OSITRON CTI COM Interface

Technical Note No. 01/2015

Release 4.6

February 2015
Revision 01/2015

OSITRON Kommunikationstechnik GmbH

Directory

1. Introduction 1

1.1. Introduction 1

2. System Configuration 1

2.1. What you need to run the COM Server 1
2.2. Development Environment 1
2.3. Installation of the COM Server 1

3. Features of the COM Interface 1

3.1. Features 1

4. Making a Call with assisted telephony 1

4.1. Making a Call 1

5. Programming the new COM-Interfaces 1

5.1. The CTI Application Object 1
5.1.1. IOSIICTIApplication Properties 1
5.1.2. IOSIICTIApplication Methods 1
5.1.3. _IOSIICTIAppEvents Events 1
5.2. The CTI Call Object 1
5.2.1. IOSICTICall Properties 1
5.2.2. IOSICTICall Methods 1
5.2.3. _IOSICTICallEvents Events 1
5.3. The Static Collection Object 1
5.3.1. IOSIStaticCollection Properties 1
5.3.2. IOSIStaticCollection Methods 1
5.4. The ACD Application Object 1
5.4.1. ACDApplication Properties 1
5.4.2. ACDApplication Methods 1

6. Programming the COM Interface ITelApp 1

6.1. Overview 1
6.2. Application Startup 1
6.3. Starting / Stopping the COM Server 1
6.3.1. Init 1
6.3.2. Exit 1
6.3.3. ServerState 1
6.4. Application Control 1
6.4.1. About 1
6.4.2. HideDown 1
6.4.3. ShowUp 1
6.4.4. ToggleMinMode 1
6.5. Handling of Calls 1
6.5.1. Drop 1
6.5.2. Info 1

6.5.3. State 1
6.6. Signaling of Call Information 1
6.6.1. CallEvent 1
6.7. Definitions 1
6.8. States of the Call 1
6.9. States of the Server 1

7. Samples in Visual Basic 1

7.1. Sample using the new Interfaces 1
7.1.1. Using the new COM Server 1
7.1.2. Using Notification 1
7.1.3. Retrieving the Call List 1
7.1.4. Retrieving Information of a Call 1
7.1.5. Using Call Information Notification 1
7.1.6. Making a Call 1
7.1.7. Drop a Call 1
7.1.8. Code Listing for Visual Basic 1
7.2. Sample using the old Interface 1
7.2.1. Using the COM Server 1
7.2.2. Using Notification 1
7.2.3. Starting / Stopping 1
7.2.4. Retrieving the Server State 1
7.2.5. Application Control 1
7.2.6. Retrieving information of the Call 1
7.2.7. Making a Call 1
7.2.8. Drop a Call 1
7.2.9. Code Listing for Visual Basic 1

8. Sample in Visual C++ 1

8.1. Using the COM Server 1
8.2. Using the Features of the COM Interface 1
8.3. Using Notification of COM Server 1
8.4. Using Call Information Notification 1
8.5. Retrieving Information of a Call 1
8.6. Making a Call 1
8.7. Drop a Call 1
8.8. Code Listing for Visual C++ 1

9. ActiveX Sample in Visual Studio C++ 1

9.1. Using the ActiveX Control 1

10. Sample in Delphi 1

10.1. Using the new COM Server 1
10.2. Using Notification 1
10.3. Retrieving the Call List 1
10.4. Retrieving Information of a Call 1
10.5. Using Call Information Notification 1
10.6. Making a Call 1
10.7. Drop a Call 1
10.8. Code Listing for Delphi 1

11. Sample in Visual Basic .Net 1

11.1. Differences between VB 6.0 and VB.Net 1
11.2. Using the COM Server 1
11.3. Using Notification 1
11.4. Retrieving the Call List 1

11.5. Retrieving Information of a Call 1
11.6. Using Call Information Notification 1
11.7. Making a Call 1
11.8. Drop a Call 1
11.9. Code Listing for Visual Basic .Net 1

12. Status of the document 1

1. Introduction

1.1. Introduction

OSITRON CTI makes it possible to get all the information of an incoming and outgoing phone call and other call information like addresses, states etc. if you are working in a Windows environment with TAPI and OSITRON CTI installed.

To be able to use this information in any other application, it is possible to use the OSITRON CTI COM-Interface directly. In combination with the Assisted Telephony, which is supported by OSITRON CTI too, you are now able not only to request an outgoing call, but also to retrieve address information of incoming calls to handle your own features of address identification and CTI functions.

The following features are supported:

- complete monitoring of incoming and outgoing calls
- simple to use on windows development environments
- with Visual Basic sample application

A new interface was designed in order to handle several calls at the same time and to get more information on the call. You are now able to send SMS (short messages service) if an ISDN adapter is installed.

2. System Configuration

2.1. What you need to run the COM Server

A Windows operating System: Windows2000®, WindowsXP®, Windows NT® 2003.

An installed version of:

- OSITRON CTI – Single 3.5 Revision 01/2005 or
- OSITRON CTI – Server 3.5 Revision 01/2005

2.2. Development Environment

Any Development Tool with COM support like VC++, VB, MS Office, Delphi. The OSITRON CTI COM Server provides a Dual Interface to enable using the standard IDispatch interface when making automation calls. This technique is also referred to as Early Binding because type checking is performed at compile time.

2.3. Installation of the COM Server

If you install the latest version of OSITRON CTI Single or Server, the registration will be done automatically.

3. Features of the COM Interface

3.1. Features

OSITRON CTI 3.5 now comes with new interfaces. The old interface *ITelApp* is still available, but developers are encouraged to use the new interface because new features will only be supported by the new interface.

Common Features:

- Handling incoming calls
- Handling outgoing calls
- Handling call states
- Handling call information

Features of the new interface:

- Multiple calls
- Extended information on calls
- Sending of SMS
- Interactive configuration

Note: The call features are available in dependency of the functions provided by the TAPI Driver. Sending of SMS is only available if you have installed an ISDN adapter.

Features of the old interface:

- Application control
- Starting / stopping the Server
- Getting information about the server states

Note: The features provided by the old interface are not necessary when you use the new interface.

4. Making a Call with assisted telephony

4.1. Making a Call

To make a telephone call anywhere in your program, it isn't necessary to create a new object of the COM Server with all his functionality. You only need to include the header file <tapi.h> or to bind the file "tapi.dll" / "tapi32.dll".

Then call the "tapiRequestMakeCall" function to start a call. All calling information are available at the COM Interface too.

Please refer to the Microsoft documentation for assisted telephony.

5. Programming the new COM-Interfaces

5.1. The CTI Application Object

The CTI application object provides the IOSICTIApplication interface with properties and methods. Events generated by the application object are reported via the _IOSICTIAppEvents dispinterface.

The IOSICTIApplication interface consists of the following method groups:

- Show different configuration dialogs
- Get the call list
- Create a new call
- Pickup a call
- Send a SMS

First, to access the OSITRON CTI methods and properties, you must create a CTI application object, e.g. in Visual Basic:

```
' Declare a variable for the OSITRON CTI object
Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp

Private Sub Form_Load()
    ' Create the OSITRON CTI object
    ' when the form is loaded
    Set OSICTIClient = New OSICTIApp
  ...
```

If You use an ACD license, you have to create an ACD application object too, e.g.

```
Dim OSIACDApp As OSICTIControlCenter.OSIACDApp

    Set OSIACDApp = New OSICTIApp
```

After creating this object you can use the methods supplied by the CTI object and receive events signalled by the CTI COM Server. In the following sub chapters the methods, properties and events are described in alphabetical order. The access to a specific call is described in the chapter 5.2.

The prototypes of properties, methods and events are described in IDL notation. The error code returned by the methods or properties are only described for those the CTI COM server generates. Errors occurring while accessing Windows API function are returned too, but are not described here.

5.1.1. IOSICTIApplication Properties

CallList

This function returns a copy of the server side list of the current calls handled by the CTI COM server.

```
[propget] HRESULT CallList (
    [out, retval] VARIANT *pVal
);
```

Parameters

pVal

Pointer to a variant variable containing a collection of call objects after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The interface contained in the pVal-Value is a IOSIStaticCollection interface. It is described later. You can query the IOSICTICall interface from the returned pointer in the result variant of the item method

If you want to track all calls currently handled by the CTI COM server, the best way is to call this method directly after the creation of the CTI application object. You can keep the resulting interface or you can use your own collection of calls, because new calls arriving after the creation of the CTI application object are signalled by the NewCallEvent event, which can be used to update your own call collection or call the Refresh method of the IOSIStaticCollection interface to refresh the collection.

See also description of RemoveCallEvent event.

5.1.2. IOSIICTIApplication Methods

About

Shows a dialog box with version information.

```
HRESULT About (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open

ConfigureAddressBooks

This method opens a dialog for setting up the address book databases and CTI journal database.

```
HRESULT ConfigureAddressBooks (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

ConfigureCallDeflectManager

This method opens a dialog for setting up deflection of calls.

```
HRESULT ConfigureCallDeflectManager (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

ConfigureCallForwardManager

This method opens a dialog for setting up call forwarding.

```
HRESULT ConfigureCallForwardManager (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_ACCESSDENIED	(CTI LAN version) if you are not allowed to configure call forwarding
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

ConfigureHotKeys

This method opens a dialog to configure the system wide active hotkeys activating special functions of OSTRON CTI.

```
HRESULT ConfigureHotKeys (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

CreateCall

To make a call this method creates a call object you can use to dial. (For use of a call object see chapter 0)

```
HRESULT CreateCall (
    [out, retval] IOSICTICall **ppiCall
);
```

Parameters

ppiCall

Pointer to pointer where the call object is stored.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

This call object is used to dial, to drop the call and to get information about the call. The creation of a call is signalled by the event NewCallEvent, even if you created the call yourself.

MakeConfiguration

This method opens a dialog to configure the OSITRON CTI.

```
HRESULT MakeConfiguration (
    [out, retval] long *plVal
);
```

Parameters

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

PopupSMSDialog

This method opens a dialog to send a SMS.

```
HRESULT PopupSMSDialog (
    [in] BSTR bstrAddressOriginator,
    [in] BSTR bstrAddressReceiver,
    [in] BSTR bstrReceiverName,
    [in] BSTR bstrReceiverInfo,
    [in] BSTR bstrMessage,
    [in] BSTR bstrErrMessage,
    [out, retval] long *plVal
);
```

Parameters

bstrAddressOriginator

Mobile phone number of user. If this parameter is empty the mobile phone number configured in OSITRON CTI will be used.

bstrAddressReceiver

Mobile phone number of the SMS receiver

bstrReceiverName

Name of the receiver (see Remarks)

bstrReceiverInfo

Info about the receiver (see Remarks)

bstrMessage

SMS message

bstrErrMessage

Error message shown on startup of dialog

plVal

Pointer to a variable that will receive the return code of the dialog. Possible values are the standard return values of dialogs:

Name	Value (decimal)
IDOK	1
IDCANCEL	2

Other values, like IDRETRY, may be returned in future versions.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the

	CTI COM server dialogs is still open
E_ACCESSDENIED	(CTI LAN version) if you are not allowed to send SMS
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

All parameters may contain an empty string or NULL. The parameters *bstrReceiverName* and *bstrReceiverInfo* are copied into the CTI journal fields “Name” and “Additional Info” if a journal entry is written.

The client remains in this method until the user closes the dialog window.

SelectLanguage

This method opens a dialog for selecting the language the user wants to see OSITRON CTI.

```
HRESULT SelectLanguage ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

SelectLines

This method opens a dialog for selecting the telephone line the user wants to use.

```
HRESULT SelectLines ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

HRESULT_FROM_WIN32 (ERROR_BUSY)	if one of the CTI COM server dialogs is still open
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The client remains in this method until the user closes the dialog window.

5.1.3. _IOSIICTIAppEvents Events

ACDAppStateChanged

The CTI COM Server signals this event if the ACD State has been changed. This could happen, when the user changes the state active by ACD controls activate and stop or the server controls the state, e.g. to prepare the next call.

```
HRESULT ACDAppStateChanged ();
```

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

This event can be used for monitoring the activity of an agent.

NewCallEvent

After creating the CTI application object (in Visual Basic the variable must be declared with the keyword “WithEvents”) this event is signalled for every new call on the line which is handled by the CTI COM server. There are three reason for this events:

- You or another application attached to the CTI COM server call the method CreateCall.
- An application uses the Telephone API to make a call on the line monitored by the CTI COM server.
- A call is incoming on the line monitored by the CTI COM server.

Prototype of the function called by the server in the context of the CTI application object on the client side:

```
HRESULT NewCallEvent (
    [in] IOSICTIApp *piApp,
    [in] IOSICTICall *piCall
);
```

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Parameters

piApp

Pointer to the CTI application object (not necessary)

piCall

Pointer to the new call object, which you can add to your call list.

Remarks

This event can be used to centrally manage the call objects in conjunction with the event RemoveCallEvent.

When this event is signaled some or all information regarding the specified call may not be available yet. If you need further information about the call you must utilize the _IOSICTICallEvents interface for the provided call object. You should also be aware that certain information like called or calling addresses can dynamically change during the lifetime of a call. Please see the provided samples for further reference.

RemoveCallEvent

The CTI COM server signals this event for every call changed to the call state “Idle”.

Prototype of the function called by the server in the context of the CTI application object on the client side:

```
HRESULT RemoveCallEvent (
    [in] IOSICTIApp *piApp,
    [in] IOSICTICall *piCall
);
```

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Parameters

piApp

Pointer to the CTI application object (not necessary)

piCall

Pointer to the removed call object, which you can removed from your call list.

Remarks

For every call, even calls created before you create the CTI application object, the CTI COM server signals this event. So the call object passed here may be not in your call list (see CallList Method).

5.2. The CTI Call Object

The CTI call object provides the IOSICTICall interface with properties and methods. Events generated by the application object are reported via the _IOSICTICallEvents dispinterface. The prototypes of properties, methods and events are described in IDL notation.

The CTI COM server signals on a call object serveral events indicating that an specific address is valid and/or has changed. The best way to read these address is to implement the event methods in your call object.

5.2.1. IOSICTICall Properties

CalledAddress

The CalledAddress is the address of the called party.

```
[propget] HRESULT CalledAddress (
    [out, retval] BSTR *pbstrCalledAddress
);
```

Parameters

pbstrCalledAddress

Pointer to a bstr variable containing the called address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

For outgoing calls this address is the dialed address. For incoming calls it is the own address (MSN)

CallingAddress

The CallingAddress is the address of the calling party.

```
[propget] HRESULT CallingAddress (
    [out, retval] BSTR *pbstrCallingAddress
);
```

Parameters

pbstrCallingAddress

Pointer to a bstr variable containing the calling address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

For outgoing calls this address is your own address (MSN). For incoming calls it is the address of the remote party.

Charging

The Charging is returned in currency values depending on your telephone system.

```
[propget] HRESULT Charging (
    [out, retval] CURRENCY *pcyCharging
);
```

Parameters

pcyCharging

Pointer to a currency variable containing the currency after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

Changing of charging is siganalled by the ChargingEvent event.

ConnectedAddress

The ConnectedAddress is the address of the remote party.

```
[propget] HRESULT ConnectedAddress (
    [out, retval] BSTR *pbstrConnectedAddress
);
```

Parameters

pbstrConnectedAddress

Pointer to a bstr variable containing the connected address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The ConnetedAddress is available in the call state OSICTICallConnected (see also ConnectedAddressEvent event).

DateTime

The property returns the detecting time of the call.

```
[propget] HRESULT DateTime (
    [out, retval] DATE *pdateDateTime
);
```

Parameters

pdateDateTime

Pointer to a DATE variable containing the detecting time after the methods exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The property returns the time the call was detected on the telephone line. For an outgoing call this is the time you pick up the handset or an application makes a call on the line. For an incoming call this is the time the call is offered to the line.

DialAddress

The DialAddress must be set before calling the Dial method.

```
[propget] HRESULT DialAddress (
    [out, retval] BSTR *pbstrDialAddress
);

[propput] HRESULT DialAddress (
    [in] BSTR bstrDialAddress
);
```

Parameters

pbstrDialAddress

Pointer to a bstr variable containing the dial address after the method exits.

bstrDialAddress

A bstr variable containing the dial address to set.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

If the keypad window opens on call of the Dial method, this address is visible in the phone number edit field. This address is changed if the user enters a phone number (see also DialAddressEvent event).

Direction

The property returns the direction of the call: Outgoing or Incoming.

```
[propget] HRESULT Direction (
    [out, retval] EOSICTICallDirection *peDirection
);
```

Parameters

peDirection

Pointer to a enum EOSICTICallDirection variable containing the direction after the methods exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

Possible values of the type EOSICTICallDirection are listed below:

Name	Value (decimal)
OSICTICallDirectionIncoming	1
OSICTICallDirectionOutcoming	2

It is recommended to use the definitions.

Display

The Display property gets a name of the call for displaying purpose.

```
[propget] HRESULT Display (
    [out, retval] BSTR *pbstrDisplay
);
```

Parameters

pbstrDisplay

Pointer to a bstr variable containing the display name after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The value returned may be empty because the underlaying TAPI provider does not support this feature.

Name

The Name is the external ID of the selected contact entry.

```
[propget] HRESULT Name (
    [out, retval] BSTR *pbstrName
);
```

Parameters

pbstrName

Pointer to a bstr variable containing the external ID after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

NormalizedAddress

The NormalizedAddress is the address of the remote party in a special format.

```
[propget] HRESULT NormalizedAddress (
    [out, retval] BSTR *pbstrNormalizedAddress
);
```

Parameters

pbstrNormalizedAddress

Pointer to a bstr variable containing the normalized address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The format is e.g. for a german phone number, 49241946980. It consists of the fields: country code, area code and local address.

RedirectingAddress

The RedirectingAddress is the address of the party which active redirects a call.

```
[propget] HRESULT RedirectingAddress (
    [out, retval] BSTR *pbstrRedirectingAddress
);
```

Parameters

pbstrRedirectingAddress

Pointer to a bstr variable containing the redirecting address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The RedirectingAddressEvent event informs you when the redirecting address is valid.

RedirectionAddress

The RedirectionAddress is the address of the party the call is redirected to.

```
[propget] HRESULT RedirectionAddress (
    [out, retval] BSTR *pbstrRedirectionAddress
);
```

Parameters

pbstrRedirectionAddress

Pointer to a bstr variable containing the redirection address after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

The RedirectionAddressEvent event informs you when the redirection address is valid.

State

The property returns the state of the call: The state transitions are described in chapter 6.8.

