



Entire Screen Builder

Version 5.2.1

User Exits

This document applies to Entire Screen Builder Version 5.2.1 and to all subsequent releases.

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

© Copyright Software AG 1999-2003

All rights reserved.

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

Table of Contents

User Exits	1
User Exits	1
General Information	2
General Information	2
Server Side	2
Viewer Side	3
Server Functions	4
Server Functions	4
NSWBefore	4
NSWAfter	5
Nswexp.h File	5
Web Viewer Client Functions (API)	7
Web Viewer Client Functions (API)	7
Screen Interface Methods	8
AboutBox	10
ComposeId	11
Connect	12
Disconnect	13
EnableControl	14
GetActiveField	15
GetFieldContent	16
GetFieldCount	17
GetFieldIDFromIndex	18
GetTitleBarText	19
IsBS2000ToolbarVisible	20
IsConnected	21
IsStatusBarVisible	22
IsTitlebarVisible	23
IsToolbarVisible	24
PrintCurrentScreen	25
PrintDirectCurrentScreen	26
RunJavascript	27
SendKey	28
SendKeyID	29
SendKeyIDWithText	32
SendKeyWithText	33
SetFieldContent	34
SetReceiveTimeout	35
SetUserNameAndPassword	36
Show3270Toolbar	37
ShowBS2000Toolbar	38
ShowControl	39
ShowNextScreens	40
ShowProperties	41
ShowStatusBar	42
ShowTitleBar	43
ShowToolbar	44
Menu Interface Methods	45

GetMenuHandle	46
MenuType	47
ShowMenu	48
Events	49
Sample: User-defined Event Handling	50
Terminal Viewer Client Functions (API)	51
Terminal Viewer Client Functions (API)	51
Methods	52
CloseSession	53
DisconnectCurrentSession	54
GetCursorPos	55
GetScreenSize	56
Initialise	57
OpenSessionByName	58
PerformEditAction	59
PrintCurrentScreen	60
PrintPreviewCurrentScreen	61
PutData	62
PutDataMapped	63
RunProcedure	64
SetCursorPos	65
ShowDialog	66
Events	67
Common Viewer Functions (API)	68
Common Viewer Functions (API)	68
Screen Interface Methods	68
GetScriptArgument	69
ResetScriptArguments	70
SetScriptArgument	71
Overview of Client Control Properties	72
Overview of Client Control Properties	72
AllowPopupMenu	74
AnonymousLogon	75
AutoDisconnect	76
Background	77
Compressed	78
Connection	79
Embedded	80
HttpPort	81
HttpServer	82
PollTimeout	83
Port	84
PortNumber	85
ReceiveTimeout	86
Repository	87
SecureHttp	88
Server	89
ServerName	90
ShowSplash	91
SslConnection	92
Tunneling	93

TunnelingPollTime	94
TunnelingType	95
UnixLogon	96
UsePCLogonName	97
UseHttpTunneling	98
Natural UNIX User Exits	99
Natural UNIX User Exits	99
Using the Shared Library	99
Sample for a Natural UNIX User Exit	99
nsw_CheckUsernameAndPassword	100

User Exits

There are two types of user exits: Entire Screen Builder user exits and Natural UNIX user exits.

With an Entire Screen Builder user exit, you can link your special processing logic to the Entire Screen Builder Server and the Entire Screen Builder viewers. The functions that you provide will then be called by the server and/or viewer when they have been created and defined. The following topics are provided:

- **General Information** General information on the server-side and viewer-side user exits.
- **Server Functions** Detailed descriptions of the server functions `NSWBefore` and `NSWAfter`, and the contents of the header file `Nswexp.h` which is delivered with Entire Screen Builder. The server functions are only available for the GUI viewers.
- **Web Viewer Client Functions (API)** Applies to the COM interface of Entire Screen Builder's Web Viewer and provides detailed descriptions of screen interface methods, menu interface methods and events.
- **Terminal Viewer Client Functions (API)** Applies to the COM interface of Entire Screen Builder's Terminal Viewer and provides detailed descriptions of methods and events.
- **Common Viewer Functions (API)** Applies to the COM interfaces of Entire Screen Builder's Web Viewer and Terminal Viewer and provides detailed descriptions of methods.
- **Overview of Client Control Properties** Detailed descriptions of all configuration data for the viewers.

With a Natural UNIX user exit, you can check the user name and password sent from the Entire Screen Builder viewers by yourself. The following topic is provided:

- **Natural UNIX User Exits** How to use the shared library, a sample for a user exit and the detailed description of the function `nsw_CheckUsernameAndPassword`.

See also: *Scripting, User Exits and APIs* in *Introducing Entire Screen Builder*.

General Information

This chapter applies to the Entire Screen Builder user exits. It provides general information on the following:

- Server Side
 - Viewer Side
-

Server Side

An Entire Screen Builder user exit is a Dynamic Link Library (DLL) with a set of user-written functions that comply with the interface definition described below. You can specify a user exit DLL for global scope, every application scope and every map scope.

When a User Exit rule has been defined for a given scope, the viewer dynamically loads the DLL when this scope is entered and tries to find the user-exit functions `NSWBefore` and `NSWAfter` in the DLL.

For each new screen in this scope, the viewer calls the user-exit DLL two times: the first time before the screen is shown, and the second time after the user has entered data into the screen - just before the data are sent back to the host. The first call allows to influence the way a screen is displayed by the viewer. The second call gives the chance to process and modify the data entered by the user before they are sent to the host.

The DLL is unloaded when a given scope is left.

To create the DLL, use a development tool such as Microsoft Visual Studio.

The user-exit functions should be written in the C or C++ programming languages. They have to be compiled and linked to a DLL. The following user-exit functions should be implemented and exported in the DLL:

Function	Usage
NSWBefore	Called each time when a screen is received from the legacy application. This allows to process the screen received before it is processed and displayed by the viewer.
NSWAfter	Called each time the viewer is about to send data to the legacy application. This allows to process the data typed by the end-user before the viewer sends them to the legacy application.

If one of the above functions is not exported in the DLL, the viewer will detect this and will never try to call it. However, there will be no error message telling about this fact.

See *Server Functions* for further information.

Viewer Side

Entire Screen Builder's Web Viewer and Terminal Viewer are Microsoft ActiveX controls. Thus, COM interfaces can be used to

- access the defined methods and properties, and to
- receive events.

Note:

A COM interface is not available for the Windows Viewer.

The COM interface for the Web Viewer is different from that used for the Terminal Viewer.

