reference windowhandle is currently in an iconised state, otherwise 0 is returned. Use ESC_DF to find the windowhandle first.

ESC _ D S *windowhandle* = showstyle ESC \

Returns 1 to the host if the setting of the window showstyle is successful, otherwise 0 is returned. Use ESC_DF to find the windowhandle first. showstyle can be:

0 hide it.

1 show window in normal restore state mode.

2 minimize the window.

3 maximize the window.

4 inactivate the window.

5 activate the window.

6 minimize the window and activate top window in system list.

7 minimize the window and don't activate.

8 restore window state and leave currently active window still active.

9 same as 1.

ESC _ D Z *windowhandle* ESC \

Returns 1 to the host if a window with the reference windowhandle is currently in a zoomed or maximized state, otherwise 0 is returned. Use ESC_DF to find the windowhandle first.

**ESC _ I scale ; *col* ; *row* ; *width* ; *dosfilename* ; *style* ; *title* ; *response* ESC **

Sequence to create or destroy an image, where:

scale

The percentage size change. 100 is no change from the original size (if it is 100, width and depth may be used to change the scaling).

col

The column position of the top left corner of the image.

row

The row position of the top left corner of the image.

width

The width of the image display window. If it is 0 or scale is not 100%, the size is determined by the image's scaled width.

depth

The depth of the image display window. If it is 0 or scale is not 100%, the size is determined by the image's scaled depth.

dosfname

The image's DOS path and filename. The image may be one of the following formats: Windows BMP, OS/2 BMP, PCX or GIF.

style

The image's display window style. SBClient allows for the following style codes, most of which can be combined:

0 child window locked to the default session

1 pop up, moveable window

2 overlapped moveable window

4 bordered window (thin)

8 caption on top of window

16 has Windows SYSMENU, requires caption

32 vertical scroll bar

64 horizontal scroll bar

128 has a minimize box, requires caption

256 has a maximize box, requires caption

512 initially minimized

1024 initially maximized

2048 thick framed window

4096 do not close window on mouse click

8192 fixed image size; scroll bars are used as opposed to dynamic resizable images (by stretching image window size)

For example, a good pop-up style is: pop up (1) + border (4) + system menu (16) + caption (8). The values for these can be combined to give a single style value of 29 (that is, 1+4+16+8).

title

The optional title to display if the style code used produces a caption. If not specified, the dosfname will be used.

response

A string sent when the image is clicked with the mouse, or a key is pressed when the image has focus. Control characters are entered using the \nnn syntax (for example, a carriage return is \013).

ESC _ L 1 ; *application* ; *topic* ESC \

DDE Connect sequence. Returns 0, then the hDDE handle (if successful) or 1 (if unsuccessful), followed by an error number.

ESC _ L 2 ; *hDDE* ESC \

DDE Disconnect sequence. Returns 0 (if successful) or 1 (if unsuccessful), followed by an error number.

ESC _ L 3 ; *hDDE* ; *timeout* ; *record* ESC \

DDE Read sequence. Returns 0, then the result of the read (if successful) or 1 (if unsuccessful), followed by an error number. The timeout value is in seconds.

ESC _ L 4 ; *hDDE* ; *timeout* ; *record* ; *data* ESC \

DDE Write sequence. Returns 0 (if successful) or 1 (if unsuccessful), followed by an error number. The timeout value is in seconds.

ESC _ L 5 ; *hDDE* ; *timeout* ; *data* ESC \

DDE Execute sequence. Returns 0 (if successful) or 1 (if unsuccessful), followed by an error number. The timeout value is in seconds.

ESC _ L 8 ; *hDDE* ESC \

DDE Get Last Error sequence. Returns 0, then the last error number (if the request is valid) or 1 (if it failed), followed by an error number.

ESC _ M *mciseq* ESC \

Runs a multimedia command via a multimedia (mci) interface.

mciseq is a string as per multimedia command interface specifications but with the following option extensions following a token:

C# convert # from columns to x pixels.

D# convert # from character depth to pixels.

HW substitute hCeo window handle.

HD substitute hCeo device context handle.

I {# # # #} invalidate rectangle.

L# convert # from character width to pixels.

R# convert # from rows to y pixels.

W# convert # from character width to pixels.

Returns 0 (if successful) or 1 (if unsuccessful).