```
[propget] HRESULT State (
    [out, retval] EOSICTICallDirection *peState
);
```

Parameters

peState

Pointer to a EOSICTICallDirection enum variable containing the state after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

Possible values of the type EOSICTICallState are listed below:

Name	Value (decimal)
OSICTICallStateIdle	0
OSICTICallStateRinging	1
OSICTICallStateDialing	2
OSICTICallStateProceeding	3
OSICTICallStateRingback	4
OSICTICallStateConnected	5
OSICTICallStateDisconnected	6
OSICTICallStateHold	7

The values are the same as used by the old COMinterface. It is recommended to use the definitions.

5.2.2. IOSICTICall Methods

Answer

For an incoming call this function accepts the call and connects to the remote party.

```
HRESULT Answer ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

If the belonging call is left, you get a RemoveCallEvent for this call.

Dial

To make a call this function is called. Before doing so you must provide a dial address with the DialAddress Property.

Depending on the configuration of OSITRON CTI this method opens a keypad window or directly dials the given dial address if the dial address is not empty.

```
HRESULT Dial ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

Starting with this method the proceeding of a call is signaled as changes of the call state, e.g. OSICTICallStateDialing. Chapter 6.8 shows the state transitions of a call.

Drop

This function disconnects or discards the call.

```
HRESULT Drop ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_FAIL	if the operation fails unexpected

Remarks

Depending on the current call state multiple states may be signalled until the last state OSICTICallStateIdle. If the call is still this state nothing is signalled. To the CTI application object the RemoveCallEvent event is signalled.

Pickup

This method picks up a call ringing at a different extension port. You can supply the phone number (MSN) of this extension.

```
HRESULT Pickup (
    [in] BSTR bstrNumber
);
```

Parameters

bstrNumber

Phone number of the extension port ringing.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

You may supply an empty phone number to pick up the current call ringing.

If there is no (more) call to pickup, you may get a RemoveCallEvent for this call.

5.2.3. _IOSICTICallEvents Events

CalledAddressEvent

The CTI COM Server signals this event if the called address has been changed.

```
HRESULT CalledAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrCalledAddress
);
```

Parameters

piCall

Pointer to the call object

bstrCalledAddress

Called address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

For outgoing calls the called address is the dialed address. For incoming calls it is the own address (MSN)

CallingAddressEvent

The CTI COM Server signals this event if the calling address has been changed.

```
HRESULT CallingAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrCallingAddress
);
```

Parameters

piCall

Pointer to the call object

bstrCallingAddress

Calling address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

For outgoing calls the calling address is your own address (MSN). For incoming calls it is the address of the remote party.

ChargingEvent

The charging is returned in currency values depending on your telephone system.

```
HRESULT ChargingEvent (
    [in] IOSICTICall *piCall,
    [in] CURRENCY cyCharging
);
```

Parameters

piCall

Pointer to the call object

cyCharging

Total sum of charging in currency values since the call starts.

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

Depending on your telephone system charging is updated in different intervals.

ConnectedAddressEvent

The CTI COM Server signals this event if the connected address has been changed.

```
HRESULT ConnectedAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrConnectedAddress
);
```

Parameters

piCall

Pointer to the call object

bstrConnectedAddress

Connected address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

DialAddressEvent

The CTI COM Server signals this event if the dial address has been changed.

```
HRESULT DialAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrDialAddress
);
```

Parameters

piCall

Pointer to the call object

bstrDialAddress

Dial address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

If the keypad window opens on call of the Dial method, this event is signalled if the user enters a phone number in the keypad window.

DirectionEvent

The CTI COM Server signals this event if the direction is valid.

```
HRESULT DirectionEvent (
    [in] IOSICTICall *piCall,
    [in] EOSICTICallDirection eDir
);
```

Parameters

piCall

Pointer to the call object

eDir

Direction of type EOSICTICallDirection

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

Possible values of the type EOSICTICallDirection are listed below:

Name	Value (decimal)
OSICTICallDirectionIncoming	1
OSICTICallDirectionOutcoming	2

It is recommended to use the definitions.

NormalizedAddressEvent

The CTI COM Server signals this event if the normalized address is valid.The normalized address is the address of the remote party in a special format.

```
HRESULT NormalizedAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrNormalizedAddress
);
```

Parameters

piCall

Pointer to the call object

bstrNormalizedAddress

Dial address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

The format of the normalized address is e.g. for a german phone number, 49241946980. It consists of the fields: country code, area code and local address.

RedirectingAddressEvent

The CTI COM Server signals this event if the call is redirected The redirecting address is the address of the party which active redirects a call.

```
HRESULT RedirectingAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrRedirectingAddress
);
```

Parameters

piCall

Pointer to the call object

bstr RedirectingAddress

Redirecting address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

Not in all circumstances the redirecting address is signalled.

RedirectionAddressEvent

The CTI COM Server signals this event if the call is redirected The redirection address is the address of the party the call is redirected to.

```
HRESULT RedirectionAddressEvent (
      [in] IOSICTICall *piCall,
      [in] BSTR bstrRedirectionAddress
);
```

Parameters

piCall

Pointer to the call object

bstr RedirectionAddress

Redirection address

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

StateEvent

The CTI COM Server signals this event if the state of the call has been changed.

```
HRESULT StateEvent (
      [in] IOSICTICall *piCall,
      [in] EOSICTICallState eState
);
```

Parameters

piCall

Pointer to the call object

eState

Call state of type EOSICTICallState

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

Possible values of the type EOSICTICallState are listed below:

Name	Value (decimal)
OSICTICallStateIdle	0
OSICTICallStateRinging	1
OSICTICallStateDialing	2
OSICTICallStateProceeding	3
OSICTICallStateRingback	4
OSICTICallStateConnected	5
OSICTICallStateDisconnected	6
OSICTICallStateHold	7

The values are the same as used by the old COMinterface. It is recommended to use the definitions.
This events may be used to track the call proceeding and start actions on state transitions.

_IOSICTICallEvents Events

NameEvent

The CTI COM Server signals this event if the external ID has changed.

```
HRESULT CallingAddressEvent (
    [in] IOSICTICall *piCall,
    [in] BSTR bstrName
);
```

Parameters

piCall

Pointer to the call object

bstrName

External ID

Return Values

Normally the client returns S_OK. In some circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

5.3. The Static Collection Object

The Static Collection Object is used by the CallList property of the OSITRON CTI application object. It provides the IOSIStaticCollection interface. The prototypes of properties, methods and events are described in IDL notation.

5.3.1. IOSIStaticCollection Properties

Count

The Count is the count of elements in the collection.

```
[propget] HRESULT Count (
    [out, retval] long *plCount
);
```

Parameters

plCount

Pointer to a long variable containing the count of elements after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

NewEnum

An enumerator to enumerate the elements of the collection.

```
[propget, rerstricted] HRESULT NewEnum (
    [out, retval] IUnknown **ppiunkEnum
);
```

Parameters

ppiunkEnum

Pointer to a IUnknown pointer variable containing the enumerator after the method exits.

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_POINTER	if the parameter ppiCall is NULL or invalid
E_OUTOFMEMORY	if memory allocation failed
E_FAIL	if the operation fails unexpected

Remarks

Not directly used by the applications. The property is restricted!

5.3.2. IOSIStaticCollection Methods

Item

This function returns the selected item.

```
HRESULT Item (
    [in] VARIANT varPos,
    [out, retval] VARIANT *pvarItem
);
```

Parameters

varPos

The position of the item to retrieve.

pvarItem

Pointer to a variant variable containing the item after the method exits

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_FAIL	if the operation fails unexpected

Remarks

Normally you should use the varPos parameter as a long value with the index in the collection (in the range from 1 to Count).

Depending on the collection you can get different items in the pvarItem. For collections of objects you get an IUnknown or IDispatch interface.

Refresh

This function refreshes the collection.

```
HRESULT Refresh ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:

E_FAIL	if the operation fails unexpected

Remarks

This function is useful after an event that signalled the changing of the underlaying collection.

5.4. The ACD Application Object

The ACD application object is only supported, if you use an ACD license.

The ACD application object provides the OSIACDApplication interface with properties and methods.

To access the OSITRON ACD methods and properties, you have to create a ACD application object, e.g. in Visual Basic:

```
' Declare a variable for the OSITRON ACD object
Dim OSIACDApp As OSICTIControlCenter.OSIACDApp

    Set OSIACDApp = New OSICTIApp
```

After creating this object you can use the methods supplied by the ACD object and receive events signalled by the CTI COM Server.

5.4.1. ACDApplication Properties

Activity

This property returns the name of the distribution list, which distributes incoming and outgoing calls.

```
[propget] HRESULT Activity([out, retval] BSTR *pVal);
```

Parameters

**pVal*

The position of the item to retrieve.

Return Values

If the function succeeds, the return value is S_OK.

AddressInfo

This information show the address book entries, which are found in the incorporated address books.

```
[propget] HRESULT AddressInfo([out, retval] BSTR *pVal);
```

Parameters

**pVal*

The position of the item to retrieve.

Return Values

If the function succeeds, the return value is S_OK.

-

CallNumber

The callnumber shows the calling address of the call. Therefore you need not to create the CTI Call object.

```
[propget] HRESULT CallNumber([out, retval] BSTR *pVal);
```

Parameters

**pVal*

The position of the item to retrieve.

Return Values

If the function succeeds, the return value is S_OK.

State

This property returns the state of the ACD agent. To see any information about the agent he has to be logged in at the ACD server.

```
[propget] HRESULT State([out, retval] EOSIACDState *pVal);
```

Parameters

**pVal*

The position of the item to retrieve.

Return Values

Normally the client returns S_OK. In same circumstances you may return an error code. (In Visual Basic you cannot return an error code.)

Remarks

Possible values of the type State are listed below:

Name	Value (decimal)
OSIACDReady	0
OSIACDNotReady	1
OSIACDWorkingBeforeCall	2
OSIACDWorkingAfterCall	3
OSIACDBusyACDOutbound	4
OSIACDBusyACDInbound	5
OSIACDBusyOutbound	6
OSIACDBusyInbound	7

5.4.2. ACDApplication Methods

SetACDPause

This method deactivates the agent for ACD. Calls will not be distributed to agents any longer.

```
HRESULT SetACDPause();
```

Return Values

If the function succeeds, the return value is S_OK.

SetACDReady

This method signalize that its in the state 'ready for ACD'. After this, calls can be distributed to the agent. It is the opposite of SetACDPause.

```
HRESULT SetACDReady();
```

Return Values

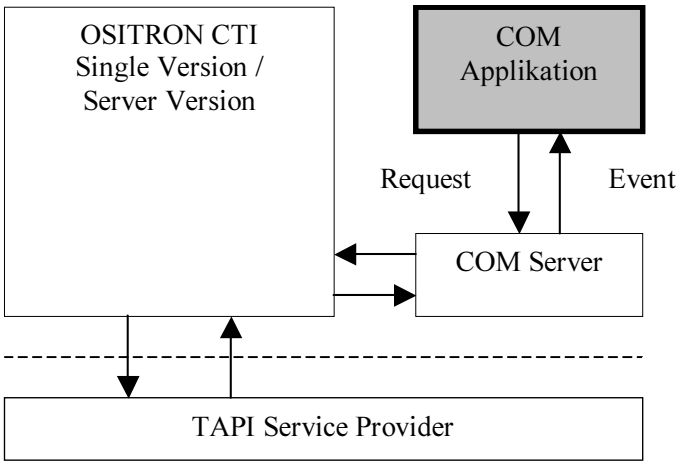
If the function succeeds, the return value is S_OK.

6. Programming the COM Interface ITelApp

This interface is only supported for backward compatibility. Please use the interface described in chapter 5.

6.1. Overview

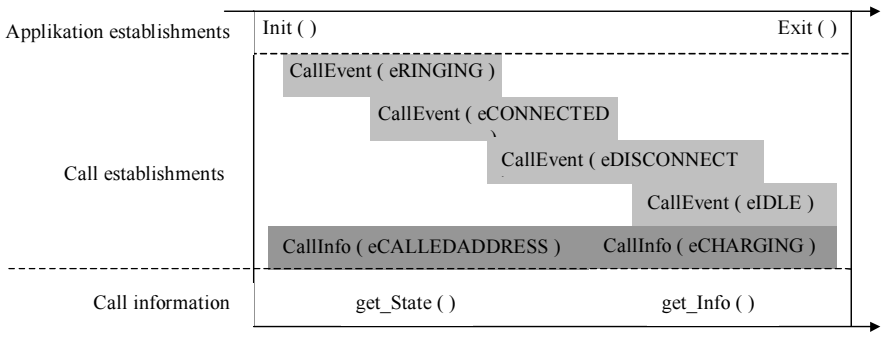
Interoperation between COM application, COM Server, OSITRON CTI and TAPI.



The following chapters describes the old interface introduced with version 1.52 of our CTI software.

6.2. Application Startup

The following figure illustrates the normal procedure of application start-up and call handling.



6.3. Starting / Stopping the COM Server

6.3.1. Init

This function announce the COM Server to OSITRON CTI to receive messages.

```
HRESULT Init ();
```

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available, but it has no effect.

6.3.2. Exit

This function will shutdown the COM Server and OSITRON CTI.

```
HRESULT Exit  ();
```

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available, but it has no effect.

6.3.3. ServerState

With this function you can read the actually COM Server state at any time.

```
HRESULT ServerState (
      [out, retval] long *pVal
);
```

Parameters

pVal

Pointer to a long value where one of the following values are deliver.

Name	Value (decimal)
READY	0
OUT_OF_SERVICE	1
UNINITIALIZED	2

ACDPREPARE	3
ACDOUTBOUND	4
ACDINBOUND	5
ACDWORKAFTERCALL	6
ACDREADY	7

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available. The return value is always READY.

6.4. Application Control

6.4.1. About

Shows a dialog box with version information.

```
HRESULT About ();
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value may be:
HRESULT_FROM_WIN32 (ERROR_BUSY) if one of the CTI COM server dialogs is still open

6.4.2. HideDown

This function hides the OSITRON CTI application window.

```
HRESULT HideDown ();
```

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available, but it has no effect.

6.4.3. ShowUp

This function shows the OSITRON CTI application window.

```
HRESULT ShowUp ();
```

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available, but it has no effect.

6.4.4. ToggleMinMode

Call this function to switch between the MinMode, Toolbar and Infobar are visible and MaxMode, the complete window is visible.

```
HRESULT ToggleMinMode ();
```

Return Values

The return value is always S_OK.

Remarks

For reason of compatibility this function is still available, but it has no effect.

6.5. Handling of Calls

6.5.1. Drop

Call this function to disconnect a call every time.

```
HRESULT Drop ();
```

Return Values

The return value is always S_OK, even there was no call to drop.

6.5.2. Info

Call this function to get the Information about an incoming or outgoing call.

```
HRESULT Info (
    [out, retval] VARIANT *pVal
);
```

Return Values

If the function succeeds, the return value is S_OK.

If the function fails, the return value is E_POINTER when the pointer *pVal* is NULL.

Parameters

pVal

Pointer to a VARIANT value where the actual call information is stored.

Remarks

The actual call information string is stored as a BSTR in the VARIANT type.

6.5.3. State

Call this function to get the state of an incoming or outgoing call.

```
HRESULT State (
    [out, retval] long *pVal
);
```

Parameters

pVal

Pointer to a long value where one of the following values are deliver.

Name	Value (decimal)
EIDLE	1
ERINGING	2
Edialing	3
ePROCEEDING	4
Eringback	5
eCONNECTED	6
eDISCONNECTED	7

Return Value

If the function succeeds, the return value is S_OK.

If the function fails, the return value is E_POINTER when the pointer *pVal* is NULL.

Remarks

You can use this function to poll for new states too without using the CALLEVENT function. When ever the state is changing, you can store the new state for the later processing.

All these information are only available after an eCALLSTATE message.

6.6. Signaling of Call Information

6.6.1. CallEvent

This function is called from the COM Server to signal a change of a callstate.

```
HRESULT CallEvent (
      [in] int Info
);
```

Parameters

Info

A int value which signal the call event. Bellow you find a list of all values.