See the sections *Web Viewer Client Functions (API)* and *Terminal Viewer Client Functions (API)* for further information.

Server Functions

This chapter applies to the Entire Screen Builder user exits. The server functions are only available for the GUI viewers. The user-exit functions are provided by the user and are called by the viewer.

The following topics are provided:

- NSWBefore
 - NSWAfter
 - Nswexp.h File
-

NSWBefore

This function is called by the viewer before the screen received from the legacy application is processed by the viewer.

All fields received from the legacy application are passed to this function which can process and modify the contents of these fields. From this function, it is also possible to call external applications with the fields received (for example, a print application or Microsoft Office).

This function can decide whether the received screen has to be shown or not by returning `NSWEXP_SHOW (1)` or `NSWEXP_NOSHOW (0)`. If `NSWEXP_NOSHOW` is returned, the viewer sends an `ENTER` to the legacy application and waits to receive the next screen.

Format

```
WORD FAR PASCAL NSWBefore ( WORD fields, SCREENFIELD FAR * field )
```

Input Parameters

<code>WORD fields</code>	Number of received input and output fields.
<code>SCREENFIELD FAR * field</code>	Array of structures containing the fields received from the legacy application.

Return Values

<code>NSWEXP_SHOW</code>	The screen will be shown.
<code>NSWEXP_NOSHOW</code>	The screen will not be shown.

NSWAfter

This function is called by the viewer before the fields are sent to the legacy application.

All fields are passed to this function which can process and modify the data strings typed by the end-user. From this function, it is also possible to call external applications with the fields received (for example, a print application or Microsoft Office).

Format

```
void FAR PASCAL NSWAfter ( WORD fields, SCREENFIELD FAR * field )
```

Input Parameters

WORD fields	Number of received input and output fields.
SCREENFIELD FAR * field	Array of structures containing the input and output fields before sending them to the legacy application.

Return Values

No return values.

Nswexp.h File

The *Nswexp.h* file (see below) is a header file that must be included in the C or C++ source code of the user exit functions on the server side. It contains all defines and function prototypes that can be used in the user exit functions.

This header file is delivered with Entire Screen Builder. You can find it in the program folder *\Entire Screen Builder 5\samples\sampleuserexit*.

Note:

This folder also contains a C source file with a framework for the user-exit functions. You can use this file to write your own program code. The file in this folder with the extension *def* can be used to build the DLL.

```

/*****
** FILE:                NSWEXP.H
**
** DESCRIPTION:        Header file for Entire Screen Builder User-exit DLLs
**
** VERSION:            4.1.1.0
**
** (C) Copyright Software AG, 2000
**
*/

#ifndef _NSWEXP_INCLUDED
#define _NSWEXP_INCLUDED

/*****
** This is the structure that defines screen fields
**
*/
#pragma pack( push ) /* Save current structure alignment (Visual C++ 4.2) */
#pragma pack(4)      /* Switch to 4-byte structure alignment (Visual C++ 4.2) */
typedef struct tagSCREENFIELD
{
    char szText[81]; /* screen field content */
    WORD nLong;      /* screen field length */
    WORD nColumn;    /* screen field column */
    WORD nRow;       /* screen field row */
    WORD nType;      /* screen field type (see below) */
} SCREENFIELD;
#pragma pack(pop) /* Restore previous structure alignment (Visual C++ 4.2) */

/*****
** Values for SCREENFIELD.nType field
**
*/
#define TYPE_INPUT 0 /* Input screen field */
#define TYPE_OUTPUT 1 /* Output screen field */

/*****
** These are the types for the exported user-exit DLL functions. There should
** be two functions
**
** WORD FAR PASCAL NSWBefore( WORD fields, SCREENFIELD FAR * field )
** void FAR PASCAL NSWtAfter( WORD fields, SCREENFIELD FAR * field )
**
*/
#ifdef _cplusplus
    extern "C" {
#endif

/* Pointers to functions types definitions */
typedef WORD (FAR PASCAL *NSWBEFOREPROC)( WORD fields, SCREENFIELD FAR * field );
typedef void (FAR PASCAL *NSWAFTERPROC)( WORD fields, SCREENFIELD FAR * field );

/* Function declarations */
WORD FAR PASCAL NSWBefore( WORD fields, SCREENFIELD FAR * field );
void FAR PASCAL NSWtAfter( WORD fields, SCREENFIELD FAR * field );

#ifdef _cplusplus
    };
#endif

#endif /* #ifndef _NSWEXP_INCLUDED */

```

Web Viewer Client Functions (API)

This section applies to the COM interface of Entire Screen Builder's Web Viewer.

This section assumes that you are familiar with the use of ActiveX controls in applications. You can use ActiveX controls from a wide variety of languages: C++, JavaScript, Visual Basic, Visual Basic for Applications, etc.

The interface consists of a set of properties, methods and events. Properties are needed for the control to know how to make the connection and its visual appearance on screen. The methods are used to handle connection/disconnection and manipulate screen data. The events are used to notify the container about changes in the control's state (e.g. an event is fired when a new screen arrives).

This chapter covers the following topics:

- Screen Interface Methods
- Menu Interface Methods
- Events

See also: *Common Viewer Functions (API)* and *Overview of Client Control Properties*

Screen Interface Methods

The following methods are available:

- AboutBox
- ComposeId
- Connect
- Disconnect
- EnableControl
- GetActiveField
- GetFieldContent
- GetFieldCount
- GetFieldIDFromIndex
- GetTitleBarText
- IsBS2000ToolbarVisible
- IsConnected
- IsStatusbarVisible
- IsTitlebarVisible
- IsToolbarVisible
- PrintCurrentScreen
- PrintDirectCurrentScreen
- RunJavascript
- SendKey
- SendKeyID
- SendKeyIDWithText
- SendKeyWithText
- SetFieldContent
- SetReceiveTimeout

- SetUserNameAndPassword
- Show3270Toolbar
- ShowBS2000Toolbar
- ShowControl
- ShowNextScreens
- ShowProperties
- ShowStatusBar
- ShowTitleBar
- ShowToolbar

AboutBox

This method shows the control's About dialog box.

This dialog box shows the version number of the control and the copyright notice.

Format

```
void AboutBox()
```

Input Parameters

No input parameters.

Return Values

No return values.

ComposeId

To find out the IDs of the automatically generated fields, you can use this method. For example, if you know that there is an input field in the current screen in coordinates "12,7", you can call `ComposeId(12,7)` to get the dialog ID of the edit control associated with this field.