The following are examples of mci sequences sent from the host to a program:

```
CRT CHAR(27):"_M":mci-text-string:CHAR(27):"\":
INPUT RTN.FLAG: ; * Did command work? Error?
INPUT VALUE: ; * Contains Error text or status text
```

Example mci text strings are:

```
OPEN \WINDOWS\CHIMES.WAV TYPE WAVEAUDIO ALIAS CHIME
PLAY CHIME WAIT
PLAY CHIME FROM 0
status chime length
status chime position
seek chime to start
play chime
seek chime to 100
play chime
save chime \sbtw\mysnd.wav
info chime file
set time format milliseconds
play chime from 100
set time format samples
play chime from 100
status chime time format
play chime from 0
status chime mode
stop chime
CLOSE CHIME
```

Similar commands are used for CD audio, DAT, digital video, overlay (analog video in a window), scanner, sequencer, VCR, videodisc and animation devices.

For example, to play track 6 of an audio CD in a PC CDROM drive:

```
open cdaudio
set cdaudio time format tmsf
play cdaudio from 6 to 7
close cdaudio
```

ESC _ m *macrocommand* ESC \

Runs a macrocommand sequence from the host. See Macro Syntax for special macro functions that may be executed.

ESC _ P *enviromentvariable* ESC \

Returns the contents of environmentvariable given back to the host. If no environmentvariable is specified, SBOPORT is assumed.

ESC _ p R

The number of rows that the printer can print, based on the selected printer and font defined in the Printer Setup and Printer Options dialogs. This escape sequence must be followed by an input statement. The number of rows is returned.

ESC _ p V ESC \

Prints the GUI window that has focus.

ESC _ R level ESC \

Restores a saved window (saved by ESC_S) from a level (slot) number between 0 and 47.

ESC _ S level ; *col* ; *row* ; *length* ; *depth* ESC \

Saves a window or area or whole screen into a slot level number between 0 and 47.

If col, row, length and depth are 0 then the whole screen is saved.

ESC _ T 1 ; *defnname* ; *dosid* ; *date* ; *time* ESC \

Header to create a button bar definition.

ESC _ T 2 ; *defnname* ESC \

This sequence destroys the named button bar.

ESC _ T 3 ; *defnname* ; *dosid* ; *date* ; *time* ESC \

Header to test existence of a button bar definition. If it exists and is up to date, it is displayed. The resulting status is sent back to host

ESC _ T 4 ; *defnname* ESC \

This sequence resets all buttons to the up state in the named button bar.

ESC _ T 6 ; *defndata* ESC \

Intermediate sequence used to load a button bar definition with parameters.

ESC _ T n ; *defnname* ESC \

Pops a pressed button between n and 100 on the named button bar.

ESC_TP1

Turns Windows print drivers on.

ESC_TP0

Turns Windows print drivers off.

ESC_TS0

Disables the Transfer pulldown menu in SBClient.

ESC_TS1

Enables the Transfer pulldown menu in SBClient.

ESC _ T x y ESC \

Allows you to set the parameters in the Print Options window.

x is the parameter in the Print Options window.

P Use Windows Printer Drivers

Q Use SBClient Print Font

R Strip Box Characters

T Use Condensed Mode

U OEM to ANSI Conversion

y is the parameter setting

1 On

0 Off

ESC _ T A x *file1* ' ' *file2* ESC \

Converts a file from ANSI to OEM, or from OEM to ANSI.

x is the conversion type

1 converts an OEM file (file1) to ANSI (file2)

0 converts an ANSI file (file1) to OEM (file2)

ESC _ T K n ; *defnname* ESC \

Resets the keyboard key mapping to emulation defaults, if n is 0, or the last saved configuration values if n is 1.

ESC _ T S n ; *defnname* ESC \

Toggles host server commands on or off. If n is 1, host server mode is possible, enabling server menu options.

ESC _ T M n ESC \

Turns the character mouse sequences on or off. If n is 1, mouse events in the character emulation window will be sent to the host.

ESC _ W ESC \

Runs the result of the 'who are you' sequence to the host. This is in the form of:

```
SBGUI SBClientTitle-Rel-version/emulation/sessionId/
```

Index

E

F

G

H

I

L

M

O

P

R

S

T

V

W

THE ART OF DATA MANAGEMENT

SBClient Programmer's Guide DSC-5003