Name	Value (decimal)
eCALLSTATE	0
ECALLINFO_CALLED_ADDRESS	1
ECALLINFO_CALLING_ADDRESS	2
ECALLINFO_CONNECTED_ADDRESS	3
ECALLINFO_REDIRECTIION_ADDRESS	4
ECALLINFO_CHARGING	5
ECALLINFO_DATETIME	6
ECALLINFO_NAME	7
ECALLINFO_DISPLAY	8
EACDSTATE	9
eACDINFO_ADDRESS	10

6.7. Definitions

CallState parameters

Name	Value (decimal)
EIDLE	1

ERINGING	2
EDIALING	3
ePROCEEDING	4
eRINGBACK	5
eCONNECTED	6
eDISCONNECTED	7

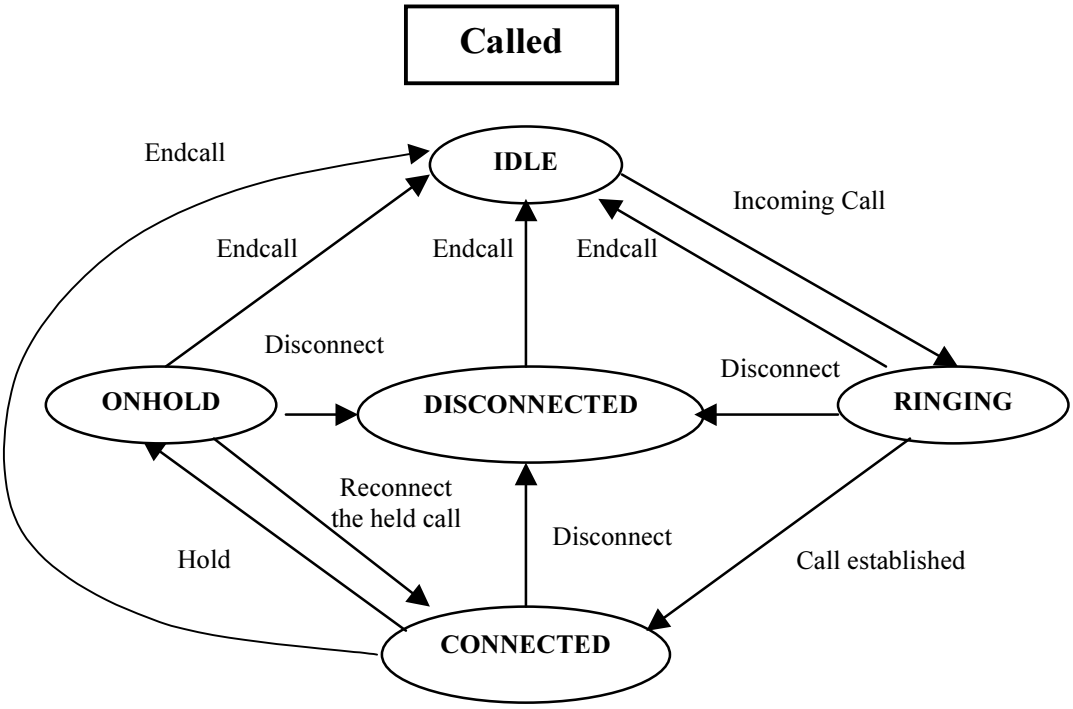
CallEvent parameters

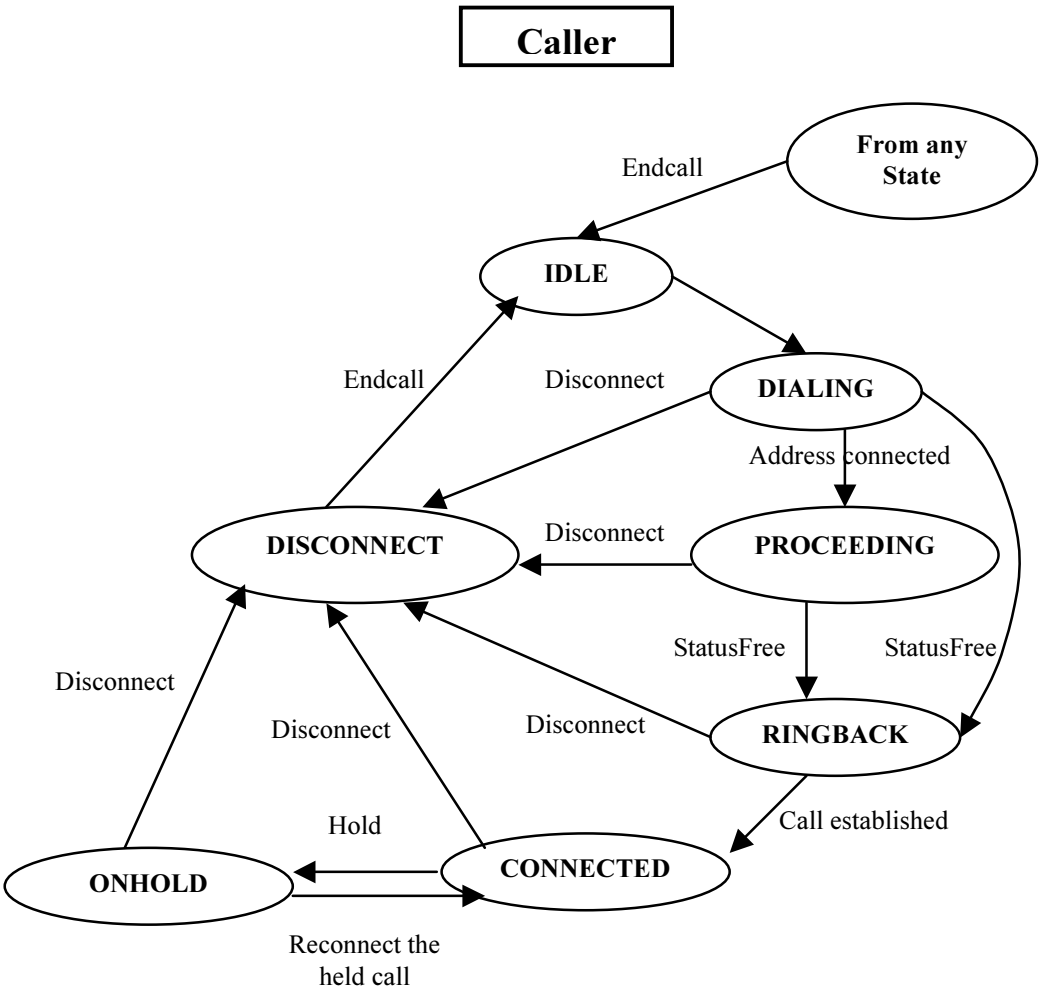
Name	Value (decimal)
eCALLSTATE	1
eCALLINFO_CALLED_ADDRESS	2
eCALLINFO_CALLING_ADDRESS	3
eCALLINFO_CONNECTED_ADDRESS	4
eCALLINFO_REDIRECTION_ADDRESS	5
eCALLINFO_CHARGING	6
eCALLINFO_DATETIME	7
eCALLINFO_NAME	8
eCALLINFO_DISPLAY	9

ServerState parameters

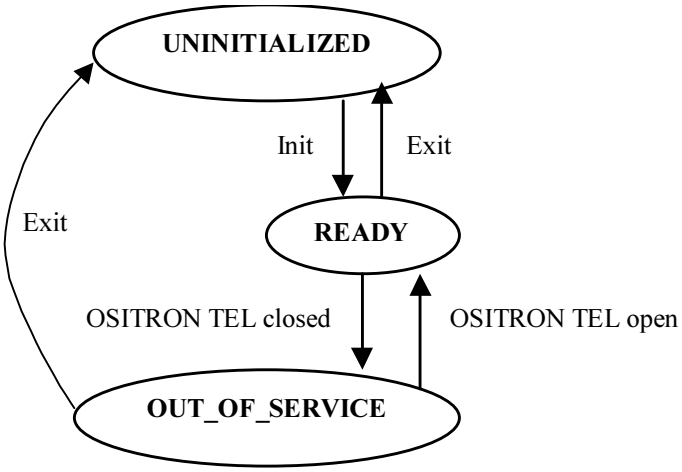
Name	Value (decimal)
READY	0
OUT_OF_SERVICE	1
UNINITIALIZED	2
ACDPREPARE	3
ACDOUTBOUND	4
ACDINBOUND	5
ACDWORKAFTERCALL	6
ACDREADY	7

6.8. States of the Call





6.9. States of the Server



7. Samples in Visual Basic

7.1. Sample using the new Interfaces

7.1.1. Using the new COM Server

To use the new COM Server you need a correctly installed and registered OSITRON CTI 3.0.

At first you declare a variable for the COM Object.

```
Dim OSICTIClient As OSICTIControlCenter.OSICTIApp
```

Then you must create a new object of OSICTIApp at start-up.

In the example it is done in the “Load” subroutine of the form.

```
Set OSICTIClient = New OSICTIApp
```

Now you have a new object of the COM Server. With this object you can call each function that is implemented in the interface.

7.1.2. Using Notification

To receive events from the COM Server you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

The events reported to this object are the NewCallEvent and the CallRemoveEvent.to indicate that a new call is arrived resp. a call is leaving.

7.1.3. Retrieving the Call List

To get a list of all established calls at application startup time you can use the CallList-property. The interface contained in the resulting variant is an IOSIStaticCollection interface. The elements returned by the Item method are interface pointers which can converted to an IOSICTICall interface by QueryInterface.

7.1.4. Retrieving Information of a Call

To retrieve information you use the methods in the IOSICTICall interface of a call object. There are several information, such as call state or addresses of the call related parties.

7.1.5. Using Call Information Notification

To receive call information events you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Public WithEvents CTICallClient As OSICTIControlCenter.OSICTICall
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

The reported events are equivalent to the call information retrieving functions.

7.1.6. Making a Call

It is a simple task to establish an outgoing call. There are two methods available to establish a new outgoing call: via assisted telephony or with the CreateCall-method of the COM-server interface.

The following prototype handles the call to associated telephony:

```
Rem *** Declare Function from TAPI.Dll to make a call ***
Private Declare Function tapiRequestMakeCall Lib "tapi32" _
    (ByVal lpDestAddr As String, _
    ByVal lpAppName As String, ByVal lpCalledParty As String, _
    ByVal lpComment As String) As Long
```

Example: `tapiRequestMakeCall “11”, “TestApp”, “”, “”`

Note: You use the OSITRON CTI 3.0 in assisted telephony only when you have selected this in the configuration.

The method via the COM-server interface is used as in the following:

```
Dim NewCall As OSICTIControlCenter.IOSICTICall

Set NewCall = OSICTIClient.CreateCall
NewCall.DialAddress = “11”
NewCall.Dial
```

The making of a call via the COM-interface is always done vie the OSITRON CTI application, independent of the setting for assisted telephony.

All calls you made yourself are also reported in the NewCallEvent-interface.

7.1.7. Drop a Call

With the Drop-function of the associated call-object you can disconnect a call anytime.

If the call is released as a result the CallRemoveEvent is signalled.

7.1.8. Code Listing for Visual Basic

The sample consists of the following 3 files:

- OSICTIClientSample.vbp is the Project file
- OSICTIClientSample.frm contains the main formular and the main code
- OSICTICallClientClass.cls contains a class which can be used for receiving events per call. This is necessary, because you can‘t use an array of „Dim WithEvents“-variables.

In addition to the fundamental principles of the programming the OSITRON-CTI-COM-server the sample handles an own call list in a TabStrip-control.

OSICTIClientSample.frm

```
Option Explicit

Rem *** Declare Function from TAPI.Dll to make a Call ***
Private Declare Function tapiRequestMakeCall Lib "tapi32" _
    (ByVal lpDestAddr As String, _
    ByVal lpAppName As String, ByVal lpCalledParty As String, _
    ByVal lpComment As String) As Long

Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp
'Connection Point
Dim OSIACDApp As OSICTIControlCenter.OSIACDApp

Private Sub About_Click()
    OSICTIClient.About
End Sub

Private Sub ACDACTIVE_Click()
    OSIACDApp.SetACDReady
End Sub

Private Sub ACDPause_Click()
    OSIACDApp.SetACDPause
End Sub

Private Sub CallTabStrip_Click()
    Dim ActCallClient As OSICTICallClientClss

    Set ActCallClient =
CallTabStrip.Tabs.Item(CallTabStrip.SelectedItem.Index).Tag
    Call UpdateInfo(ActCallClient)
End Sub

Private Sub Clear_Click()
    StateInfo.Clear         'Clear list of all shown Messages
End Sub

Private Sub ConfigureAddressBooks_Click()
    OSICTIClient.ConfigureAddressBooks
End Sub

Private Sub ConfigureCallDeflectManager_Click()
    OSICTIClient.ConfigureCallDeflectManager
End Sub

Private Sub ConfigureCallForwardManager_Click()
    OSICTIClient.ConfigureCallForwardManager
End Sub

Private Sub ConfigureHotKeys_Click()
    OSICTIClient.ConfigureHotKeys
End Sub

Private Sub Dial_Click()    'Dial the given Number
    'tapiRequestMakeCall DialNo.Text, "TestApp", "", ""
    'or alternative:
    Dim NewCall As OSICTIControlCenter.IOSICTICall

    Set NewCall = OSICTIClient.CreateCall
    NewCall.DialAddress = DialNo.Text
    NewCall.Dial
End Sub

Private Sub Drop_Click()
    Dim ActCallClient As OSICTICallClientClss
    Dim i As Integer

    For i = 1 To CallTabStrip.Tabs.Count
        If CallTabStrip.Tabs.Item(i).Selected Then
            Set ActCallClient = CallTabStrip.Tabs.Item(i).Tag
            ActCallClient.CTICallClient.Drop
```

```
        End If
    Next
End Sub

Private Sub Form_Load()
    Dim i As Integer
    Dim CallColl As IOSIStaticCollection

    Set OSICTIClient = New OSICTIApp
    Set OSIACDApp = New OSICTIApp
    Set CallColl = OSICTIClient.CallList
    CallTabStrip.Tabs.Remove (1)
    For i = 1 To CallColl.Count
        OSICTIClient_NewCallEvent OSICTIClient, CallColl.Item(i)
    Next
End Sub

Private Sub MakeConfiguration_Click()
    OSICTIClient.MakeConfiguration
End Sub

Private Sub OSICTIClient_ACDAppStateChanged()
    Dim State As String

    Dialog.StateInfo.AddItem "ACD State changed"

    Select Case OSIACDApp.State
        Case OSIACDBusyACDInbound
            State = "OSIACDBusyACDInbound"
        Case OSIACDBusyACDOutbound
            State = "OSIACDBusyACDOutbound"
        Case OSIACDBusyInbound
            State = "OSIACDBusyInbound"
        Case OSIACDBusyOutbound
            State = "OSIACDBusyOutbound"
        Case OSIACDNotReady
            State = "OSIACDNotReady"
        Case OSIACDReady
            State = "OSIACDReady"
        Case OSIACDWorkingBeforeCall
            State = "OSIACDWorkingBeforeCall"
        Case OSIACDWorkingAfterCall
            State = "OSIACDWorkingAfterCall"
    End Select

    Dialog.StateInfo.AddItem "     " + State

    InfoACDState.Caption = State
    InfoActivity.Caption = OSIACDApp.Activity
    InfoAddressInfo.Caption = OSIACDApp.AddressInfo
    InfoCallNumber.Caption = OSIACDApp.CallNumber
End Sub

Private Sub OSICTIClient_CallRemoveEvent(ByVal piCTI As
OSICTIControlCenter.IOSICTIApp, ByVal piCall As
OSICTIControlCenter.IOSICTICall)
    Dim ActCall As IOSICTICall
    Dim ActCallClient As OSICTICallClientClss
    Dim i As Integer

    For i = 1 To CallTabStrip.Tabs.Count
        Set ActCallClient = CallTabStrip.Tabs.Item(i).Tag
        Set ActCall = ActCallClient.CTICallClient
        If piCall Is ActCall Then
            If CallTabStrip.Tabs.Item(i).Selected Then
                If CallTabStrip.Tabs.Count > 1 Then
                    If i = 1 Then
                        CallTabStrip.Tabs.Item(2).Selected = True
                    Else
```

```
                        CallTabStrip.Tabs.Item(i - 1).Selected =
True
                    End If
                End If
            End If
            Set CallTabStrip.Tabs(i).Tag = Nothing 'neccessary,
otherwise the Reference will not be freed
            CallTabStrip.Tabs.Remove (i)
            Exit For
        End If
    Next
    If CallTabStrip.Tabs.Count = 0 Then
        Dialog.Frame1.Visible = False
        Dialog.Frame2.Visible = False
    End If
End Sub

Private Sub OSICTIClient_NewCallEvent(ByVal piCTI As
OSICTIControlCenter.IOSICTIApp, ByVal piCall As
OSICTIControlCenter.IOSICTICall)
    Dim NewTab As ITab
    Dim ActCallClient As OSICTICallClientClss
    Dim ActCallClient2 As IUnknown

    Set ActCallClient = New OSICTICallClientClss
    Set ActCallClient2 = ActCallClient
    Set ActCallClient.CTICallClient = piCall
    If CallTabStrip.Tabs.Count = 0 Then
        Dialog.Frame1.Visible = True
        Dialog.Frame2.Visible = True
        Call UpdateInfo(ActCallClient)
    End If
    Set NewTab = CallTabStrip.Tabs.Add
    Set ActCallClient.m_ITab = NewTab
    NewTab.Tag = ActCallClient2
End Sub

Private Sub SelectLanguage_Click()
    OSICTIClient.SelectLanguage
End Sub

Private Sub SelectLines_Click()
    OSICTIClient.SelectLines
End Sub

Private Sub UpdateInfo(ByVal CallClient As OSICTICallClientClss)
    InfoCalledAdr.Caption = CallClient.CTICallClient.CalledAddress
    InfoCallingAdr.Caption =
CallClient.CTICallClient.CallingAddress
    InfoCallState.Caption =
CallClient.GetCallState(CallClient.CTICallClient.State)
    InfoCharging.Caption = CallClient.CTICallClient.Charging
    InfoConnectedAdr.Caption =
CallClient.CTICallClient.ConnectedAddress
    InfoDateTime.Caption = CallClient.CTICallClient.DateTime
    InfoDisplay.Caption = CallClient.CTICallClient.Display
    InfoName.Caption = CallClient.CTICallClient.Name
    InfoRedirecting.Caption =
CallClient.CTICallClient.RedirectingAddress
    InfoRedirection.Caption =
CallClient.CTICallClient.RedirectionAddress
End Sub
```