Returns IDs for dialog controls when using basic rules. When basic rules are applied, the screen fields are directly translated to dialog controls: output fields in the screen are mapped to static Windows controls, and input fields in the screen are mapped to edit dialog controls.

When using extended dialogs, there is no need to call this method, as the user already knows the IDs of the controls. These IDs are set by the user from the resource editor used to generate the dialog.

Format

```
long ComposeId(long iRow, long iCol)
```

Input Parameters

<code>long iRow</code>	Row number.
<code>long iCol</code>	Column number.

Return Values

<code>long</code>	The Windows control identification.
-------------------	-------------------------------------

Connect

This method connects the control to the server.

This method gets its parameters from the control properties and connects to the server with the appropriate method. If an error occurs when connecting, the ActiveX shows it to the user.

After this method has been called, two exclusive events can be fired: `Connected` or `ConnectFailed`. The first is fired if the control succeeds in connecting, the latter is fired if the connection fails.

Format

```
void Connect ( )
```

Input Parameters

No input parameters.

Return Values

No return values.

Disconnect

This method forces the control to disconnect.

If an error occurs, the ActiveX notifies the user with a message box. If the control has successfully been disconnected, the `Disconnected` event is fired from the control.

Format

```
void Disconnect ()
```

Input Parameters

No input parameters.

Return Values

No return values.

EnableControl

This method enables or disables a control. It allows individual controls on a dialog to be disabled separately from the repository settings.

Format

```
EnableControl(long ControlID, long Enable)
```

Input Parameters

long ControlID	The control ID is usually returned from GetFieldIDFromIndex.
long Enable	Can either be set to non-zero (enable) or zero (disable).

Return Values

Always true.

GetActiveField

This method returns the control identification with the current focus.

Format

```
long GetActiveField()
```

Input Parameters

No input parameters.

Return Values

long Control identification.

GetFieldContent

This method returns the window text of the control with ID `iFieldId` in the currently shown dialog.

This method can be used for extended and basic dialogs. To find the ID of a control in a basic dialog, use the `ComposeId` method.

For example, to get the contents of the field in row 12 and column 7 in a basic dialog, call the following:

```
control.GetFieldContent( control.ComposeId( 12, 7 ) )
```

For example, to get the name of a button with ID 34572 in an extended dialog, call the following:

```
control.GetFieldContent( 34572 )
```

Format

```
BSTR GetFieldContent(long iFieldId)
```

Input Parameters

`long iFieldId` Control identification in the dialog.

Return Values

`BSTR` A string with the field content.

GetFieldCount

This method returns the number of all fields in the current screen.

You can iterate all controls as follows:

```
int iTotal = m_cNWWClient.GetFieldCount();
for (int iLoop = 0; iLoop < iTotal; iLoop++)
{
    int iTestItem = m_cNWWClient.GetFieldIDFromIndex(iLoop);

    // Now iTestItem can be used in functions needing
    // a control ID
}/* End for : iLoop of available fields */
```

Format

GetFieldCount

Input Parameters

No input parameters.

Return Values

Total of fields.

GetFieldIDFromIndex

This method returns the field ID of the n th field of the screen. See the `GetFieldCount` sample.

Format

```
GetFieldIDFromIndex(long FieldIndex)
```

Input Parameters

<code>long FieldIndex</code>	The index of the field. For example, the 10th field on the screen.
------------------------------	--

Return Values

Field ID or 0 if not found.

GetTitleBarText

This method returns the text that is defined for the title bar.

Format

GetTitleBarText

Input Parameters

No input parameters.

Return Values

String containing the current title.

IsBS2000ToolbarVisible

This method returns true if the BS2000 toolbar is currently visible.

Format

```
BOOL IsBS2000ToolbarVisible()
```

Input Parameters

No input parameters.

Return Values

true	The BS2000 toolbar is visible.
false	The BS2000 toolbar is not visible.

IsConnected

This method returns true if the control is connected.

Format

```
boolean IsConnected ()
```

Input Parameters

No input parameters.

Return Values

true	The viewer is connected to the server.
false	The viewer is not connected to the server.

IsStatusbarVisible

This method returns true if the status bar is currently visible.

Format

```
BOOL IsStatusbarVisible()
```

Input Parameters

No input parameters.

Return Values

true	The status bar is visible.
false	The status bar is not visible.

IsTitlebarVisible

This method returns true if the title bar is currently visible.

Format

```
BOOL IsTitlebarVisible()
```

Input Parameters

No input parameters.

Return Values

true	The title bar is visible.
false	The title bar is not visible.

IsToolbarVisible

This method returns true if the toolbar is currently visible.

Format

```
BOOL IsToolbarVisible()
```

Input Parameters

No input parameters.

Return Values

true	The toolbar is visible.
false	The toolbar is not visible.

PrintCurrentScreen

This method prints the contents of the current screen on a printer that is defined in Windows. The Print dialog box appears in which you can select another printer and/or modify the print properties.

You can either print the GUI screen on which the transformation rules are applied or the character screen (i.e. the actual screen as sent from the host).

Format

```
BOOL PrintCurrentScreen(long IPrintType);
```

Input Parameters

long IPrintType Can either be set to 0 (character screen) or 1 (GUI screen).

Return Values

True if printed.

PrintDirectCurrentScreen

This method immediately prints the contents of the current screen on the default printer that is defined in Windows. The Print dialog box does not appear.

You can either print the GUI screen on which the transformation rules are applied or the character screen (i.e. the actual screen as sent from the host).

Format

```
BOOL PrintDirectCurrentScreen(long IPrintType);
```

Input Parameters

long IPrintType Can either be set to 0 (character screen) or 1 (GUI screen).

Return Values

True if printed.

RunJavascript

This method executes any JavaScript procedure on the Entire Screen Builder Server.

The JavaScript routine must be specified in `pcJavaScriptProcedure`.

Format

```
void RunJavascript(BSTR pcJavaScriptProcedure)
```

Input Parameters

BSTR `pcJavaScriptProcedure` The JavaScript procedure name.

Return Values

No return values.

SendKey

This method simulates a key stroke for the current dialog. You must specify the ID of the control to which the cursor is to be positioned.

The key name must be a string from the following list:

Key Name	Description
ATTN	Send the "Attention" key to the server.
CLEAR	Send the "Clear" key to the server.
PA1 through PA3	Send the specified application key number to the server.
PF1 through PF48	Send the specified function key number to the server.
RESET	Send the "Reset" key to the server.
RETURN	Send the "Enter" key to the server.
SYSREQ	Send the "System Request" key to the server.