OSICTICallClientClass.cls

```
Option Explicit

Public m_State As String
Public m_ITab As ITab
Public WithEvents CTICallClient As OSICTIControlCenter.OSICTICall
```

```
Private Sub CTICallClient_CalledAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrCalledAddress As String)
    Dialog.StateInfo.AddItem "CallInfo Called Address:"
    Dialog.StateInfo.AddItem "     " + bstrCalledAddress
    If m_ITab.Selected Then
        Dialog.InfoCalledAdr.Caption = bstrCalledAddress
    End If
End Sub

Private Sub CTICallClient_CallingAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrCallingAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Calling Address:"
    Dialog.StateInfo.AddItem "     " + bstrCallingAddress
    If m_ITab.Selected Then
        Dialog.InfoCallingAdr.Caption = bstrCallingAddress
    End If
End Sub

Private Sub CTICallClient_CanonicalAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrCanonicalAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Canonical Address:"
    Dialog.StateInfo.AddItem "     " + bstrCanonicalAddress
    If m_ITab.Selected Then
        'Dialog.InfoCanonicalAdr.Caption = bstrCanonicalAddress
    End If
End Sub

Private Sub CTICallClient_ChargingEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal varCharging As Currency)
    Dialog.StateInfo.AddItem "CallInfo Charging:"
    Dialog.StateInfo.AddItem "     " + varCharging
    If m_ITab.Selected Then
        Dialog.InfoCharging.Caption = varCharging
    End If
End Sub

Private Sub CTICallClient_ConnectedAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrConnectedAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Connected Address:"
    Dialog.StateInfo.AddItem "     " + bstrConnectedAddress
    If m_ITab.Selected Then
        Dialog.InfoConnectedAdr.Caption = bstrConnectedAddress
    End If
End Sub

Private Sub CTICallClient_DateTimeEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal varDateTime As Date)
    Dialog.StateInfo.AddItem "CallInfo DateTime:"
    Dialog.StateInfo.AddItem "     " + varDateTime
    If m_ITab.Selected Then
        Dialog.InfoDateTime.Caption = varDateTime
    End If
End Sub

Private Sub CTICallClient_DialableAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrDialableAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Dialable Address:"
    Dialog.StateInfo.AddItem "     " + bstrDialableAddress
    If m_ITab.Selected Then
        'Dialog.InfoDialableAdr.Caption = bstrDialableAddress
    End If
End Sub

Private Sub CTICallClient_DialAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrDialAddress As String)
```

```
    Dialog.StateInfo.AddItem "CallInfo Dial Address:"
    Dialog.StateInfo.AddItem "     " + bstrDialAddress
    If m_ITab.Selected Then
        'Dialog.InfoDialAdr.Caption = bstrDialAddress
    End If
End Sub

Private Sub CTICallClient_DirectionEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal eDirection As
OSICTIControlCenter.EOSICTICallDirection)
    Dialog.StateInfo.AddItem "CallInfo Direction:"
    Dialog.StateInfo.AddItem "     " + eDirection
    If m_ITab.Selected Then
        Dialog.InfoDirection.Caption = eDirection
    End If
End Sub

Private Sub CTICallClient_DisplayEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrDisplay As String)
    Dialog.StateInfo.AddItem "CallInfo Display:"
    Dialog.StateInfo.AddItem "     " + bstrDisplay
    If m_ITab.Selected Then
        Dialog.InfoDisplay.Caption = bstrDisplay
    End If
End Sub

Private Sub CTICallClient_NameEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrName As String)
    Dialog.StateInfo.AddItem "CallInfo Name:"
    Dialog.StateInfo.AddItem "     " + bstrName
    If m_ITab.Selected Then
        Dialog.InfoName.Caption = bstrName
    End If
End Sub

Private Sub CTICallClient_NormalizedAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrNormalizedAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Normalized Address:"
    Dialog.StateInfo.AddItem "     " + bstrNormalizedAddress
    If m_ITab.Selected Then
        'Dialog.InfoNormalizedAdr.Caption = bstrNormalizedAddress
    End If
End Sub

Private Sub CTICallClient_RedirectingAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrRedirectingAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Redirecting Address:"
    Dialog.StateInfo.AddItem "     " + bstrRedirectingAddress
    If m_ITab.Selected Then
        Dialog.InfoRedirecting.Caption = bstrRedirectingAddress
    End If
End Sub

Private Sub CTICallClient_RedirectionAddressEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal bstrRedirectionAddress As
String)
    Dialog.StateInfo.AddItem "CallInfo Redirection Address:"
    Dialog.StateInfo.AddItem "     " + bstrRedirectionAddress
    If m_ITab.Selected Then
        Dialog.InfoRedirection.Caption = bstrRedirectionAddress
    End If
End Sub

Private Sub CTICallClient_StateEvent(ByVal piCall As
OSICTIControlCenter.IOSICTICall, ByVal eState As
OSICTIControlCenter.EOSICTICallState)
    Dim State As String
    Dim ActCallClient As OSICTICallClientClss
```

```
    Dim ActCall As IOSICTICall
    Dim i As Integer

    State = GetCallState(eState)
    Dialog.StateInfo.AddItem "CallState:"
    Dialog.StateInfo.AddItem "     " + State
    If m_ITab.Selected Then
        Dialog.InfoCallState.Caption = State
    End If

    For i = 1 To Dialog.CallTabStrip.Tabs.Count
        Set ActCallClient = Dialog.CallTabStrip.Tabs.Item(i).Tag
        Set ActCall = ActCallClient.CTICallClient
        If piCall Is ActCall Then
            Set Dialog.CallTabStrip.SelectedItem =
Dialog.CallTabStrip.Tabs(i)
        End If
    Next

End Sub

Public Function GetCallState(ByVal eState As
OSICTIControlCenter.EOSICTICallState) As String
    Dim State As String

    Select Case eState
        Case OSICTICallStateIdle
            State = "Idle"
        Case OSICTICallStateRinging
            State = "Ringing"
        Case OSICTICallStateDialing
            State = "Dialing"
        Case OSICTICallStateProceeding
            State = "Proceeding"
        Case OSICTICallStateRingback
            State = "Ringback"
        Case OSICTICallStateConnected
            State = "Connected"
        Case OSICTICallStateDisconnected
            State = "Disconnected"
        Case OSICTICallStateHold
            State = "Hold"
    End Select
    GetCallState = State
End Function
```

7.2. Sample using the old Interface

7.2.1. Using the COM Server

To use the COM Server you need the DLL file and two function calls.

At first you declare a variable for the COM Object.

```
Dim OsiTelEvent As OTTELCOMLib.TelApp
```

Then you must create a new object of OTTELCOMLib.TelApp at start-up.

In the example it is done in the “Load” function of the dialog.

```
Set OsiTelEvent = New OTTELCOMLib.TelApp
```

Now you have a new object of the COM Server. With this object you can call each function that is implemented in the interface.

7.2.2. Using Notification

To receive Events from the COM Server you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Dim WithEvents OsiTelEvent As OTTELCOMLib.TelApp
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

7.2.3. Starting / Stopping

With the “Init()” and “Exit()” functions you can register your application at the COM Server. Exit will Stop the communication with the Server.

You will only receive intonations in the phase between successfully calling Init() and Exit().

7.2.4. Retrieving the Server State

At any time it is possible to get the actual COM Server state.

You only need to call the “ServerState” function from the COM Server.

```
Dim lTest As Long
lTest = ConnectPoint.ServerState
For more Information please read “5.2 Start / Stopping the COM
Server”.
```

7.2.5. Application Control

With the „ShowUp()“, „HideDown“ and „ToggleMinMode“ functions you are able to control the OSITRON CTI application window.

With „ShowUp()“ and „HideDown()“ you can show or hide the application window only if the „Init()“ function was called. Otherwise nothing happends.

7.2.6. Retrieving information of the Call

All messages are handled by the “CallEvent” function which send you the state of an incoming or outgoing phone call.

In the CallEvent handle you can retrieve further intonations by calling get_State and get_Info.

```
Dim lState As Long
Dim StringDateTime As Variant

Private Sub OsiTelEvent_CallEvent(ByVal Info As Long)
    Select Case Info
        Case eCALLSTATE
            lState =  OsiTelEvent.State
        Case eDATETIME
            StringDateTime = OsiTelEvent.Info
End Select
```

7.2.7. Making a Call

It is a simple task to establish an outgoing call. The following prototype handles the call to associated telephony:

```
Private Declare Function tapiRequestMakeCall Lib "tapi32" _
    (ByVal lpDestAddr As String, _
```

```
    ByVal lpAppName As String, ByVal lpCalledPrty As String, _
    ByVal lpComment As String) As Long
```

Example: `tapiRequestMakeCall "11", "TestApp", "", ""`

7.2.8. Drop a Call

With the „Drop()“ function you can disconnect a call anytime, if the „Init()“ function was called before. If not, nothing happens.

7.2.9. Code Listing for Visual Basic

```
Public Enum TCINFOTYPE
    eCALLSTATE
    eCALLINFO_CALLED_ADDRESS
    eCALLINFO_CALLING_ADDRESS
    eCALLINFO_CONNECTED_ADDRESS
    eCALLINFO_REDIRECTIUON_ADDRESS
    eCALLINFO_CHARGING
    eCALLINFO_DATETIME
    eCALLINFO_NAME
    eCALLINFO_DISPLAY
End Enum

Public Enum TCCALLSTATE
    eIDLE
    eRINGING
    eDIALING
    ePROCEEDING
    eRINGBACK
    eCONNECTED
    eDISCONNECTED
End Enum

Public Enum TCSERVERSTATE
    READY
    OUT_OF_SERVICE
    UNINITIALIZED
End Enum

Rem *** Declare Function from TAPI.Dll to make a call ***
Private Declare Function tapiRequestMakeCall Lib "tapi32" _
    (ByVal lpDestAddr As String, _
    ByVal lpAppName As String, ByVal lpCalledParty As String, _
    ByVal lpComment As String) As Long

Dim WithEvents OsiTelEvent As OTTELCOMLib.TelApp    'Connection
Point
Dim Info As TCINFOTYPE                              'Info Message
from "CallEvent"
Dim State As TCCALLSTATE                            'State Message
from "get_State"
Dim ServerState As TCSERVERSTATE                    'ServerState
from "get_ServerState"
Dim StateCallInfo As Variant                        'Information
String from "get_Info"

Private Sub About_Click()
    OsiTelEvent.About       'Show dialog box with Copyright
information
End Sub

Private Sub Clear_Click()
    StateInfo.Clear         'Clear list of all shown messages
End Sub
```

```
Private Sub Dial_Click()    'dial the given number
    tapiRequestMakeCall DialNo.Text, "TestApp", "", ""
End Sub

Private Sub Drop_Click()
    OsiTelEvent.Drop        'break connection
End Sub

Private Sub EnableIncomingCallsTrue_Click()
    OsiTelEvent.EnableIncomingCalls (True)
End Sub

Private Sub EnableIncomingFalse_Click()
    OsiTelEvent.EnableIncomingCalls (False)
End Sub

Private Sub EnableOutgoingCallsTrue_Click()
    OsiTelEvent.EnableOutgoingCalls (True)
End Sub

Private Sub EnableOutgoingFalse_Click()
    OsiTelEvent.EnableOutgoingCalls (False)
End Sub

Private Sub Exit_Click()
    OsiTelEvent.Exit        'Notice from OSITRON CTI
End Sub

Private Sub Form_Load()
    Set OsiTelEvent = New OTTELCOMLib.TelApp
End Sub

Private Sub HideApp_Click() 'hide the OSITRON CTI application
    OsiTelEvent.HideDown
End Sub

Private Sub ShowApp_Click() 'show the OSITRON CTI application
    OsiTelEvent.ShowUp
End Sub

Private Sub Toggle_Click()  'toggle between MinMode and MaxMode
    OsiTelEvent.ToggleMinMode
End Sub

Private Sub Init_Click()
    OsiTelEvent.Init        'Announce to OSITRON CTI
End Sub

Private Sub OsiTelEvent_CallEvent(ByVal Info As Long)
    Select Case Info
        Rem *** Event: eCALLSTATE ***
        Case eCALLSTATE
            Select Case OsiTelEvent.State
                Rem *** State: eIDLE ***
                Case eIDLE
                    StateInfo.AddItem "eCallState:"
                    StateInfo.AddItem "     eIDLE"
                    InfoCalledAdr.Caption = ""
                    InfoCallingAdr.Caption = ""
                    InfoConnectedAdr.Caption = ""
                    InfoRedirection.Caption = ""
                    InfoCharging.Caption = ""
                    InfoDateTime.Caption = ""
                    InfoName.Caption = ""
                    InfoDisplay.Caption = ""
                    InfoCallState.Caption = "eIDLE"
                Rem *** State: eIDLE ***
                Case eRINGING
                    StateInfo.AddItem "eCallState:"
                    StateInfo.AddItem "     eRINGING"
```

```
                InfoCallState.Caption = "eRINGING"
            Rem *** State: eDIALING ***
            Case eDIALING
                StateInfo.AddItem "eCallState:"
                StateInfo.AddItem "     eDIALING"
                InfoCallState.Caption = "eDIALING"
            Rem *** State: ePROCEEDING ***
            Case ePROCEEDING
                StateInfo.AddItem "eCallState:"
                StateInfo.AddItem "     ePROCEEDING"
                InfoCallState.Caption = "ePROCEEDING"
            Rem *** State: eRINGBACK ***
            Case eRINGBACK
                StateInfo.AddItem "eCallState:"
                StateInfo.AddItem "      eRINGBACK"
                InfoCallState.Caption = "eRINGBACK"
            Rem *** State: eCONNECTED ***
            Case eCONNECTED
                StateInfo.AddItem "eCallState:"
                StateInfo.AddItem "     eCONNECTED"
                InfoCallState.Caption = "eCONNECTED"
            Rem *** State: eDISCONNECTED ***
            Case eDISCONNECTED
                StateInfo.AddItem "eCallState:"
                StateInfo.AddItem "     eDISCONNECTED"
                InfoCallState.Caption = "eDISCONNECTED"
        End Select

    Rem *** Event: eCALLINFO_CALLED_ADDRESS ***
    Case eCALLINFO_CALLED_ADDRESS
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo Called:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoCalledAdr.Caption = StateCallInfo

    Rem *** Event: eCALLINFO_CALLING_ADDRESS ***
    Case eCALLINFO_CALLING_ADDRESS
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo Calling:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoCallingAdr.Caption = StateCallInfo

    Rem *** Event: eCALLINFO_CONNECTED_ADDRESS ***
    Case eCALLINFO_CONNECTED_ADDRESS
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo Connected:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoConnectedAdr.Caption = StateCallInfo

    Rem *** Event: eCALLINFO_REDIRECTIUON_ADDRESS ***
    Case eCALLINFO_REDIRECTIUON_ADDRESS
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo Redirection:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoRedirection.Caption = StateCallInfo

    Rem *** Event: eCALLINFO_CHARGING ***
    Case eCALLINFO_CHARGING
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo Charging:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoCharging.Caption = StateCallInfo

    Rem *** Event: eCALLINFO_DATETIME ***
    Case eCALLINFO_DATETIME
        StateCallInfo = OsiTelEvent.Info
        StateInfo.AddItem "eCallInfo DateTime:"
        StateInfo.AddItem "     " + StateCallInfo
        InfoDateTime.Caption = StateCallInfo
```

```
        Rem *** Event: eCALLINFO_NAME ***
        Case eCALLINFO_NAME
            StateCallInfo = OsiTelEvent.Info
            StateInfo.AddItem "eCallInfo Name:"
            StateInfo.AddItem "     " + StateCallInfo
            InfoName.Caption = StateCallInfo

        Rem *** Event: eCALLINFO_DISPLAY ***
        Case eCALLINFO_DISPLAY
            StateCallInfo = OsiTelEvent.Info
            StateInfo.AddItem "eCallInfo Display:"
            StateInfo.AddItem "     " + StateCallInfo
            InfoDisplay.Caption = StateCallInfo
    End Select
End Sub
```

8. Sample in Visual C++

8.1. Using the COM Server

To use the COM Server you need a correctly installed and registered OSITRON CTI 3.0.

At first you have to import the Type Library of “OSICTIControlCenter 3.0” into your project .You can do this by processing the following steps:

1. You have to copy the OSITELCOMServer.tlb into the include directory .

2. Add the string “ #import"OSITELCOMServer.tlb" no_namespace named_guids ” in the StdAfx.h .

3. Build your project .

After the build you can find two files in the Output Debug directory “ OSITELCOMServer.tlh and “OSITELCOMServer.tli “ , here you find informations about all functions of OSITELCOMServer. The next step is to initialize the OSITELCOMServer:

1. Call the Function **CoInitialize(NULL)** before you use the OSITELCOMServer (for example in the constructor of your application or in the **InitInstance()** if you have a dialog-based application) .

2. Call the function **CoUninitialize()** at the end of your application (for example in the destructor or in the **ExitInstance()**) .

After you have initialized the COM Server you can create an instance of it:

```
IOSICTIAppPtr m_ptrCtiApp;
m_ptrCtiApp.CreateInstance(__uuidof(OSICTIApp));
```

Now you are ready to use the OSITELCOMServer.

8.2. Using the Features of the COM Interface

Same functions are available on the application interface. Call related functions can be called on the call interface.

For example :

```
IOSICTIAppPtr m_ptrCtiApp;
m_ptrCtiApp.CreateInstance(__uuidof(OSICTIApp));
```

Send SMS:

```
m_ptrCtiApp->PopupSMSDialog("",(class
_bstr_t)m_csNumber,"","","","");
```

Call select lines dialog:

```
m_ptrCtiApp->SelectLines();
```

Call dialog to configure the address book:

```
m_ptrCtiApp->ConfigureAddressBooks();
```

8.3. Using Notification of COM Server

To receive events from the OSITELCOMServer you must establish a connection point, which can be called by the server. To establish a connection point you have to do the following steps :

1. Declare a new class derived from **CCmdTarget** .The function **EnableAutomation ()** must be called in the constructor of the class object to enable the COM automation .

2. Declare and define message and interface maps for this class (see OSICTIAppEvents.cpp and OSICTIAppEvents.h files).Please note that the functions ID´s must be the same as in the interface declaration (OSITELCOMServer.tlh, OSITELCOMServer.tli).

```
BEGIN_DISPATCH_MAP(OSICTIAppEvents, CCmdTarget)
//{{AFX_DISPATCH_MAP(OSICTIAppEvents)
DISP_FUNCTION_ID(OSICTIAppEvents, "NewCallEvent", 1 ,NewCallEvent,
VT_BOOL, VTS_UNKNOWN VTS_UNKNOWN)
DISP_FUNCTION_ID(OSICTIAppEvents, "CallRemoveEvent", 2 ,
CallRemoveEvent, VT_BOOL, VTS_UNKNOWN VTS_UNKNOWN)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
BEGIN_INTERFACE_MAP(OSICTIAppEvents, CCmdTarget)
INTERFACE_PART(OSICTIAppEvents, DIID__IOSICTIAppEvents,
Dispatch)
END_INTERFACE_MAP()
```

3. Declare functions for handling of events:

```
HRESULT OSICTIAppEvents::NewCallEvent(IOSICTIApp *piApp,
IOSICTICall *piCall)
{
//Your code for handling by this event
}
HRESULT OSICTIAppEvents::CallRemoveEvent(IOSICTIApp *piApp,
IOSICTICall *piCall)
{
//Your code for handling by this event
}
```

4. Call the function **AfxConnectionAdvise(…)**.Include " **AFXCTL.H** " to your project to use this function.

Example :

```
OSICTIAppEvents *m_OSICTIAppEvents;
m_OSICTIAppEvents = new OSICTIAppEvents(m_ptrCtiApp,this);
AfxConnectionAdvise(m_ptrCtiApp, IOSICTIAppEvents,
m_OSICTIAppEvents -> GetInterface(&IID_IUnknown), TRUE,
&m_dwCtiCookie);
```

Now you can use NewCallEvent() and RemoveCallEvent() of CTI COM Server.

5. Call **AfxConnectionUnadvise(…)** before you call **CoUninitialize()** to delete the connection point .

Example:

```
AfxConnectionUnadvise(m_ptrCtiApp, DIID__IOSICTIAppEvents,
m_OSICTIAppEvents->GetInterface (&IID_IUnknown) , TRUE,
m_dwCtiCookie);
```

Note that the Variable m_ dwCtiCookie has to be equal to m_dwCtiCookie from AfxConnectionAdvise() .

8.4. Using Call Information Notification

To receive events for a call you establish a connection point for every new call instance. You can do this when you receive a NewCallEvent of OSITELCOMServer and delete the connection point when points:

1. Declare a new class derived from the **CCmdTarget** . The function **EnableAutomation ()**

has to be called in the constructor of the class object.

2. Declare and define message and interface maps for this class(see OSICTICallEvent.cpp and OSICTICallEvent.h files):

```
BEGIN_DISPATCH_MAP(OSICTICallEvent, CCmdTarget)
//{{AFX_DISPATCH_MAP(OSICTICallEvent)
DISP_FUNCTION_ID(OSICTICallEvent, "StateEvent",
0x13, StateEvent, VT_BOOL, VTS_UNKNOWN VTS_I2)
.
.
.
DISP_FUNCTION_ID(OSICTICallEvent, "DateTimeEvent", 0xf,
DateTimeEvent,
VT_BOOL, VTS_UNKNOWN VTS_DATE)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
BEGIN_INTERFACE_MAP(OSICTICallEvent, CCmdTarget)
INTERFACE_PART(OSICTICallEvent, DIID__IOSICTICallEvents,
Dispatch)
END_INTERFACE_MAP()
```

3. Call the function **AfxConnectionAdvise(…)** when you receive a NewCallEvent .

Example:

```
DWORD dwCtiCallCookie;
OSICTICallEvent *pCallEvent = new OSICTICallEvent(pCall,this);
AfxConnectionAdvise(pCall, DIID__IOSICTICallEvents,
pCallEvent ->GetInterface(&IID_IUnknown), TRUE, &dwCtiCallCookie);
```

4. Declare functions for handling of events:

```
HRESULT OSICTICallEvent::ConnectedAddressEvent(IOSICTICall *piCall,
_bstr_t bstrConnectedAddress)
{
//Your code for handling by this event
}
.
.
.
HRESULT OSICTICallEvent::CalledAddressEvent(IOSICTICall *piCall,
_bstr_t bstrCalledAddress)
{
//Your code for handling by this event
}
```

The reported events are equivalent to the call information retrieving functions.

4. Call **AfxConnectionUnadvise(…)** when you receive RemoveCallEvent:

```
AfxConnectionUnadvise(pCall, DIID__IOSICTICallEvents,
pCallEvent->GetInterface (&IID_IUnknown) ,TRUE,dwCtiCallCookie);
```

Now you can use events of the calls.

8.5. Retrieving Information of a Call

To retrieve information of a call object you use the methods in the IOSICTICall interface. There are several information, such as call state or addresses of the call related parties.

8.6. Making a Call

It is a simple task to establish an outgoing call. There is a method available to establish a new outgoing call with the CreateCall-method of the COM-server interface.

```
void CCTICOMDemoAppDlg::OnDial()
{
// Dial the number in Number EditBox (may be empty)
    bstr_t bstr;
```

```
    bstr = m_csNumber;
    IOSICTICallPtr Call;
    Call = m_ptrCtiApp->CreateCall();
    Call ->PutDialAddress(bstr);
    Call->Dial();
    SetDlgItemText(IDC_NUMBER,"");
}
```

8.7. Drop a Call

The Drop method is used to release a call at any time. The call will be released by the server and you receive the signal CallRemoveEvent.

For example:

```
void CCTICOMDemoAppDlg::OnDrop()
{
    //Drop the Call
    LPARAM lCall ;
    IOSICTICallPtr Call;
    IOSICTICall *pCall;
    int i ;
    i= m_TabCtrl.GetCurSel();
    TCITEM tcItem;
    tcItem.mask = TCIF_PARAM;
    if (i >= 0)
    {
        m_TabCtrl.GetItem(i,&tcItem);
        lCall = tcItem.lParam;
        CConnInfo *pcCInf;
        pcCInf = (CConnInfo *)lCall;
        pCall = pcCInf->m_piCall;
        pCall->QueryInterface(IID_IOSICTICall,(void**)&Call);
        Call->Drop();
    }
}
```

8.8. Code Listing for Visual C++

The sample consists of the following files:

- StdAfx.h , StdAfx.cpp - Standard-System-Include files

- ConnInfo.h ConnInfo.cpp - contain a class which can be used for saving information about connection between application and new call object

- CTICOMDemoApp.h CTICOMDemoApp.cpp main file of the application

- CTICOMDemoAppDlg.h CTICOMDemoAppDlg.cpp contain a class of main dialog

- OSICTIAppEvents.h OSICTIAppEvents.cpp contain a class which can be used for receiving events per application .

- OSICTICallEvent.h OSICTICallEvent.cpp contain a class which can be used for receiving events per call.

In addition to the fundamental principles of programming the OSITELCOMServer, the sample handles an own call list in a TabStrip-control.

The complete project can be found on the CD of OSITRON CTI 3.0.

There are excerpts of source code that you need for COM integration :

StdAfx.h

```
…
#import"OSITELCOMServer.tlb" no_namespace named_guids
…
```

ConnInfo.h

```
#if
!defined(AFX_CONNINFO_H__F333C9BD_CF6C_4E29_A87D_F609DECB5968__INCL
UDED_)
#define
AFX_CONNINFO_H__F333C9BD_CF6C_4E29_A87D_F609DECB5968__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class OSICTICallEvent;
class CConnInfo
{
public:
OSICTICallEvent *m_pCallEvent;
IOSICTICall * m_piCall;
DWORD m_dwCoockie;
CConnInfo(DWORD,IOSICTICall *,OSICTICallEvent *);
CConnInfo();
virtual ~CConnInfo();
};
#endif //
!defined(AFX_CONNINFO_H__F333C9BD_CF6C_4E29_A87D_F609DECB5968__INCL
UDED_)
```

ConnInfo.cpp

```
//////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "stdafx.h"
#include "ConnInfo.h"
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
//////////////////////////////////////////////////////////////////////
///
CConnInfo::CConnInfo(DWORD dwCoockie,IOSICTICall
*piCall,OSICTICallEvent *pCallEvent)
{
    m_dwCoockie = dwCoockie;
    m_piCall = piCall;
    m_pCallEvent = pCallEvent;
}
CConnInfo::CConnInfo()
{}
CConnInfo::~CConnInfo()
{}
```

CTICOMDemoApp.cpp

```
...
BOOL CCTICOMDemoAppApp::InitInstance()
{
    AfxEnableControlContainer();
    #ifdef _AFXDLL
    Enable3dControls();
    #else
    Enable3dControlsStatic();
    #endif
    //Initialize of COM-Server
    CoInitialize(NULL);
    CCTICOMDemoAppDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
    }
    else if (nResponse == IDCANCEL)
    {
    }
    return FALSE;
```

```
}
int CCTICOMDemoAppApp::ExitInstance()
{
    //Uninitialize The COM Server
    CoUninitialize();
    TRACE("CoUninitialize()\n");
    return CWinApp::ExitInstance();
}
…
```

CTICOMDemoAppDlg.cpp

```
...
CCTICOMDemoAppDlg::~CCTICOMDemoAppDlg()
{
    // Disconnect CTI COM Server
    if (m_dwCtiCookie != 0)
    {
        BOOL res = AfxConnectionUnadvise(m_ptrCtiApp,
        DIID__IOSICTIAppEvents,
        m_OSICTIAppEvents->GetInterface (&IID_IUnknown) ,
        TRUE, m_dwCtiCookie);
    }
    delete m_OSICTIAppEvents;
}
void CCTICOMDemoAppDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCTICOMDemoAppDlg)
    DDX_Control(pDX, IDC_TAB1, m_TabCtrl);
    DDX_Text(pDX, IDC_EVENTWND, m_csEvents);
    DDX_Text(pDX, IDC_NUMBER, m_csNumber);
    DDX_Text(pDX, IDC_STATEINFO, m_csStateinfo);
    DDX_Text(pDX, IDC_DIRECTION, m_csDirection);
    //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CCTICOMDemoAppDlg, CDialog)
//{{AFX_MSG_MAP(CCTICOMDemoAppDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_DIAL, OnDial)
ON_BN_CLICKED(IDC_DROP, OnDrop)
ON_BN_CLICKED(IDC_ANSWER, OnAnswer)
ON_BN_CLICKED(IDC_SNDSMS, OnSndSms)
ON_BN_CLICKED(IDC_SELLINES, OnSelLines)
ON_BN_CLICKED(IDC_MKCONFIG, OnMakeConfig)
ON_BN_CLICKED(IDC_CONFHKEYS, OnConfHKeys)
ON_BN_CLICKED(IDC_CONFADDRBOOK, OnConfAddrBook)
ON_BN_CLICKED(IDC_SELLANG, OnSelLang)
ON_BN_CLICKED(IDC_CONFCALLDEFMGR, OnConfCallDefMgr)
ON_BN_CLICKED(IDC_CONFCALLFWDMGR, OnConfCallFwdMgr)
ON_BN_CLICKED(IDC_ABOUTCTI, OnAboutcti)
ON_EN_CHANGE(IDC_NUMBER, OnChangeNumber)
ON_BN_CLICKED(IDC_CLEAR, OnClear)
ON_NOTIFY(TCN_SELCHANGE, IDC_TAB1, OnSelchangeTab)
ON_WM_CLOSE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////
//////////
// CCTICOMDemoAppDlg Nachrichten-Handler
BOOL CCTICOMDemoAppDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
```

```
            strAboutMenu.LoadString(IDS_ABOUTBOX);
            if (!strAboutMenu.IsEmpty())
            {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                strAboutMenu);
            }
        }
        SetIcon(m_hIcon, TRUE);
        SetIcon(m_hIcon, FALSE);
        //Create an Instance of OSICTICOMServer as member variable of
        dialog
        long lres = m_ptrCtiApp.CreateInstance(__uuidof(OSICTIApp));
        //Connection Point
        m_OSICTIAppEvents = new OSICTIAppEvents(m_ptrCtiApp,this);
        BOOL res = AfxConnectionAdvise(m_ptrCtiApp,
        DIID__IOSICTIAppEvents,
        m_OSICTIAppEvents ->GetInterface(&IID_IUnknown), TRUE,
        &m_dwCtiCookie);
        return TRUE;
    }
    void CCTICOMDemoAppDlg::OnSysCommand(UINT nID, LPARAM lParam)
    {
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
    }
    void CCTICOMDemoAppDlg::OnPaint()
    {
        if (IsIconic())
        {
            CPaintDC dc(this);
            SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
            0);
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;
            dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
            CDialog::OnPaint();
        }
    }
    HCURSOR CCTICOMDemoAppDlg::OnQueryDragIcon()
    {
        return (HCURSOR) m_hIcon;
    }
    void CCTICOMDemoAppDlg::OnDial()
    {
        // Dial the number in Number EditBox (may be empty)
        _bstr_t bstr;
        bstr = m_csNumber;
        IOSICTICallPtr Call;
        Call = m_ptrCtiApp->CreateCall();
        Call ->PutDialAddress(bstr);
        Call->Dial();
        SetDlgItemText(IDC_NUMBER,"");
    }
    void CCTICOMDemoAppDlg::OnDrop()
    {
```

```
    //Drop the Call
    LPARAM lCall ;
    IOSICTICallPtr Call;
    IOSICTICall *pCall;
    int i ;
    i= m_TabCtrl.GetCurSel();
    TCITEM tcItem;
    tcItem.mask = TCIF_PARAM;
    if (i >= 0)
    {
        m_TabCtrl.GetItem(i,&tcItem);
        lCall = tcItem.lParam;
        CConnInfo *pcCInf;
        pcCInf = (CConnInfo *)lCall;
        pCall = pcCInf->m_piCall;
        pCall->QueryInterface(IID_IOSICTICall,(void**)&Call);
        Call->Drop();
    }
}
void CCTICOMDemoAppDlg::OnAnswer()
{
    // Answer the incoming Call
    LPARAM lCall ;
    IOSICTICallPtr Call;
    IOSICTICall *pCall;
    int i ;
    i= m_TabCtrl.GetCurSel();
    TCITEM tcItem;
    tcItem.mask = TCIF_PARAM;
    if (i>=0)
    {
        m_TabCtrl.GetItem(i,&tcItem);
        lCall = tcItem.lParam;
        CConnInfo *pcCInf;
        pcCInf= (CConnInfo *)lCall;
        pCall = pcCInf->m_piCall;
        pCall->QueryInterface(IID_IOSICTICall,(void**)&Call);
        if (Call->Direction == OSICTICallDirectionIncoming)
            Call->Answer();
    }
}
void CCTICOMDemoAppDlg::OnSndSms()
{
    // Call the "Send SMS" Dialog
    _bstr_t bstr;
    bstr = m_csNumber;
    m_ptrCtiApp->PopupSMSDialog("",bstr,"","","","");
    SetDlgItemText(IDC_NUMBER,"");
}
void CCTICOMDemoAppDlg::OnSelLines()
{
    // Call the "Select Lines" Dialog
    m_ptrCtiApp->SelectLines();
}
void CCTICOMDemoAppDlg::OnMakeConfig()
{
    // Call the "Configuratin of telephone options" Dialog
    m_ptrCtiApp->MakeConfiguration();
}
void CCTICOMDemoAppDlg::OnConfHKeys()
{
    // Call the "Set Key Combinations" Dialog
    m_ptrCtiApp->ConfigureHotKeys();
}
void CCTICOMDemoAppDlg::OnConfAddrBook()
{
    // Call the Dialog for Address Book configuration
    m_ptrCtiApp->ConfigureAddressBooks();
}
void CCTICOMDemoAppDlg::OnSelLang()
```

```
{
    // Call the "Select Language" Dialog
    m_ptrCtiApp->SelectLanguage();
}
void CCTICOMDemoAppDlg::OnConfCallDefMgr()
{
    // Call the "CallManager Configuration" Dialog
    m_ptrCtiApp->ConfigureCallDeflectManager();
}
void CCTICOMDemoAppDlg::OnConfCallFwdMgr()
{
    //Call the Dialog for Configuration of call Division on aktive
    //line
    m_ptrCtiApp->ConfigureCallForwardManager();
}
void CCTICOMDemoAppDlg::OnAboutcti()
{
    //Call the "About Cti" Dialog
    m_ptrCtiApp->About();
}
void CCTICOMDemoAppDlg::AddTab(CConnInfo *CInfo)
{
    //Add Control Tab with Call Info to Dialog
    TCITEM tcItem;
    tcItem.mask = TCIF_PARAM;
    int i = m_TabCtrl.GetItemCount();
    LPARAM lp;
    lp = (LPARAM)CInfo;
    tcItem.lParam = lp ;
    m_TabCtrl.InsertItem(i,& tcItem);
    m_TabCtrl.SetCurSel(i);
}
void CCTICOMDemoAppDlg::DelTab(int TabNr)
{
    // Delete Tab by end of call
    m_TabCtrl.DeleteItem(TabNr);
    LPARAM lp;
    int TabCount = m_TabCtrl.GetItemCount();
    if (TabCount == 0)
    {
        SetDlgItemText(IDC_CALLEDADR,"");
        SetDlgItemText(IDC_CALLINGADR,"");
        SetDlgItemText(IDC_CONNECTEDADR,"");
        SetDlgItemText(IDC_DIRECTION,"");
        SetDlgItemText(IDC_STATEINFO,"");
    }
    else
    {
        IOSICTICall *pCall;
        m_TabCtrl.SetCurSel(0);
        TCITEM tcItem;
        tcItem.mask = TCIF_PARAM;
        m_TabCtrl.GetItem(0,&tcItem);
        lp = tcItem.lParam;
        CConnInfo *pcCInf = (CConnInfo *)lp;
        pCall = pcCInf->m_piCall;
        UpdateWnd(pCall);
    }
    Invalidate();
}
void CCTICOMDemoAppDlg::OnChangeNumber()
{
    GetDlgItemText(IDC_NUMBER,m_csNumber);
}
void CCTICOMDemoAppDlg::OnClear()
{
    // Clear the Events Window
    SetDlgItemText(IDC_EVENTWND,(m_csEvents = ""));
}
HRESULT CCTICOMDemoAppDlg::NewCallEvent(IOSICTIApp *piApp,
```

```
IOSICTICall *piCall)
{
    // Handle for NewCallEvent
    IOSICTICall *pCall;
    piCall->QueryInterface(IID_IOSICTICall,(void**)&pCall);
    //Connection Point
    DWORD dwCtiCallCookie;
    OSICTICallEvent *pCallEvent = new OSICTICallEvent(piCall,this);
    BOOL res = AfxConnectionAdvise(piCall, DIID__IOSICTICallEvents,
    pCallEvent ->GetInterface(&IID_IUnknown),TRUE,
    &dwCtiCallCookie);
    CConnInfo *pConnInfo = new
    CConnInfo(dwCtiCallCookie,pCall,pCallEvent);
    AddTab(pConnInfo);
    Invalidate();
    return true;
}
HRESULT CCTICOMDemoAppDlg::CallRemoveEvent(IOSICTIApp *piApp,
IOSICTICall *piCall)
{
    //Handle for CallRemoveEvent
    IOSICTICall *pCall;
    piCall->QueryInterface(IID_IOSICTICall,(void**)&pCall);
    LPARAM lp;
    OSICTICallEvent *pCallEvent;
    DWORD dwCtiCallCookie;
    TCITEM tcItem;
    tcItem.mask = TCIF_PARAM;
    int i = GetTab(pCall);
    m_TabCtrl.GetItem(i,&tcItem);
    lp = tcItem.lParam;
    CConnInfo *pcCInf;
    pcCInf = (CConnInfo *)lp;
    dwCtiCallCookie = pcCInf->m_dwCoockie;
    pCallEvent = pcCInf->m_pCallEvent;
    DelTab(i);
    // Disconnect Call Object
    if (dwCtiCallCookie != 0)
    {
        BOOL res = AfxConnectionUnadvise(piCall,
        DIID__IOSICTICallEvents,
        pCallEvent->GetInterface (&IID_IUnknown) ,TRUE,
        dwCtiCallCookie);
    }
    delete pcCInf;
    delete pCallEvent;
    return true;
}
void CCTICOMDemoAppDlg::UpdateWnd(IOSICTICall *piCall)
{
    // Update Window by new Call or Application events
    IOSICTICallPtr Call;
    piCall -> QueryInterface(IID_IOSICTICall,(void**)&Call);
    int iState = Call->State;
    switch (iState)
    {
    case 0:
           m_csStateinfo = "Idle";
           break;
    case 1:
           m_csStateinfo = "Ringing";
           break;
    case 2:
           m_csStateinfo = "Dialing";
           break;
    case 3:
           m_csStateinfo = "Proceeding";
           break;
    case 4:
           m_csStateinfo = "Ringback";
```

```
            break;
        case 5:
            m_csStateinfo = "Connected";
            break;
        case 6:
            m_csStateinfo = "Disconnected";
            break;
        default:
            m_csStateinfo = "Idle";
            break;
        }
        int iDirect = Call->Direction;
        switch (iDirect)
        {
        case 1:
            m_csDirection = "Incoming";
            break;
        case 2:
            m_csDirection = "Outgoing";
            break;
        default:
            m_csDirection = "";
            break;
        }
        SetDlgItemText(IDC_CALLEDADR,Call->GetCalledAddress());
        SetDlgItemText(IDC_CALLINGADR,Call->GetCallingAddress());
        SetDlgItemText(IDC_CONNECTEDADR,Call->GetConnectedAddress());
        SetDlgItemText(IDC_DIRECTION,m_csDirection);
        SetDlgItemText(IDC_STATEINFO,m_csStateinfo);
        int iTabNr;
        IOSICTICall * pCall;
        piCall->QueryInterface(IID_IOSICTICall,(void**)&pCall);
        iTabNr = GetTab(pCall);
        m_TabCtrl.SetCurSel(iTabNr);
    }
    void CCTICOMDemoAppDlg::OnSelchangeTab(NMHDR* pNMHDR, LRESULT*
    pResult)
    {
        // Handle for the Tab selection
        TCITEM tcItem;
        tcItem.mask = TCIF_PARAM;
        LPARAM lp;
        CConnInfo *pCInfo;
        int i;
        IOSICTICall *pCall;
        i= m_TabCtrl.GetCurSel();
        m_TabCtrl.GetItem(i,&tcItem);
        lp = tcItem.lParam;
        pCInfo = (CConnInfo *)lp;
        pCall = pCInfo->m_piCall;
        UpdateWnd(pCall);
        *pResult = 0;
    }
    int CCTICOMDemoAppDlg::GetTab(IOSICTICall *piCall)
    {
        LPARAM lp;
        TCITEM tcItem;
        tcItem.mask = TCIF_PARAM;
        IOSICTICall *pCall;
        CConnInfo *pCInfo;
        int i = 0;
        do
        {
            m_TabCtrl.GetItem(i,&tcItem);
            lp = tcItem.lParam;
            pCInfo = (CConnInfo *)lp;
            i++;
            pCall = pCInfo->m_piCall;
        }while (piCall != pCall);
        return i-1;
```

```
}
void CCTICOMDemoAppDlg::UpdateEventWnd(CString csEvent)
{
    m_csEvents= m_csEvents + csEvent + " ---------------------------
    ----------- ";
    SetDlgItemText(IDC_EVENTWND,m_csEvents);
}
void CCTICOMDemoAppDlg::OnClose()
{
    //Handle for Close and Cancel
    int i = m_TabCtrl.GetItemCount();
    if ( i == 0)
        CDialog::OnCancel();
    else
        MessageBox(" Drop first all calls ,Please ! ");
    }
void CCTICOMDemoAppDlg::OnCancel()
{
    OnClose();
}
…
```