To send PF keys in AS/400 style, the session has to be defined as an AS/400 session. See *Communication Properties for Telnet TN3270* in Entire Screen Builder's *System Management Hub* documentation. Entire Screen Builder provides a key scheme with a special layout for AS400 host systems. See *Key Schemes* in Entire Screen Builder's *System Management Hub* documentation.

If the key name is not found in this list, a "RETURN" will be sent.

Format

```
void SendKey(BSTR szKeyName, long iFieldId)
```

Input Parameters

BSTR szKeyName	Key name to be sent to the server.
long iFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyID

This method simulates a key stroke for the current dialog. You must specify the ID of the control to which the cursor is to be positioned.

The key ID must be one from the following list:

Key Name	Key ID
RETURN	0x7D
SYSREQ	0xF0
CLEAR	0x6D
ATTN	0x7E
RESET	0x03
PA1	0x6C
PA2	0x6E
PA3	0x6B
PF1	0xF1
PF2	0xF2
PF3	0xF3
PF4	0xF4
PF5	0xF5
PF6	0xF6
PF7	0xF7
PF8	0xF8
PF9	0xF9
PF10	0x7A
PF11	0x7B
PF12	0x7C
PF13	0xC1
PF14	0xC2
PF15	0xC3
PF16	0xC4
PF17	0xC5
PF18	0xC6
PF19	0xC7
PF20	0xC8

Key Name	Key ID
PF21	0xC9
PF22	0x4A
PF23	0x4B
PF24	0x4C
PF25	0x4D
PF26	0x4E
PF27	0x4F
PF28	0x50
PF29	0x51
PF30	0x52
PF31	0x53
PF32	0x54
PF33	0x55
PF34	0x56
PF35	0x57
PF36	0x58
PF37	0x59
PF38	0x5A
PF39	0x5B
PF40	0x5C
PF41	0x5D
PF42	0x5E
PF43	0x5F
PF44	0x60
PF45	0x61
PF46	0x62
PF47	0x63
PF48	0x64

If the key ID is not found in this list, a "RETURN" will be sent.

Format

```
void SendKeyID(long KeyID, long iFieldId)
```

Input Parameters

long KeyID	Key ID to be sent to the server.
long iFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyIDWithText

This method simulates a key stroke for the current dialog. It also allows you to send text with the key stroke, for example, a user ID or password. The text cannot be seen by the user.

Format

```
void SendKeyIDWithText(BSTR TextToSend, long FieldId, long KeyID, long KeyFieldId)
```

Input Parameters

BSTR TextToSend	Text to be sent to the host.
long FieldId	Field identification of the control to which the text is to written.
long KeyID	Key ID to be sent to the host. See the SendKeyID method for all valid key IDs.
long KeyFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyWithText

This method simulates a key stroke for the current dialog. It also allows you to send text with the key stroke, for example, a user ID or password. The text cannot be seen by the user.

Format

```
SendKeyWithText(BSTR TextToSend, long FieldId, BSTR KeyName, long KeyFieldId)
```

Input Parameters

BSTR TextToSend	Text to be sent to the host.
long FieldId	Field identification of the control to which the text is to be written.
BSTR KeyName	Key name to be sent to the host. See the <code>SendKey</code> method for all valid key names.
long KeyFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SetFieldContent

This method sets the text of a control. It is the complement of GetFieldContent.

The new text of the control must be specified in pcContent.

Format

```
void SetFieldContent(long iFieldId, BSTR pcContent)
```

Input Parameters

long iFieldId	Field identification of the control in which data has to be replaced.
BSTR pcContent	Data.

Return Values

No return values.

SetReceiveTimeout

This method defines how long the ActiveX is to wait for data before a timeout occurs.

This method can be called once the ActiveX is connected. A good place for calling this method is in the `Connected()` event which is fired when the connection with the server has been established.

When the ActiveX has not received any data after the defined number of seconds, the communication with the server is closed and a message box is displayed indicating the error.

Format

```
void SetReceiveTimeout(long ITimeout)
```

Input Parameters

<code>long ITimeout</code>	Number of seconds. When you specify 0, the ActiveX waits infinitely; a timeout does not occur.
----------------------------	--

Return Values

No return values.

SetUserNameAndPassword

This method lets the client connect to the Entire Screen Builder Server using the user name and password passed in the first and second parameter.

This method must be called before the `Connect` method. When the `SetUserNameAndPassword` method is called, the User Authentication dialog box is not shown.

Format

```
void SetUserNameAndPassword(BSTR pcUsername, BSTR pcPassword)
```

Input Parameters

BSTR pcUsername	The user name used to connect to the Entire Screen Builder Server.
BSTR pcPassword	The password used to connect to the Entire Screen Builder Server.

Return Values

No return values.

Show3270Toolbar

This method shows or hides the 3270 toolbar.

Format

```
Show3270Toolbar(long Show)
```

Input Parameters

long Show Can either be set to non-zero (show) or zero (hide).

Return Values

No return values.

ShowBS2000Toolbar

This method shows or hides the BS2000 toolbar.

Format

```
ShowBS2000Toolbar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

ShowControl

This method shows or hides a control.

This is the same as `EnableControl`. It only hides instead of enabling.

Format

```
ShowControl(long ControlID, long Show)
```

Input Parameters

<code>long ControlID</code>	The control ID is usually returned from <code>GetFieldIDFromIndex</code> .
<code>long Show</code>	Can either be set to non-zero (show) or zero (hide).

Return Values

Always true.

ShowNextScreens

This method shows the next screens. If set to true, the next screens will be shown. If set to false, the next screens will not be shown.

Format

```
void ShowNextScreens(bool bShow)
```

Input Parameters

bool bShow True or false.

Return Values

No return values.

ShowProperties

This method shows the properties dialog of the ActiveX control. This dialog can be used to edit the control properties from a graphical user interface.

This method does not return until the user chooses the **OK** or **Cancel** button in the properties' dialog.

Format

```
void ShowProperties()
```

Input Parameters

No input parameters.

Return Values

No return values.

ShowStatusBar

This method shows or hides the status bar.

Format

```
ShowStatusBar(long Show)
```

Input Parameters

long Show Can either be set to non-zero (show) or zero (hide).

Return Values

No return values.

ShowTitleBar

This method shows or hides the title bar.

Format