OSICTIAppEvents.cpp

```
// OSICTIAppEvents.cpp
//
#include "stdafx.h"
#include "CTICOMDemoApp.h"
#include "OSICTIAppEvents.h"
#include "CTICOMDemoAppDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////
/////////
// OSICTIAppEvents
//IMPLEMENT_DYNCREATE(OSICTIAppEvents, CCmdTarget)
OSICTIAppEvents::OSICTIAppEvents()
{
}
OSICTIAppEvents::~OSICTIAppEvents()
{
}
OSICTIAppEvents::OSICTIAppEvents(IOSICTIApp
*pOsiCtiApp,CCTICOMDemoAppDlg *DLG)
{
    Dlg = DLG;
    m_CtiAppEvents = pOsiCtiApp;
    EnableAutomation ();
}
BEGIN_MESSAGE_MAP(OSICTIAppEvents, CCmdTarget)
//{{AFX_MSG_MAP(OSICTIAppEvents)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////
/////////
BEGIN_DISPATCH_MAP(OSICTIAppEvents, CCmdTarget)
//{{AFX_DISPATCH_MAP(OSICTIAppEvents)
DISP_FUNCTION_ID(OSICTIAppEvents, "NewCallEvent",
1,NewCallEvent, VT_BOOL, VTS_UNKNOWN VTS_UNKNOWN)
DISP_FUNCTION_ID(OSICTIAppEvents, "CallRemoveEvent", 2,
CallRemoveEvent, VT_BOOL, VTS_UNKNOWN VTS_UNKNOWN)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
BEGIN_INTERFACE_MAP(OSICTIAppEvents, CCmdTarget)
INTERFACE_PART(OSICTIAppEvents, DIID__IOSICTIAppEvents,
Dispatch)
END_INTERFACE_MAP()
HRESULT OSICTIAppEvents::NewCallEvent(IOSICTIApp *piApp,
```

```
IOSICTICall *piCall)
{
    Dlg->NewCallEvent(piApp,piCall);
    TRACE("OSICTIAppEvents::NewCallEvent\n");
    return true;
}
HRESULT OSICTIAppEvents::CallRemoveEvent(IOSICTIApp *piApp,
IOSICTICall *piCall)
{
    Dlg->CallRemoveEvent(piApp,piCall);
    TRACE ("OSICTIAppEvents::CallRemoveEvent\n");
    return true;
}
```

OSICTIAppEvents.h

```
#if
!defined(AFX_OSICTIAPPEVENTS_H__C0EED527_2C67_4E8C_8570_558229128E6
F__INCLUDED_)
#define
AFX_OSICTIAPPEVENTS_H__C0EED527_2C67_4E8C_8570_558229128E6F__INCLUD
ED_
#include "CTICOMDemoAppDlg.h"
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// OSICTIAppEvents.h
//
class CCTICOMDemoAppDlg;
/////////////////////////////////////////////////////////////////////
/////////
class OSICTIAppEvents : public CCmdTarget
{
    // DECLARE_DYNCREATE(OSICTIAppEvents)
    OSICTIAppEvents();
    public:
    OSICTIAppEvents(IOSICTIApp *,CCTICOMDemoAppDlg *);
    CCTICOMDemoAppDlg *Dlg;
    IOSICTIApp *m_CtiAppEvents;
    HRESULT NewCallEvent ( struct IOSICTIApp * piApp, struct
    IOSICTICall * piCall );
    HRESULT CallRemoveEvent ( struct IOSICTIApp * piApp, struct
    IOSICTICall * piCall );
    //{{AFX_VIRTUAL(OSICTIAppEvents)
    //}}AFX_VIRTUAL
    virtual ~OSICTIAppEvents();
    protected:
    //{{AFX_MSG(OSICTIAppEvents)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
    DECLARE_DISPATCH_MAP ()
    DECLARE_INTERFACE_MAP ()
};
/////////////////////////////////////////////////////////////////////
/////////
//{{AFX_INSERT_LOCATION}}
#endif //
AFX_OSICTIAPPEVENTS_H__C0EED527_2C67_4E8C_8570_558229128E6F__INCLUD
ED_
```

OSICTICallEvent.h

```
#if
!defined(AFX_OSICTICALLEVENT_H__E6E6B202_DDD2_4E86_BCD1_B6925E245A5
3__INCLUDED_)
#define
AFX_OSICTICALLEVENT_H__E6E6B202_DDD2_4E86_BCD1_B6925E245A53__INCLUD
ED_
#include "CTICOMDemoAppDlg.h"
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

```
// OSICTICallEvent.h
class CCTICOMDemoAppDlg;
/////////////////////////////////////////////////////////////////
/////////
class OSICTICallEvent : public CCmdTarget
{
DECLARE_DYNCREATE(OSICTICallEvent)
OSICTICallEvent();
public:
OSICTICallEvent(IOSICTICall *,CCTICOMDemoAppDlg *);
public:
HRESULT CanonicalAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrCanonicalAddress );
HRESULT DisplayEvent ( struct IOSICTICall * piCall, _bstr_t
bstrDisplay );
HRESULT NameEvent ( struct IOSICTICall * piCall, _bstr_t
bstrName );
HRESULT ChargingEvent ( struct IOSICTICall * piCall, const
CURRENCY & cyCharging );
HRESULT RedirectionAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrRedirectionAddress );
HRESULT RedirectingAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrRedirectingAddress );
HRESULT StateEvent ( struct IOSICTICall * piCall, enum
EOSICTICallState eState );
HRESULT DateTimeEvent ( struct IOSICTICall * piCall, DATE
dateDateTime );
HRESULT DirectionEvent ( struct IOSICTICall * piCall, enum
EOSICTICallDirection eDirection );
HRESULT CalledAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrCalledAddress );
HRESULT ConnectedAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrConnectedAddress );
HRESULT CallingAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrCallingAddress );
HRESULT NormalizedAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrNormalizedAddress );
HRESULT DialAddressEvent ( struct IOSICTICall * piCall, _bstr_t
bstrDialAddress );
HRESULT DialableAddressEvent ( struct IOSICTICall * piCall,
_bstr_t bstrDialableAddress );
CCTICOMDemoAppDlg *Dlg;
IOSICTICall *m_CallEvent;
//{{AFX_VIRTUAL(OSICTICallEvent)
//}}AFX_VIRTUAL
virtual ~OSICTICallEvent();
protected:
//{{AFX_MSG(OSICTICallEvent)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
DECLARE_DISPATCH_MAP ()
DECLARE_INTERFACE_MAP ()
};
/////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
#endif //
AFX_OSICTICALLEVENT_H__E6E6B202_DDD2_4E86_BCD1_B6925E245A53__INCLUD
ED_
```

OSICTICallEvent.cpp

```
// OSICTICallEvent.cpp
//
#include "stdafx.h"
#include "CTICOMDemoApp.h"
#include "OSICTICallEvent.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////////////////////////////////////
//////////
// OSICTICallEvent
IMPLEMENT_DYNCREATE(OSICTICallEvent, CCmdTarget)
OSICTICallEvent::OSICTICallEvent(IOSICTICall*pCtiCall,
CCTICOMDemoAppDlg * DLG)
{
    Dlg = DLG;
    m_CallEvent = pCtiCall;
    EnableAutomation ();
    // TRACE("OSICTICallEvent::OSICTICallEvent()\n");
}
OSICTICallEvent::OSICTICallEvent()
{
}
OSICTICallEvent::~OSICTICallEvent()
{
    // TRACE("OSICTICallEvent::~OSICTICallEvent()\n");
}
BEGIN_MESSAGE_MAP(OSICTICallEvent, CCmdTarget)
//{{AFX_MSG_MAP(OSICTICallEvent)
// HINWEIS - Der Klassen-Assistent fügt hier
Zuordnungsmakros ein und entfernt diese.
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////
//////////
BEGIN_DISPATCH_MAP(OSICTICallEvent, CCmdTarget)
//{{AFX_DISPATCH_MAP(OSICTICallEvent)
DISP_FUNCTION_ID(OSICTICallEvent, "StateEvent",
0x13, StateEvent, VT_BOOL, VTS_UNKNOWN VTS_I2)
DISP_FUNCTION_ID(OSICTICallEvent, "DialableAddressEvent",
0x7, DialableAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "DialAddressEvent",
0x4, DialAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "NormalizedAddressEvent",
0x9, NormalizedAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "CallingAddressEvent",
0xa, CallingAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "ConnectedAddressEvent",
0xb, ConnectedAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "CalledAddressEvent",
0x3, CalledAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "DirectionEvent",
0x10, DirectionEvent, VT_BOOL, VTS_UNKNOWN VTS_I2)
DISP_FUNCTION_ID(OSICTICallEvent, "RedirectingAddressEvent",
0xc, RedirectingAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "RedirectionAddressEvent",
0xd, RedirectionAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "ChargingEvent",
0xe, ChargingEvent, VT_BOOL, VTS_UNKNOWN VTS_VARIANT)
DISP_FUNCTION_ID(OSICTICallEvent, "NameEvent",
0x12, NameEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "DisplayEvent",
0x11, DisplayEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "CanonicalAddressEvent",
0x8, CanonicalAddressEvent, VT_BOOL, VTS_UNKNOWN VTS_BSTR)
DISP_FUNCTION_ID(OSICTICallEvent, "DateTimeEvent",
0xf, DateTimeEvent, VT_BOOL, VTS_UNKNOWN VTS_DATE)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
BEGIN_INTERFACE_MAP(OSICTICallEvent, CCmdTarget)
INTERFACE_PART(OSICTICallEvent, DIID__IOSICTICallEvents,
Dispatch)
END_INTERFACE_MAP()
HRESULT OSICTICallEvent::DialableAddressEvent(IOSICTICall *piCall,
_bstr_t bstrDialableAddress)
{
    CString csEvent;
```

```
    csEvent.Format("%-30s: %-30s", "DialableAddressEvent",
    bstrDialableAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("DialableAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::DialAddressEvent(IOSICTICall *piCall,
_bstr_t bstrDialAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","DialAddressEvent",
    bstrDialAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("DialAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::NormalizedAddressEvent(IOSICTICall
*piCall, _bstr_t bstrNormalizedAddress)
{
    CString csEvent;
    csEvent.Format("%-27s: %-30s","NormalizedAddressEvent",
    bstrNormalizedAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("NormalizedAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::CallingAddressEvent(IOSICTICall *piCall,
_bstr_t bstrCallingAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","CallingAddressEvent",
    bstrCallingAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("CallingAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::ConnectedAddressEvent(IOSICTICall *piCall,
_bstr_t bstrConnectedAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","ConnectedAddressEvent",
    bstrConnectedAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("ConnectedAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::CalledAddressEvent(IOSICTICall *piCall,
_bstr_t bstrCalledAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","CalledAddressEvent",
    bstrCalledAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("CalledAddressEvent\n");
    return true;
}
HRESULT OSICTICallEvent::DirectionEvent(IOSICTICall *piCall,
EOSICTICallDirection eDirection)
{
    CString csEvent,csDirection;
    switch (eDirection)
    {
    case OSICTICallDirectionIncoming:
           csDirection = "Incoming";
```

```
            break;
        case OSICTICallDirectionOutgoing:
            csDirection = "Outgoing";
            break;
        default:
            csDirection = "";
        }
        csEvent.Format("%-30s: %-30s ","DirectionEvent",csDirection);
        Dlg->UpdateWnd(piCall);
        Dlg->UpdateEventWnd(csEvent);
        // TRACE("DirectionEvent\n");
        return true;
    }
    HRESULT OSICTICallEvent::DateTimeEvent(IOSICTICall *piCall, DATE
    dateDateTime)
    {
        CString csEvent;
        csEvent.Format("%-30s: %-30s ","DateTimeEvent","" );
        Dlg->UpdateWnd(piCall);
        Dlg->UpdateEventWnd(csEvent);
        // TRACE("DateTimeEvent\n");
        return true;
    }
    HRESULT OSICTICallEvent::StateEvent(IOSICTICall
    *piCall,EOSICTICallState eState)
    {
        CString csEvent,csState;
        switch (eState)
        {
        case OSICTICallStateIdle:
            csState = "Idle";
            break;
        case OSICTICallStateRinging:
            csState = "Ringing";
            break;
        case OSICTICallStateDialing:
            csState = "Dialing";
            break;
        case OSICTICallStateProceeding:
            csState = "Proceeding";
            break;
        case OSICTICallStateRingback:
            csState = "Ringback";
            break;
        case OSICTICallStateConnected:
            csState = "Connected";
            break;
        case OSICTICallStateDisconnected:
            csState = "Disconnected";
            break;
        default:
            csState ="";
        }
        csEvent.Format("%-30s: %-30s ","StateEvent ",csState);
        Dlg->UpdateWnd(piCall);
        Dlg->UpdateEventWnd(csEvent);
        // TRACE("StateEvent %d \n",Dlg->m_pCtiCall->State);
        return true;
    }
    HRESULT OSICTICallEvent::RedirectingAddressEvent(IOSICTICall
    *piCall, _bstr_t bstrRedirectingAddress)
    {
        CString csEvent;
        csEvent.Format("%-30s: %-30s","RedirectingAddressEvent",
        bstrRedirectingAddress);
        Dlg->UpdateWnd(piCall);
        Dlg->UpdateEventWnd(csEvent);
        // TRACE("RedirectingAddressEvent \n");
        return true;
    }
```

```
HRESULT OSICTICallEvent::RedirectionAddressEvent(IOSICTICall
*piCall, _bstr_t bstrRedirectionAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","RedirectionAddressEvent",
    bstrRedirectionAddress);
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("RedirectionAddressEvent \n");
    return true;
}
HRESULT OSICTICallEvent::ChargingEvent(IOSICTICall *piCall, const
CURRENCY &cyCharging)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s ","ChargingEvent"," " );
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("ChargingEvent \n");
    return true;
}
HRESULT OSICTICallEvent::NameEvent(IOSICTICall *piCall, _bstr_t
bstrName)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s ","NameEvent",bstrName );
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("NameEvent \n");
    return true;
}
HRESULT OSICTICallEvent::DisplayEvent(IOSICTICall *piCall, _bstr_t
bstrDisplay)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s ","DisplayEvent",bstrDisplay );
    Dlg->UpdateWnd(piCall);
    Dlg->UpdateEventWnd(csEvent);
    // TRACE("DisplayEvent \n");
    return true;
}
HRESULT OSICTICallEvent::CanonicalAddressEvent(IOSICTICall *piCall,
_bstr_t bstrCanonicalAddress)
{
    CString csEvent;
    csEvent.Format("%-30s: %-30s","  CanonicalAddressEvent",
    bstrCanonicalAddress );
    Dlg->UpdateWnd(piCall);
    // TRACE("CanonicalAddressEvent \n");
    return true;
}
```

9. ActiveX Sample in Visual Studio C++

9.1. Using the ActiveX Control

The ActiveX Sample Project describe the creation of an ActiveX Control to use OSITRON COM Interface. You can test the control with the sample html-Page. One has to enable ActiveX and java script in his browser.

The Ositron ocx activeX control has to be registered in Windows system, typically using the command RegSvr32.exe

Current implementation:

1. Commands:

MakeCall(BSTR bstrPhoneNumber)

➔ makes new call on the required phone number

AnswerCall ()

➔ answers the call, if it is in the ringing state

ClearCall ()

➔ makes hangup, if the call is in the connected state (in or out)

Name	Value (decimal)
READY	0
OUT_OF_SERVICE	1
UNINITIALIZED	2
ACDPREPARE	3
ACDOUTBOUND	4
ACDINBOUND	5
ACDWORKAFTERCALL	6
ACDREADY	7

2: Events:

CallEvent(LONG iIndex, LONG iCallState, BSTR bstrCallData)

➔ is signaled when the call state of the call changed

Parameters

iIndex

index in the call array, on used extension. typically this value is 0:

0 only one call present

1 or higher if the consultation call or queued (knocking) call is present

iCallState

actual state of the call

CallState	Value
IDLE	0
DIALING	1
RINGING	2
RINGBACK	3
INCONNECTED	4
OUTCONNECTED	5
DISCONNECTED	6

bstrCallData

string, containing semicolon separated following tokens:

"CallingAddress;CalledAddress;NormalizedAddress;ConnectedAddress;Display;Name;RedirectingAddress;RedirectionAddress"

CallDetailEvent(LONG iWhatDetail, LONG iIndex, LONG iCallState, BSTR bstrDetailData)

➔ signaled when the single field in the call data is changed or call state is changed

Parameters

iWhatDetail

Value	Description
0	called address changed
1	calling address changed
2	dialable address changed
3	connected address changed
4	dial address changed
5	normalized address changed
6	redirecting address changed
7	redirection address changed
8	name changed
9	display changed
10	canonical address changed
11	charging info changed
12	date time info changed
13	Unused
14	Unused
15	call state changed

iIndex

see CallEvent description

iCallState

see CallEvent description

bstrDetaildData

the corresponding string with actual information, or empty, if only call state changed

The calling application can handle the call event as a whole, with all infos already set, or react for the singlecall detail changes, like for example calling number (if state is ringing) or called number (if state is ringback)

10. Sample in Delphi

10.1. Using the new COM Server

To use the new COM Server you need a correctly installed and registered OSITRON CTI 3.0.

At first you have to import the Type Library of “OSICTIControlCenter 3.0”. You can do this by processing the following steps. In the menu “Project” choose “Import Typelibrary” and then “OSICTIControlCenter 3.0 Typbibliothek(Version 3.0)”. The next step is to install the new classes. Choose “ Palette Page” – “Standard” – “Install” to do this. Now you have 3 new classes “TOSICTICall”, “TTelApp” and “TOSICTIApp”. In the property page choose “Autoconnect=True” and “ConnectKind=ckRunningOrNew”.

Please declare a variable for the COM Call Object.

```
Var Call:OSICTICall;
```

Then you must create a new object of TOSICTIApp. This is done in the example in the Form Tform1.

Now you have a new object of the COM Server. With this object you can call each function that is implemented in the interface.

10.2. Using Notification

To receive events from the COM Server you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

The events reported to this object are the NewCallEvent and the CallRemoveEvent.to indicate that a new call is arrived resp. a call is leaving.

10.3. Retrieving the Call List

To get a list of all established calls at application startup time you can use the CallList-property. The interface contained in the resulting variant is an IOSIStaticCollection interface. The elements returned by the Item method are interface pointers which can be converted to an IOSICTICall interface by QueryInterface.

10.4. Retrieving Information of a Call

To retrieve information you use the methods in the IOSICTICall interface of a call object. There are several information, such as call state or addresses of the call related parties.

10.5. Using Call Information Notification

To receive call information events you must establish a connection point, which can be called by the Server. This will be done automatically, therefor you only have to implement the corresponding Event procedure.

```
procedure TForm1.OSICTICall1CallingAddressEvent(Sender: TObject;
  var piCall, bstrCallingAddress: OleVariant);

begin
end;
```

The reported events are aequivalent to the call information retrieving functions.

10.6. Making a Call

It is a simple task to establish an outgoing call. The method via the COM-server interface is used as in the following:

```
procedure TForm1.DialButtonClick(Sender: TObject);
begin
        (* Wählen *)
        OSICTIApp1.CreateCall;
        Call.Set_DialAddress(DialNumWnd.Text);
        Call.Dial;
        DialButton.Enabled := false;
end;
```

The making of a call via the COM-interface is always done vie the OSITRON CTI application, independent of the setting for assisted telephony.

All calls you made yourself are also reported in the NewCallEvent-interface.

10.7. Drop a Call

With the Drop-function of the associated call-object you can disconnect a call anytime.

If the call is released as a result the CallRemoveEvent is signalled.

10.8. Code Listing for Delphi

The complete project can be found on the cd of OSITRON CTI 3.0.

```
unit cticomdemoapp;

interface

uses
  ActiveX,Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  OleServer, OSICTIControlCenter_TLB, StdCtrls, Db, DBClient,
MConnect,
  SConnect, MPlayer, ComCtrls, Spin, Tabs, ExtCtrls;

type
```

```
  TForm1 = class(TForm)
    OSICTIApp1: TOSICTIApp;
    DialButton: TButton;
    DropButton: TButton;
    SelLinesButton: TButton;
    MakeConfButton: TButton;
    ConfHKeyButton: TButton;
    ConfAdrBookButton: TButton;
    SelLangButton: TButton;
    ConfCallDeflMgrButton: TButton;
    ConfCallForwMgrButton: TButton;
    SendSMSButton: TButton;
    AnswerButton: TButton;
    StateWind: TGroupBox;
    StateWnd: TMemo;
    EndAppButton: TButton;
    OSICTICall1: TOSICTICall;
    ClearButton: TButton;
    NumWind: TGroupBox;
    DialNumWnd: TEdit;
    AboutButton: TButton;
    Edit8: TEdit;
    Label1: TLabel;
    procedure DialButtonClick(Sender: TObject);
    procedure DropButtonClick(Sender: TObject);
    procedure SelLinesButtonClick(Sender: TObject);
    procedure MakeConfButtonClick(Sender: TObject);
    procedure ConfHKeyButtonClick(Sender: TObject);
    procedure ConfAdrBookButtonClick(Sender: TObject);
    procedure SelLangButtonClick(Sender: TObject);
    procedure ConfCallDeflMgrButtonClick(Sender: TObject);
    procedure ConfCallForwMgrButtonClick(Sender: TObject);
    procedure SendSMSButtonClick(Sender: TObject);
    procedure AnswerButtonClick(Sender: TObject);
    procedure OSICTIApp1NewCallEvent(Sender: TObject; var piApp,
      piCall: OleVariant);
    procedure OSICTIApp1CallRemoveEvent(Sender: TObject; var piApp,
      piCall: OleVariant);
    procedure EndAppButtonClick(Sender: TObject);
    procedure ClearButtonClick(Sender: TObject);
    procedure OSICTICall1CallingAddressEvent(Sender: TObject; var
piCall,
      bstrCallingAddress: OleVariant);
    procedure OSICTICall1StateEvent(Sender: TObject;
      var piCall: OleVariant; eState: TOleEnum);
    procedure OSICTICall1CalledAddressEvent(Sender: TObject; var
piCall,
      bstrCalledAddress: OleVariant);
    procedure AboutButtonClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  Call :OSICTICall;

  List: TList;
  Count:INTEGER;
implementation



{$R *.DFM}

procedure TForm1.DialButtonClick(Sender: TObject);
```

```
begin
        (* Wählen *)
        OSICTIApp1.CreateCall;
        Call.Set_DialAddress(DialNumWnd.Text);
        Call.Dial;
        DialButton.Enabled := false;
end;

procedure TForm1.DropButtonClick(Sender: TObject);

begin
        (*Anruf beenden*)
        Call.Drop;
        DropButton.Enabled := false;
end;

procedure TForm1.SelLinesButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Leitungen Wählen"*)
        OSICTIApp1.SelectLines;
end;

procedure TForm1.MakeConfButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Konfiguration *)
        OSICTIApp1.MakeConfiguration;
end;

procedure TForm1.ConfHKeyButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges  "HotKeys Konfigurieren"*)
        OSICTIApp1.ConfigureHotKeys;
end;

procedure TForm1.ConfAdrBookButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Adressbuch Konfigurieren"*)
        OSICTIApp1.ConfigureAddressBooks;
end;

procedure TForm1.SelLangButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Sprache Auswählen" *)
        OSICTIApp1.SelectLanguage;
end;

procedure TForm1.ConfCallDeflMgrButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Anrufmanager Konfigurieren"*)
        OSICTIApp1.ConfigureCallDeflectManager;
end;

procedure TForm1.ConfCallForwMgrButtonClick(Sender: TObject);
begin
        (*Aufruf des Dialoges "Rufumleitung Konfiguriren"*)
        OSICTIApp1.ConfigureCallForwardManager;
end;

procedure TForm1.SendSMSButtonClick(Sender: TObject);
begin
        (*Aufruf des popup Dialoges "SMS Senden" *)
        OSICTIApp1.PopupSMSDialog('',DialNumWnd.text,'','','','');
        DialNumWnd.text := '';
end;

procedure TForm1.AnswerButtonClick(Sender: TObject);
begin
        (*Ruf Annehmen*)
        Call.Answer;
        DropButton.Enabled := true ;
```

```
        AnswerButton.Enabled := false ;
end;

procedure TForm1.OSICTIApp1NewCallEvent(Sender: TObject; var piApp,
  piCall: OleVariant);
begin
        if list = nil then list := TList.Create;
        list.Add(@piCall);
        OSICTICall1.ConnectTo(IUNKNOWN (piCall) as IOSICTICall);
        if List.Count = 1 then Call := (IUNKNOWN (piCall) as
IOSICTICall);
        Edit8.Text := IntToStr(list.Count);
end;

procedure TForm1.OSICTIApp1CallRemoveEvent(Sender: TObject; var
piApp,
  piCall: OleVariant);
begin
        StateWnd.text:=  'Idle' + #13 + #10 + #13 + #10+
StateWnd.text + #13 +#10;
        form1.StateWnd.Update;
        AnswerButton.Enabled :=false;
        DropButton.Enabled := false;
        DialButton.Enabled := true;

        List.Delete(List.IndexOf(@piCall));
        Edit8.Text := IntToStr(list.Count);
        OSICTICall1.Disconnect;
end;
procedure TForm1.EndAppButtonClick(Sender: TObject);
begin
        (*Programm Beenden*)
        form1.Close;
end;

procedure TForm1.ClearButtonClick(Sender: TObject);
begin
        StateWnd.Text := '';
end;

procedure TForm1.OSICTICall1CallingAddressEvent(Sender: TObject;
  var piCall, bstrCallingAddress: OleVariant);

begin
        AnswerButton.Enabled := true;
        DropButton.Enabled := false;
        StateWnd.text:= 'Anruf von                    : ' +
Call.CallingAddress + #13 + #10
                    +'Nornalisierte Nummer : ' +
Call.NormalizedAddress + #13 + #10 + #13 +#10
                    + StateWnd.text ;
        AnswerButton.Enabled := true ;
        form1.StateWnd.Update;
        DialButton.Enabled := false;
end;

procedure TForm1.OSICTICall1StateEvent(Sender: TObject;
  var piCall: OleVariant; eState: TOleEnum);
begin
  If call.State=5 then
        begin
                DropButton.Enabled := true;
                AnswerButton.Enabled := false;
        end;
  If call.State=2 then
        begin
                DialButton.Enabled := false;
                DropButton.Enabled := true;
        end;
  If call.State=0 then
```

```
        begin
                DialButton.Enabled := true;
        end;
end;

procedure TForm1.OSICTICall1CalledAddressEvent(Sender: TObject; var
piCall,
  bstrCalledAddress: OleVariant);
begin
    if Call.State = 2 then
    begin
        StateWnd.Text:= 'Wahlvogang' + #13 +#10  + #13 +#10 +
StateWnd.Text;
        StateWnd.Text:=    'Wählende Nummer     :' +
Call.DialAddress + #13 + #10
                      + 'Gewählte Nummer      :' +
Call.DialableAddress + #13+ #10
                      + 'Normalisierte Nummer :' +
Call.NormalizedAddress + #13 + #10
                      + #13 + #10 + StateWnd.Text ;

        Form1.StateWnd.Update;
        DropButton.Enabled := true;
        DialNumWnd.Text := '';
    end;

end;

procedure TForm1.AboutButtonClick(Sender: TObject);
begin
    OSICTIApp1.About;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
   if call <> nil then call.Drop;
   If List <> nil then list.Destroy;
   OSICTICall1.Destroy;
end;

end.
```

11. Sample in Visual Basic .Net

11.1. Differences between VB 6.0 and VB.Net

There are several differences between Projects created in VB 6.0 and VB.Net. It must be pointed out that VB 6.0 compilers produce unmanaged whereas VB.Net compilers produce managed Code (Intermediate Language or IL). This Managed code runs in the Common Language Runtime. Because of the Just In Time, or JIT compiling of the IL event handling has to run asynchronous. This can be realised by calling the handling procedure in the event-handler through an delegate with the same parameters as used by the Event-handler. The procedure can then later be invoked by means of the delegate instance:

```
Dim <TempVarName> As <NameOfDelegate> = New <NameOfDelegate>
(AddressOf <NameOfHandlingSub>)
```

Invoking the Procedure has to be done in the thread of the Form which created the controls, therefore you must have a reference to the Instance of the Form in every class that accesses the Forms controls.

```
Me.BeginInvoke(<TempVarName>, New Object()
{<1.ParameterOfDelegate>,<2.ParameterOfDelegate>,…, <n.…>})
```

Or when the Calling-Class is another then the Form-Class

```
<ReferenceToFormInstance>.BeginInvoke(<TempVarName>, New
Object(){<1.ParameterOfDelegate>,<2.ParameterOfDelegate>,…, <n.…>})
```

11.2. Using the COM Server

At first you declare a variable for the COM Object.

```
Dim OSICTIClient As OSICTIControlCenter.OSICTIApp
```

Then you must create a new object of OSICTIApp at start-up.

In the example it is done in the “Load” subroutine of the form.

```
Set OSICTIClient = New OSICTIApp
```

Now you have a new object of the COM Server. With this object you can call each function that is implemented in the interface.

11.3. Using Notification

To receive events from the COM Server you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

The events reported to this object are the NewCallEvent and the CallRemoveEvent.to indicate that a new call is arrived resp. a call is leaving.

11.4. Retrieving the Call List

To get a list of all established calls at application startup time you can use the CallList-property. The interface contained in the resulting variant is an IOSIStaticCollection interface. The elements returned by the Item method are interface pointers which can converted to an IOSICTICall interface by QueryInterface.

11.5. Retrieving Information of a Call

To retrieve information you use the methods in the IOSICTICall interface of a call object. There are several information, such as call state or addresses of the call related parties.

11.6. Using Call Information Notification

To receive call information events you must establish a connection point, which can be called by the Server. This is done by assigning the “WithEvents” at the variable declaration for the COM Server.

```
Public WithEvents CTICallClient As OSICTIControlCenter.OSICTICall
```

You can then choose the Eventhandler in the Visual Basic Wizard to implement your own handling.

The reported events are equivalent to the call information retrieving functions.

11.7. Making a Call

It is a simple task to establish an outgoing call. The CreateCall-method of the COM-server interface establishes a Call.

To make a Call via the COM-server interface is shown as in the following:

```
Dim NewCall As OSICTIControlCenter.IOSICTICall

NewCall = OSICTIClient.CreateCall
NewCall.DialAddress = "11"
NewCall.Dial
```

The making of a call via the COM-interface is always done vie the OSITRON CTI application, independent of the setting for assisted telephony.

All calls you made yourself are also reported in the NewCallEvent-interface.

11.8. Drop a Call

With the Drop-function of the associated call-object you can disconnect a call anytime.

If the call is released as a result the CallRemoveEvent is signalled.

11.9. Code Listing for Visual Basic .Net

The sample consists of the following 3 files:

- OSICTIClientSample.vbproj is the Project file
- OSICTIClientSample.vb contains the main code
- OSICTIClientSample.Designer.vb the main formular
- OSICTICallClientClass.vb contains a class which can be used for receiving events per call. This is necessary, because you can‘t use an array of „Dim WithEvents“-variables.
- AssemblyInfo.vb Assembly Information for .Net JIT
- OSICTIClientSample.resx .Net Managed Resource File

In addition to the fundamental principles of the programming the OSITRON-CTI-COM-server the sample handles an own call list in a TabStrip-control.

OSICTIClientSample.vb

```
Option Strict Off
Option Explicit On

Friend Class Dialog
    Inherits System.Windows.Forms.Form

Dim WithEvents OSICTIClient As OSICTIControlCenter.OSICTIApp 'COM-
Connection Point
'For each asynchrone Event Call of the OSICTIClient is an Delegate-
Sub needed with the same parameters used by the COM-Events
Delegate Sub NewCallEventDelegate(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall)
Delegate Sub CallRemoveEventDelegate(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall)

Private Sub About_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles About.Click
    OSICTIClient.About()
End Sub

Private Sub CallTabStrip_ClickEvent(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
CallTabStrip.ClickEvent
    Dim ActCallClient As OSICTICallClientClss
    ActCallClient = CallTabStrip.Tabs.Item(
    CallTabStrip.SelectedItem.Index).Tag
    Call UpdateInfo(ActCallClient)
End Sub

Private Sub Clear_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles Clear.Click
    StateInfo.Items.Clear() 'Clear list of all shown Messages
End Sub
```

```
Private Sub ConfigureAddressBooks_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
ConfigureAddressBooks.Click
    OSICTIClient.ConfigureAddressBooks()
End Sub

Private Sub ConfigureCallDeflectManager_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
ConfigureCallDeflectManager.Click
    OSICTIClient.ConfigureCallDeflectManager()
End Sub

Private Sub ConfigureCallForwardManager_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
ConfigureCallForwardManager.Click
    OSICTIClient.ConfigureCallForwardManager()
End Sub

Private Sub ConfigureHotKeys_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
ConfigureHotKeys.Click
    OSICTIClient.ConfigureHotKeys()
End Sub

Private Sub Dial_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles Dial.Click 'Dial the given
Number
    Dim NewCall As OSICTIControlCenter.IOSICTICall
    NewCall = OSICTIClient.CreateCall 'Create new Call Object
    NewCall.DialAddress = DialNo.Text 'asign Number
    NewCall.Dial() 'Dial Number
End Sub

Private Sub Drop_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles Drop.Click
    Dim ActCallClient As OSICTICallClientClss
    Dim i As Short

    For i = 1 To CallTabStrip.Tabs.Count
        If CallTabStrip.Tabs.Item(i).Selected Then
            ActCallClient = CallTabStrip.Tabs.Item(i).Tag
            ActCallClient.CTICallClient.Drop()
        End If
    Next
End Sub