```
ShowTitleBar(long Show)
```

Input Parameters

long Show Can either be set to non-zero (show) or zero (hide).

Return Values

No return values.

ShowToolbar

This method shows or hides the toolbar.

Format

```
ShowToolbar(long Show)
```

Input Parameters

long Show Can either be set to non-zero (show) or zero (hide).

Return Values

No return values.

Menu Interface Methods

The menu interface methods can be only used when the `AllowPopupMenu` property is set to "true".

The following methods are available:

- `GetMenuHandle`
- `MenuType`
- `ShowMenu`

GetMenuHandle

Internal only.

MenuType

This method returns the type of the popup menu before it is shown.

Menu types:

Value 1: Main popup

Value 2: Edit popup

Value 3: ListView popup

This method can be used together with the method `ShowMenu` to show or hide the popup menu, depending of the menu type.

Format

```
short MenuType()
```

Input Parameters

No input parameters.

Return Values

short The menu type.

ShowMenu

This methods shows or hides the popup menu, depending of the value passed in the first parameter. If the value is 1, the popup menu will be shown. If the value is 0, the popup menu will not be shown.

Format

```
void ShowMenu(BOOL bShowMenu)
```

Input Parameters

BOOL bShowMenu 1 (show) or 0 (do not show).

Return Values

No return values.

Events

Events are used to notify the container of changes in the state of the control. The following events are available:

Event	Description
void Connected()	This event is fired when the control has successfully been connected to the server. It follows a call to <code>Connect</code> . It is not fired if the connection fails.
void Disconnected()	This event is fired when the control has been disconnected from the server. It can be fired due to a call to <code>Disconnect</code> , or because the server disconnected, or because the user chose the command Disconnect from the menu.
void NewScreenShown()	This event is fired when a new screen is received from the host. It is fired when the dialog is about to appear in the user's screen and not after the new data arrives. At the moment this event is fired, the control has a valid dialog to interact with.
void ScreenSent()	This event is fired after a screen has been sent to the host. When this event has been fired, there is no valid dialog available until <code>NewScreenShown</code> is received again.
void ConnectFailed()	This event is fired when <code>Connect</code> has been called but the connection has failed. A connect can fail if, for example, the server is not accepting connections.
void ShowingPopupMenu(MenuInterface)	<p>This event is fired when the right mouse button is pressed. A pointer to the <code>MenuInterface</code> class is passed in the first parameter, which can be used to get the menu type and to show or hide the popup menu. For example:</p> <pre>MenuInterface->ShowMenu(TRUE/FALSE) MenuType = MenuInterface->MenuType)</pre> <p>where <code>MenuType</code> is:</p> <ul style="list-style-type: none"> Main popup Menu 0x01 Edit control Menu 0x02 Listview Menu 0x03

Event	Description
void UserDefinedEvent (szText)	This event is fired when the user chooses a push button or static text control for which a user-defined event has been defined in the SDK. szText contains the text defined in the SDK. For further information, see <i>Defining the Control Properties</i> (the description of the Action page) in the documentation <i>Defining the Rules Using the SDK</i> . When a user-defined event is fired, data is not sent to the server (that is: there is no communication with the server).

Sample: User-defined Event Handling

The user-defined event in this sample is used to check and filter the command that has been entered in the dialog. A user-defined event has been defined for the **OK** button (with text "ok-button") and for a static control (with text "other"). The used field IDs depend on the dialog that has been used and can be seen in the SDK.

```
<script language="VBScript">
<!--
Sub cControl_UserDefinedEvent(szText)
if szText="ok-button" Then
    input = cControl.GetFieldContent(3004)
    i = 0
    if input="avs" Then i = i + 1
    if input="myp" Then i = i + 1
    id = 3005
    if i=0 Then call cControl.SetFieldContent(id, "Please enter a valid command")
    if i=1 Then call cControl.SendKey(Return, 3004)
end If

if szText="other" Then
    msgbox "other was chosen"
end If
end sub
-->
</script>
```

Terminal Viewer Client Functions (API)

This section applies to the COM interface of Entire Screen Builder's Terminal Viewer.

This section assumes that you are familiar with the use of ActiveX controls in applications. You can use ActiveX controls from a wide variety of languages: C++, JavaScript, Visual Basic, Visual Basic for Applications, etc.

The interface consists of a set of properties, methods and events. Properties are needed for the control to know how to make the connection and its visual appearance on screen. The methods are used to handle connection/disconnection and manipulate screen data. The events are used to notify the container about changes in the control's state (e.g. an event is fired when a new screen arrives).

This chapter covers the following topics:

- Methods
- Events

See also: *Common Viewer Functions (API)* and *Overview of Client Control Properties*

Methods

The following methods are available:

- CloseSession
- DisconnectCurrentSession
- GetCursorPos
- GetScreenSize
- Initialise
- OpenSessionByName
- PerformEditAction
- PrintCurrentScreen
- PrintPreviewCurrentScreen
- PutData
- PutDataMapped
- RunProcedure
- SetCursorPos
- ShowDialog

CloseSession

This method closes any open host session.

Format

```
boolean CloseSession();
```

Input Parameters

No input parameters.

Return Values

True if successful, false if not.

DisconnectCurrentSession

This method disconnects the control from the server and also closes any host sessions.

Format

```
boolean DisconnectCurrentSession();
```

Input Parameters

No input parameters.

Return Values

True if successful, false if not.

GetCursorPos

This method gets the current cursor position.

Format

```
boolean GetCursorPos(short* Row, short* Column);
```

Input Parameters

<code>short* Row</code>	The current row position of the cursor.
<code>short* Column</code>	The current column position of the cursor.

Return Values

True if connected, false if not.

GetScreenSize

This method gets the current screen size. It can be called in response to the `ScreenSizeChanged` event.

Format

```
boolean GetScreenSize(short* NumRows, short* NumColumns);
```

Input Parameters

<code>short* NumRows</code>	Current number of rows of the host screen.
<code>short* NumColumns</code>	Current number of columns of the host screen.

Return Values

True if connected, false if not.

Initialise

This method connects the control to the defined server.

Format

```
boolean Initialise();
```

Input Parameters

No input parameters.

Return Values

True if successful, false if not.

OpenSessionByName

This method opens the named host session which must be defined for the server.

Format