Private Sub Dialog_Load(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles MyBase.Load
    Dim i As Short
    Dim CallColl As OSICollection.IOSIStaticCollection
    OSICTIClient = New OSICTIControlCenter.OSICTIApp
    CallColl = OSICTIClient.CallList
    CallTabStrip.Tabs.Remove((1))
    For i = 1 To CallColl.Count
        OSICTIClient_NewCallEvent(OSICTIClient, CallColl.Item(i))
    Next
End Sub

Private Sub MakeConfiguration_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
MakeConfiguration.Click
    OSICTIClient.MakeConfiguration()
End Sub

Private Sub OSICTIClient_CallRemoveEvent(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall) Handles
OSICTIClient.CallRemoveEvent
    Dim CallRemoveEventD As CallRemoveEventDelegate = New
    CallRemoveEventDelegate(AddressOf CallRemoveEventDel)
```

```
    Me.BeginInvoke(CallRemoveEventD, New Object() {piCTI, piCall})
End Sub

Private Sub CallRemoveEventDel(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall)
    Dim ActCall As OSICTIControlCenter.IOSICTICall
    Dim ActCallClient As OSICTICallClientClss
    Dim i As Short

    For i = 1 To CallTabStrip.Tabs.Count
        ActCallClient = CallTabStrip.Tabs.Item(i).Tag
        ActCall = ActCallClient.CTICallClient
        If piCall Is ActCall Then
            If CallTabStrip.Tabs.Item(i).Selected Then
                If CallTabStrip.Tabs.Count > 1 Then
                    If i = 1 Then
                        CallTabStrip.Tabs.Item(2).Selected = True
                    Else
                        CallTabStrip.Tabs.Item(i - 1).Selected = True
                    End If
                End If
            End If
            CallTabStrip.Tabs(i).Tag = Nothing
            CallTabStrip.Tabs.Remove((i))
            Exit For
        End If
    Next
    If CallTabStrip.Tabs.Count = 0 Then
        Me.Frame1.Visible = False
        Me.Frame2.Visible = False
    End If
End Sub

Private Sub OSICTIClient_NewCallEvent(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall) Handles OSICTIClient.NewCallEvent
    Dim NewCallEventD As NewCallEventDelegate = New
    NewCallEventDelegate(AddressOf NewCallEventDel)
    Me.BeginInvoke(NewCallEventD, New Object() {piCTI, piCall})
End Sub

Private Sub NewCallEventDel(ByVal piCTI As
OSICTIControlCenter.OSICTIApp, ByVal piCall As
OSICTIControlCenter.OSICTICall)
    Dim NewTab As MSComctlLib.ITab
    Dim ActCallClient As OSICTICallClientClss
    Dim ActCallClient2 As System.Object
    ActCallClient = New OSICTICallClientClss(Me)
    ActCallClient2 = ActCallClient
    ActCallClient.CTICallClient = piCall
    If CallTabStrip.Tabs.Count = 0 Then
        Me.Frame1.Visible = True
        Me.Frame2.Visible = True
        Call UpdateInfo(ActCallClient)
    End If
    NewTab = CallTabStrip.Tabs.Add
    ActCallClient.m_ITab = NewTab
    NewTab.let_Tag(ActCallClient2)
End Sub

Private Sub SelectLanguage_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
SelectLanguage.Click
    OSICTIClient.SelectLanguage()
End Sub

Private Sub SelectLines_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles SelectLines.Click
    OSICTIClient.SelectLines()
```

```
End Sub

Private Sub UpdateInfo(ByVal CallClient As OSICTICallClientClss)
    InfoCalledAdr.Text = CallClient.CTICallClient.CalledAddress
    InfoCallingAdr.Text = CallClient.CTICallClient.CallingAddress
    InfoCallState.Text =
    CallClient.GetCallState(CallClient.CTICallClient.State)
    InfoCharging.Text = CallClient.CTICallClient.Charging
    InfoConnectedAdr.Text =
    CallClient.CTICallClient.ConnectedAddress
    InfoDateTime.Text = CallClient.CTICallClient.DateTime.ToString
    InfoDisplay.Text = CallClient.CTICallClient.Display
    InfoName.Text = CallClient.CTICallClient.Name
    InfoRedirecting.Text =
    CallClient.CTICallClient.RedirectingAddress
    InfoRedirection.Text =
    CallClient.CTICallClient.RedirectionAddress
End Sub
End Class
```

OSICTICallClientClass.vb

```
Option Strict Off
Option Explicit On

Friend Class OSICTICallClientClss
    Public m_State As String
    Public m_ITab As MSComctlLib.ITab
    'this reference to the Form is needed to make it possible that
    the eventhandling-Subs can be executed in the Forms-Thread
    Dim Parent As Dialog
    'COM-Connection Point
    Public WithEvents CTICallClient As
    OSICTIControlCenter.OSICTICall

    'For each asynchrone Event Call of the OSICTIClient is an
    Delegate-Sub needed with the same parameters used by the COM-
    Events
    Delegate Sub CalledAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrCalledAddress As
    String)
    Delegate Sub CallingAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrCallingAddress As
    String)
    Delegate Sub CanonicalAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrCanonicalAddress As
    String)
    Delegate Sub ChargingEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal varCharging As Decimal)
    Delegate Sub ConnectedAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrConnectedAddress As
    String)
    Delegate Sub DateTimeEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal varDateTime As Date)
    Delegate Sub DialableAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrDialableAddress As
    String)
    Delegate Sub DialAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrDialAddress As
    String)
    Delegate Sub DirectionEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal eDirection As
    OSICTIControlCenter.EOSICTICallDirection)
    Delegate Sub DisplayEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrDisplay As String)
    Delegate Sub NameEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrName As String)
```

```
    Delegate Sub NormalizedAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrNormalizedAddress As
    String)
    Delegate Sub RedirectingAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrRedirectingAddress As
    String)
    Delegate Sub RedirectionAddressEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal bstrRedirectionAddress As
    String)
    Delegate Sub StateEventDelegate(ByVal piCall As
    OSICTIControlCenter.OSICTICall, ByVal eState As
    OSICTIControlCenter.EOSICTICallState)

Public Sub New(ByRef aParent As Dialog)
    Parent = aParent
End Sub

Private Sub CTICallClient_CalledAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCalledAddress As String)
Handles CTICallClient.CalledAddressEvent
    Dim CalledAddressEventD As CalledAddressEventDelegate = New
    CalledAddressEventDelegate(AddressOf CalledAddressEventDel)
    Parent.BeginInvoke(CalledAddressEventD, New Object() {piCall,
    bstrCalledAddress})
End Sub

Private Sub CalledAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCalledAddress As String)
    Parent.StateInfo.Items.Add("CallInfo Called Address:")
    Parent.StateInfo.Items.Add("     " & bstrCalledAddress)
    If m_ITab.Selected Then
        Parent.InfoCalledAdr.Text = bstrCalledAddress
        m_ITab.Caption = bstrCalledAddress
    End If
End Sub

Private Sub CTICallClient_CallingAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCallingAddress As String)
Handles CTICallClient.CallingAddressEvent
    Dim CallingAddressEventD As CallingAddressEventDelegate = New
    CallingAddressEventDelegate(AddressOf CallingAddressEventDel)
    Parent.BeginInvoke(CallingAddressEventD, New Object() {piCall,
    bstrCallingAddress})
End Sub

Private Sub CallingAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCallingAddress As String)
    Parent.StateInfo.Items.Add("CallInfo Calling Address:")
    Parent.StateInfo.Items.Add("     " & bstrCallingAddress)
    If m_ITab.Selected Then
        Parent.InfoCallingAdr.Text = bstrCallingAddress
    End If
End Sub

Private Sub CTICallClient_CanonicalAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCanonicalAddress As
String) Handles CTICallClient.CanonicalAddressEvent
    Dim CanonicalAddressEventD As CanonicalAddressEventDelegate =
    New CanonicalAddressEventDelegate(AddressOf
    CanonicalAddressEventDel)
    Parent.BeginInvoke(CanonicalAddressEventD, New Object()
    {piCall, bstrCanonicalAddress})
End Sub

Private Sub CanonicalAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrCanonicalAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Canonical Address:")
    Parent.StateInfo.Items.Add("     " & bstrCanonicalAddress)
    If m_ITab.Selected Then
```

```
        'Parent.InfoCanonicalAdr.Caption = bstrCanonicalAddress
    End If
End Sub

Private Sub CTICallClient_ChargingEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal varCharging As Decimal)
Handles CTICallClient.ChargingEvent
    Dim ChargingEventD As ChargingEventDelegate = New
    ChargingEventDelegate(AddressOf ChargingEventDel)
    Parent.BeginInvoke(ChargingEventD, New Object() {piCall,
    varCharging})
End Sub

Private Sub ChargingEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal varCharging As Decimal)
    Parent.StateInfo.Items.Add("CallInfo Charging:")
    Parent.StateInfo.Items.Add(CStr(CDbl("     ") + varCharging))
    If m_ITab.Selected Then
        Parent.InfoCharging.Text = CStr(varCharging)
    End If
End Sub

Private Sub CTICallClient_ConnectedAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrConnectedAddress As
String) Handles CTICallClient.ConnectedAddressEvent
    Dim ConnectedAddressEventD As ConnectedAddressEventDelegate =
    New ConnectedAddressEventDelegate(AddressOf
    ConnectedAddressEventDel)
    Parent.BeginInvoke(ConnectedAddressEventD, New Object()
    {piCall, bstrConnectedAddress})
End Sub

Private Sub ConnectedAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrConnectedAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Connected Address:")
    Parent.StateInfo.Items.Add("     " & bstrConnectedAddress)
    If m_ITab.Selected Then
        Parent.InfoConnectedAdr.Text = bstrConnectedAddress
    End If
End Sub

Private Sub CTICallClient_DateTimeEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal varDateTime As Date) Handles
CTICallClient.DateTimeEvent
    Dim DateTimeEventD As DateTimeEventDelegate = New
    DateTimeEventDelegate(AddressOf DateTimeEventDel)
    Parent.BeginInvoke(DateTimeEventD, New Object() {piCall,
    varDateTime})
End Sub

Private Sub DateTimeEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal varDateTime As Date)
    Parent.StateInfo.Items.Add("CallInfo DateTime:")

    Parent.StateInfo.Items.Add(CStr(System.DateTime.FromOADate(CDat
    e("     ").ToOADate + varDateTime.ToOADate)))
    If m_ITab.Selected Then
        Parent.InfoDateTime.Text = CStr(varDateTime)
    End If
End Sub

Private Sub CTICallClient_DialableAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDialableAddress As
String) Handles CTICallClient.DialableAddressEvent
    Dim DialableAddressEventD As DialableAddressEventDelegate = New
    DialableAddressEventDelegate(AddressOf DialableAddressEventDel)
    Parent.BeginInvoke(DialableAddressEventD, New Object() {piCall,
    bstrDialableAddress})
End Sub
```

```
Private Sub DialableAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDialableAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Dialable Address:")
    Parent.StateInfo.Items.Add("     " & bstrDialableAddress)
    If m_ITab.Selected Then
        'Parent.InfoDialableAdr.Caption = bstrDialableAddress
    End If
End Sub

Private Sub CTICallClient_DialAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDialAddress As String)
Handles CTICallClient.DialableAddressEvent
    Dim DialAddressEventD As DialAddressEventDelegate = New
    DialAddressEventDelegate(AddressOf DialAddressEventDel)
    Parent.BeginInvoke(DialAddressEventD, New Object() {piCall,
    bstrDialAddress})
End Sub

Private Sub DialAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDialAddress As String)
    Parent.StateInfo.Items.Add("CallInfo Dial Address:")
    Parent.StateInfo.Items.Add("     " & bstrDialAddress)
    If m_ITab.Selected Then
        'Parent.InfoDialAdr.Caption = bstrDialAddress
    End If
End Sub

Private Sub CTICallClient_DirectionEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal eDirection As
OSICTIControlCenter.EOSICTICallDirection) Handles
CTICallClient.DirectionEvent
    Dim DirectionEventD As DirectionEventDelegate = New
    DirectionEventDelegate(AddressOf DirectionEventDel)
    Parent.BeginInvoke(DirectionEventD, New Object() {piCall,
    eDirection})
End Sub

Private Sub DirectionEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal eDirection As
OSICTIControlCenter.EOSICTICallDirection)
    Parent.StateInfo.Items.Add("CallInfo Direction:")
    Parent.StateInfo.Items.Add(CStr(CDbl("     ") + eDirection))
    If m_ITab.Selected Then
        Parent.InfoDirection.Text = CStr(eDirection)
    End If
End Sub

Private Sub CTICallClient_DisplayEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDisplay As String)
Handles CTICallClient.DisplayEvent
    Dim DisplayEventD As DisplayEventDelegate = New
    DisplayEventDelegate(AddressOf DisplayEventDel)
    Parent.BeginInvoke(DisplayEventD, New Object() {piCall,
    bstrDisplay})
End Sub
Private Sub DisplayEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrDisplay As String)
    Parent.StateInfo.Items.Add("CallInfo Display:")
    Parent.StateInfo.Items.Add("     " & bstrDisplay)
    If m_ITab.Selected Then
        Parent.InfoDisplay.Text = bstrDisplay
    End If
End Sub

Private Sub CTICallClient_NameEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrName As String) Handles
CTICallClient.NameEvent
```

```
    Dim NameEventD As NameEventDelegate = New
    NameEventDelegate(AddressOf NameEventDel)
    Parent.BeginInvoke(NameEventD, New Object() {piCall, bstrName})
End Sub

Private Sub NameEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrName As String)
    Parent.StateInfo.Items.Add("CallInfo Name:")
    Parent.StateInfo.Items.Add("     " & bstrName)
    If m_ITab.Selected Then
        Parent.InfoName.Text = bstrName
    End If
End Sub

Private Sub CTICallClient_NormalizedAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrNormalizedAddress As
String) Handles CTICallClient.NormalizedAddressEvent
    Dim NormalizedAddressEventD As NormalizedAddressEventDelegate =
    New NormalizedAddressEventDelegate(AddressOf
    NormalizedAddressEventDel)
    Parent.BeginInvoke(NormalizedAddressEventD, New Object()
    {piCall, bstrNormalizedAddress})
End Sub

Private Sub NormalizedAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrNormalizedAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Normalized Address:")
    Parent.StateInfo.Items.Add("     " & bstrNormalizedAddress)
    If m_ITab.Selected Then
        'Parent.InfoNormalizedAdr.Caption = bstrNormalizedAddress
    End If
End Sub

Private Sub CTICallClient_RedirectingAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrRedirectingAddress As
String) Handles CTICallClient.RedirectingAddressEvent
    Dim RedirectingAddressEventD As RedirectingAddressEventDelegate
    = New RedirectingAddressEventDelegate(AddressOf
    RedirectingAddressEventDel)
    Parent.BeginInvoke(RedirectingAddressEventD, New Object()
    {piCall, bstrRedirectingAddress})
End Sub

Private Sub RedirectingAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrRedirectingAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Redirecting Address:")
    Parent.StateInfo.Items.Add("     " & bstrRedirectingAddress)
    If m_ITab.Selected Then
        Parent.InfoRedirecting.Text = bstrRedirectingAddress
    End If
End Sub

Private Sub CTICallClient_RedirectionAddressEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrRedirectionAddress As
String) Handles CTICallClient.RedirectionAddressEvent
    Dim RedirectionAddressEventD As RedirectionAddressEventDelegate
    = New RedirectionAddressEventDelegate(AddressOf
    RedirectionAddressEventDel)
    Parent.BeginInvoke(RedirectionAddressEventD, New Object()
    {piCall, bstrRedirectionAddress})
End Sub

Private Sub RedirectionAddressEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal bstrRedirectionAddress As
String)
    Parent.StateInfo.Items.Add("CallInfo Redirection Address:")
    Parent.StateInfo.Items.Add("     " & bstrRedirectionAddress)
    If m_ITab.Selected Then
```

```
        Parent.InfoRedirection.Text = bstrRedirectionAddress
    End If
End Sub

Private Sub CTICallClient_StateEvent(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal eState As
OSICTIControlCenter.EOSICTICallState) Handles
CTICallClient.StateEvent
    Dim StateEventD As StateEventDelegate = New
    StateEventDelegate(AddressOf StateEventDel)
    Parent.BeginInvoke(StateEventD, New Object() {piCall, eState})
End Sub

Private Sub StateEventDel(ByVal piCall As
OSICTIControlCenter.OSICTICall, ByVal eState As
OSICTIControlCenter.EOSICTICallState)
    Dim State As String
    State = GetCallState(eState)
    Parent.StateInfo.Items.Add("CallState:")
    Parent.StateInfo.Items.Add("     " & State)
    If m_ITab.Selected Then
        Parent.InfoCallState.Text = State
    End If
End Sub

Public Function GetCallState(ByVal eState As
OSICTIControlCenter.EOSICTICallState) As String
    Dim State As String
    State = ""
    Select Case eState
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateIdle
        State = "Idle"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateRinging
        State = "Ringing"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateDialing
        State = "Dialing"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateProceedi
        ng
        State = "Proceeding"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateRingback
        State = "Ringback"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateConnecte
        d
        State = "Connected"
        Case
        OSICTIControlCenter.EOSICTICallState.OSICTICallStateDisconne
        cted
        State = "Disconnected"
    End Select
    GetCallState = State
End Function
End Class
```

12. Status of the document

Date	Editor	Status	Changes
20.09.1999	HTR	Preliminary Rev. 02/99	Changing of the Defines for ServerState, State and Info.
29.09.1999	HTR	Preliminary Rev. 03/99	The following functions are added - ShowUp(), HideDown(), ToggleMinMode(), Drop()
1. November 1999	RSCH	Release Rev 03/99	Released
1.Mai 2000	RSCH	Version 3.0 Rev 01/2000	New Release for OSITRON CTI 3.0
8. January 2001	HOSP	Version 3.0 Rev 01/2001	New COM Interface
9. January 2001	THK	Version 3.0 Rev 01/2001	New COM Interface
31. January 2001	RSCH	Version 3.0 Rev 01/2001	Delphi Sample
23. March 2001	JGE	Version 3.0 Rev 01/2001	Visual C++ Sample
12. February 2004	RSCH	Version 3.3 Rev. 01/2004	ACD Support added.
25. April 2005	KKO	Version 3.5 Rev. 01/2005	Hold Info added. ACDAppStateChanged ACD Application Object
15. January 2007	MHO	Version 3.7 Rev 01.2007	Visual Basic .Net Sample
25. April 2007	JKA	Version 3.7 Rev 01.2007	ActiveX Sample
28. October 2011	RSA	Version 4.0	Information about the NewCallEvent extended.