```
boolean OpenSessionByName(BSTR SessionName, BSTR SessionIPAddress, long PortNumber);
```

Input Parameters

BSTR SessionName	Defines the host session to be opened.
BSTR SessionIPAddress	The IP address for the host session.
long PortNumber	The port number for the host session.

If `SessionIPAddress` and `PortNumber` are set, they will be used instead of the values defined in the server configuration.

It is not required to set these parameters. If `SessionIPAddress` is "" and `PortNumber` is 0, the defined server values will be used.

Return Values

True if successful, false if not.

PerformEditAction

This method automates the edit actions for the control.

Format

```
boolean PerformEditAction(long lEditID);
```

Input Parameters

long lEditID Specify one of the following numbers:

- 0 = Cut
- 1 = Copy
- 2 = Paste
- 3 = Append Copy
- 4 = Clear
- 5 = Select All

Return Values

True if successful, false if not.

PrintCurrentScreen

This method automates printing the current screen.

Format

```
boolean PrintCurrentScreen(short bDirect);
```

Input Parameters

<code>short bDirect</code>	If true, the print dialog will not be shown. If false, the print dialog will be shown before the screen is printed.
----------------------------	---

Return Values

True if successful, false if not.

PrintPreviewCurrentScreen

This method displays the current screen in print preview mode.

Format

```
boolean PrintPreviewCurrentScreen();
```

Input Parameters

No input parameters.

Return Values

True if successful, false if not.

PutData

This method sends data to the current cursor position. In addition, a function key code can be passed to the server. The keycode will *not* use the mapping defined in the key table for the open session.

The key codes for this method can be found in the *Samples\Definitions* folder of the Entire Screen Builder CD-ROM. The file name is *TerminalViewerKeycodes.h*.

Format

```
boolean PutData(BSTR TextToPut, short KeyCode);
```

Input Parameters

BSTR TextToPut	Text to be copied to the current screen position.
short KeyCode	The function key code to be passed to the server.

Return Values

True if successful, false if not.

PutDataMapped

This method sends data to the current cursor position. In addition, a function key code can be passed to the server. The keycode will use the mapping defined in the key table for the open session.

The key codes for this method can be found in the *Samples\Definitions* folder of the Entire Screen Builder CD-ROM. The file name is *TerminalViewerKeycodes.h*.

Format

```
boolean PutDataMapped(BSTR TextToPut, short MappedKeyCode);
```

Input Parameters

BSTR TextToPut	Text to be copied to the current screen position.
short MappedKeyCode	The function key code to be passed to the server.

Return Values

True if successful, false if not.

RunProcedure

This method attempts to start the named script on the server.

Format

```
boolean RunProcedure(BSTR sProcedureName);
```

Input Parameters

BSTR sProcedureName Must contain the name of the script on the server.

Return Values

True if successful, false if not.

SetCursorPos

This method changes the current cursor position.

Format

```
boolean SetCursorPos(short Row, short Column);
```

Input Parameters

short Row	Row to which the cursor is to be set.
short Column	Column to which the cursor is to be set.

Return Values

True if connected, false if not.

ShowDialog

This method is used to automatically show the control's configuration dialogs.

Format

```
boolean ShowDialog(long lDialogID);
```

Input Parameters

long lDialogID Specify one of the following numbers:

0 = Open session dialog

1 = Task list dialog

2 = Color dialog

3 = Font dialog

Return Values

True if successful, false if not.

Events

Events are used to notify the container of changes in the state of the control. The following events are available:

Event	Description
<code>void Connected()</code>	This event is fired when the control is connected to the server.
<code>void Disconnected()</code>	This event is fired when the control is disconnected from the server.
<code>void ScreenSizeChanged()</code>	This event is fired when the screen size in the open session has changed (e.g. when the connected session changes from mode 3 to mode 4).
<code>void SessionClosed()</code>	This event is fired when an active host session has been closed.
<code>void SessionOpened()</code>	This event is fired when an active host session has been opened.

Common Viewer Functions (API)

This section applies to the COM interfaces of Entire Screen Builder's Web Viewer and Terminal Viewer.

This section assumes that you are familiar with the use of ActiveX controls in applications. You can use ActiveX controls from a wide variety of languages: C++, JavaScript, Visual Basic, Visual Basic for Applications, etc.

This chapter covers the following topics:

- Screen Interface Methods

See also: *Web Viewer Client Functions (API)* and *Terminal Viewer Client Functions (API)*

Screen Interface Methods

The following methods are available:

- GetScriptArgument
- ResetScriptArguments
- SetScriptArgument

GetScriptArgument

This method returns values that are defined in the list of script arguments.

Format

```
BSTR SetScriptArgument(BSTR ArgName);
```

Input Parameters

BSTR ArgName	Name for the argument.
--------------	------------------------

Return Values

BSTR	The value of the defined argument. Blank, if the argument is not found.
------	---

ResetScriptArguments

This method allows the removal of all script arguments in one call.

Format

```
boolean ResetScriptArguments()
```

Input Parameters

No input parameters.

Return Values

True if successful.

SetScriptArgument

This method allows script arguments to be defined that can be passed to and from the server JavaScript engine.

Format

```
boolean SetScriptArgument(BSTR ArgName, BSTR ArgValue);
```

Input Parameters

BSTR ArgName	Name for the argument.
BSTR ArgValue	Value for the argument. If blank, the argument is removed from the list of script arguments.

Return Values

True if set correctly, false if not.

Overview of Client Control Properties

Properties are configuration data for the viewers. They define the behavior and appearance of the viewers and also tell them what to do at startup time.

You must set the properties before the viewer is activated by the `Connect` method for the GUI viewers or the `Initialise` method for the Terminal Viewer. Once this method has been called, changing properties has no effect, i.e. these properties are only read *before* `Connect` or `Initialise` is called and never read again unless the control is disconnected and `Connect` or `Initialise` is called again.

For most properties, there are default values which are used by the viewers if the property is not explicitly set.

The way you set the properties and the exact syntax depends on the viewer, but more on the language you use. The sample HTML files you find after installation in the Entire Screen Builder folders *web viewer* and *terminal viewer* show you how to preset the properties in HTML (`param` tag in the `object` tag; for example, `<param name="BACKGROUND" value="192,192,192">`), and how to set the properties with Visual Basic Script which is embedded in the HTML code (see the `onclick` handling for the **Connect** button).

For the Windows Viewer, the properties can be set when you invoke the Windows Viewer or in the SDK's Client Control Properties dialog box.

The following properties are available:

- AllowPopupMenu
- AnonymousLogon
- AutoDisconnect
- Background
- Compressed
- Connection
- Embedded
- HttpPort
- HttpServer
- PollTimeout
- Port
- PortNumber
- ReceiveTimeout

- Repository
- SecureHttp
- Server
- ServerName
- ShowSplash
- SslConnection
- Tunneling
- TunnelingPollTime
- TunnelingType
- UnixLogon
- UsePCLogonName
- UseHttpTunneling

Note:

The property names are case-insensitive. The upper-lower case spelling for the above property names is only used for better readability.

AllowPopupMenu

Determines whether a context menu is shown when the user clicks the right mouse button (true) in a viewer or not (false).

Type

boolean

Default Value

true

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	No

AnonymousLogon

If set to true, an anonymous logon is made to the Entire Screen Builder Server.

If set to false, the User Authentication dialog box appears and the user has to specify user name and password as defined with the System Management Hub. The user is then logged on to his user profile in the Entire Screen Builder Server. See *Users* in Entire Screen Builder's *System Management Hub* documentation for further information.

Type

boolean

Default Value

true

Usage

Terminal Viewer: Yes

Web Viewer: Yes

Windows Viewer: Yes

AutoDisconnect

If set to true, the session is disconnected automatically when a multi screen rule fails (for example, when the extended dialog cannot be loaded or when the detection of a green screen fails).

If set to false, the session is not disconnected when a multi screen rule fails. In this case, the screen will be displayed with the rules defined for this screen (normal rules processing).

Type

boolean

Default Value

false

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

Background

The color for the area outside the defined dialog.

Type

COLORREF

Default Value

192,192,192

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

Compressed

Enable or disable the compression mechanism between the Entire Screen Builder Server and the viewer.

Type

boolean

Default Value

true

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

Connection

Corresponds to the **Session ID** property defined with the System Management Hub. See *Host Sessions* in Entire Screen Builder's *System Management Hub* documentation for further information.

The connection number 0 has a special meaning. See *Startup Scripts* in the *Script Files* documentation.

Type

long

Default Value

1

Usage

Terminal Viewer:	Yes
Web Viewer:	Yes
Windows Viewer:	Yes

Embedded

If set to true, a screen or dialog is shown inside the HTML page (embedded). All controls are then created inside the HTML page.

If set to false, an additional window is created outside the HTML page. You can then, for example, move or minimize this window independently of the browser window.

Type

boolean

Default Value

true for the GUI Viewers

false for the Terminal Viewer

Usage

Terminal Viewer:	Yes
Web Viewer:	Yes
Windows Viewer:	No

HttpPort

This property applies to the GUI viewers. Its behavior depends on the setting of the `Tunneling` property (i.e. whether tunneling is enabled or disabled).

If tunneling is disabled, this is the number of the port where the internal image server listens.

If tunneling is enabled, this is the number of the port where the HTTP server used for tunneling listens.

Type

short

Default Value

30000

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

HttpServer

This property applies to the GUI viewers. Its behavior depends on the setting of the Tunneling property (i.e. whether tunneling is enabled or disabled).

If tunneling is disabled, this is the IP address or host name of the machine on which the Entire Screen Builder Server is running.

If tunneling is enabled, this is the IP address or host name of the machine on which the HTTP server used for tunneling is running.

Type

BSTR

Default Value

localhost

Enter the name of the PC, if you encounter problems with images that are not displayed.

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

PollTimeout

Applies only when `UseHttpTunneling` has been enabled for the Terminal Viewer.

The time in seconds the terminal viewer polls the tunneling server for new data (asynchronous messages and screens).

Note:

The GUI viewers use the `TunnelingPollTime` property.

Type

long

Default Value

20 seconds

Usage

Terminal Viewer:	Yes
Web Viewer:	No
Windows Viewer:	No

Port

The number of the port where the Entire Screen Builder Server listens for the GUI viewers.

This property is not used when tunneling has been enabled with `Tunneling`. See also `HttpPort`.

Note:

The Terminal Viewer uses the `PortNumber` property.

Type

short

Default Value

22367

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

PortNumber

This property applies to the Terminal Viewer. Its behavior depends on the setting of the `UseHttpTunneling` property (i.e. whether tunneling is enabled or disabled).

If tunneling is disabled, this is the number of the port where the Entire Screen Builder Server listens for the Terminal Viewers.

If tunneling is enabled, this is the number of the port where the HTTP server that is used for tunneling listens.

Note:

The GUI viewers use the `Port` property.

Type

short

Default Value

22340

Usage

Terminal Viewer:	Yes
Web Viewer:	No
Windows Viewer:	No

ReceiveTimeout

This timeout is used to stop the viewer hanging in the highly unlikely event of network disruption causing a TCP packet to be dropped during transmission. The viewer will wait the defined number of seconds. If there is no response from the server, the connection will be terminated.

If this timeout is set to zero, it will not be used.

It is recommended that this timeout be set only in the case where you experience this specific problem.

Type

long

Default Value

0 seconds

Usage

Terminal Viewer:	Yes
Web Viewer:	Yes
Windows Viewer:	No

Repository

An alias name defined in the system in which the internal image server or the Web server is running. This alias points to a folder which contains the images that are used in your applications.

For the internal image server, this alias is defined with the System Management Hub. See *Alias List* in Entire Screen Builder's *System Management Hub* documentation.

Type

BSTR

Default Value

ESB_Repository

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

SecureHttp

Applies only when Tunneling has been enabled for the GUI viewers. Can only be used for an external Web server.

If set to true, the SSL (Secure Socket Layer) protocol is enabled for all HTTP communication, thus providing secure communication. HTTP is used for downloading images.

You must also enable SSL on your HTTP server.

Note:

The Terminal Viewer uses the `SslConnection` property.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

Server

This property applies to the GUI viewers. It defines the IP address or host name of the machine on which the Entire Screen Builder Server is running.

If the Entire Screen Builder Server and the SDK are on the same machine, you may use "localhost" for the local machine.

This property is not used when tunneling has been enabled with `Tunneling`. See also `HttpServer`.

Note:

The Terminal Viewer uses the `ServerName` property.

Type

BSTR

Default Value

localhost

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

ServerName

This property applies to the Terminal Viewer. Its behavior depends on the setting of the `UseHttpTunneling` property (i.e. whether tunneling is enabled or disabled).

If tunneling is disabled, this is the IP address or host name of the machine on which the Entire Screen Builder Server is running.

If tunneling is enabled, this is the IP address or host name of the machine on which the HTTP server used for tunneling is running.

If the Entire Screen Builder Server and the SDK are on the same machine, you may use "localhost" for the local machine.

Note:

The GUI viewers use the `Server` property.

Type

BSTR

Default Value

localhost

Usage

Terminal Viewer:	Yes
Web Viewer:	No
Windows Viewer:	No

ShowSplash

If set to true, the splash screen for the Entire Screen Builder viewer is shown when not connected to the Entire Screen Builder Server.

If set to false, the client screen is blank.

Type

boolean

Default Value

true

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	No

SslConnection

Applies only when `UseHttpTunneling` has been enabled for the Terminal Viewer. Can only be used for an external Web server.

If set to true, the SSL (Secure Socket Layer) protocol is enabled for all HTTP communication, thus providing secure communication. HTTP is used for downloading images.

You must also enable SSL on your HTTP server.

Note:

The GUI viewers use the `SecureHttp` property.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	Yes
Web Viewer:	No
Windows Viewer:	No

Tunneling

This property applies to the GUI viewers.

Enable or disable the tunneling mechanism for sending and receiving the packets through the Web server (to traverse firewalls and/or proxies).

This property should only be set to true when connected from the Internet and when the Entire Screen Builder extension modules have been installed on the machine on which the Web server is running. See *HTTP Tunneling* in the *Installation and Configuration* documentation for further information.

Note:

The Terminal Viewer uses the `UseHttpTunneling` property.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

TunnelingPollTime

Applies only when Tunneling has been enabled for the GUI viewers.

The time in seconds the GUI viewers poll the tunneling server for new data (asynchronous messages and screens).

Note:

The Terminal Viewer uses the PollTimeout property.

Type

long

Default Value

20 seconds

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

TunnelingType

This property is used when tunneling has been enabled with `Tunneling` for the GUI Viewers, or with `UseHttpTunneling` for the Terminal Viewer.

In this case, you must also define the type of external Web server that is installed in your environment (including the production environment).

Use either "M" (Microsoft Internet Information Server) or "A" (Apache Web Server).

Type

BSTR

Default Value

M

Usage

Terminal Viewer:	Yes
Web Viewer:	Yes
Windows Viewer:	Yes

UnixLogon

Used to connect to a Natural UNIX or Natural OpenVMS application.

For Natural UNIX, this only applies when security has been enabled in *\$NATDIR/\$NSWNODE/services.dat* on the Natural UNIX host; see *Installing Entire Screen Builder on Natural UNIX Hosts* in the *Installation and Configuration* documentation.

Under OpenVMS, user name and password are always checked; if this property is set to false, login will fail.

If set to true, a dialog box is shown to get the logon information for a UNIX or OpenVMS system. See also: *Using the GUI Viewers with Natural on UNIX and OpenVMS Hosts* in the *Installation and Configuration* documentation.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	No
Web Viewer:	Yes
Windows Viewer:	Yes

UsePCLogonName

If set to true, the client attempts to connect to the Entire Screen Builder Server using the name of the PC user that is currently logged on.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	Yes
Web Viewer:	Yes
Windows Viewer:	Yes

UseHttpTunneling

This property applies to the Terminal Viewer.

Enable or disable the tunneling mechanism for sending and receiving the packets through the Web server (to traverse firewalls and/or proxies).

This property should only be set to true when connected from the Internet and when the Entire Screen Builder extension modules have been installed on the machine on which the Web server is running. See *HTTP Tunneling* in the *Installation and Configuration* documentation for further information.

Note:

The GUI viewers use the Tunneling property.

Type

boolean

Default Value

false

Usage

Terminal Viewer:	Yes
Web Viewer:	No
Windows Viewer:	No

Natural UNIX User Exits

The Natural UNIX user exits are external libraries built on the UNIX machine to be called by the different Entire Screen Builder components running on the UNIX machine. They are only called by the Entire Screen Builder components in Natural UNIX, this means: the Natural UNIX user exits are never called by the Entire Screen Builder Server.

The Natural UNIX user exits are available for all types of viewers (GUI viewers and Terminal Viewer).

This chapter assumes that you are familiar with programming in C, makefiles and the shared library concept of UNIX.

The following topics are covered below:

- Using the Shared Library
 - Sample for a Natural UNIX User Exit
 - `nsw_CheckUsernameAndPassword`
-

Using the Shared Library

A shared library is a set of user-written functions that can be loaded dynamically by any program that needs to execute any function defined in the library. It is the same as a DLL in Windows.

To use the shared library, you must set the environment variable `NSWUSEREXIT1` before starting the `nswsrvd` daemon. The variable must point to the library as follows:

```
NSWUSEREXIT1=$NSWDIR/samples/userexit/libnswuserexit1.sl ; export NSWUSEREXIT1.
```

The extension depends on the UNIX operating system. For example, `sl` applies to HP and `so` applies to Solaris.

Sample for a Natural UNIX User Exit

A sample for the Natural UNIX user exit can be found in the directory `$NSWDIR/samples/userexit`. There are two files in this directory:

- *Makefile*

Create the shared library using the following UNIX command:

```
make lib
```

- *nswuserexit1.c*

An example of the user exit.

nsw_CheckUsernameAndPassword

This function checks whether user name and password that have been sent from the PC are correct.

Important:

The user exit is responsible for security. The *nswusr* program does not perform any security checks.

When the security option C is used in the *services.dat* file, the function `nsw_CheckUsernameAndPassword` is called from the *nswusr* program. The user name and password sent from the PC are then passed to this function which checks these values.

`pUserMessage` can be used to display a user message in the viewer instead of the standard message. If `pUserMessage` is empty, the standard message is displayed. If `pUserMessage` is not empty, its content (i.e. the user message) is displayed.

When this function returns 0, the *nswusr* program sends either the standard message ("Invalid user name or password") or the user message (for example, "Invalid credentials") to the PC and then waits for a new user name and password.

When this function returns 1, the *nswusr* program starts the shell script defined in the *services.dat* file to run the Natural application. A standard message is not displayed. However, if a user message has been defined (for example, "Congratulations"), it is displayed.

Syntax

```
int nsw_CheckUsernameAndPassword
    (const char *username, const char *password, char *pUserMessage)
```

Input Parameters

<code>const char *username</code>	The user name sent from the PC.
<code>const char *password</code>	The password sent from the PC.
<code>char *pUserMessage</code>	User message to be displayed in the viewer. Maximum size: 512 bytes.

Return Values

- 0 Standard message "Invalid user name or password" or user message.
- 1 Valid user name and password.