Kofax Communication Server

TCRT Technical Manual

Version: 10.3.0

Date: 2019-12-13

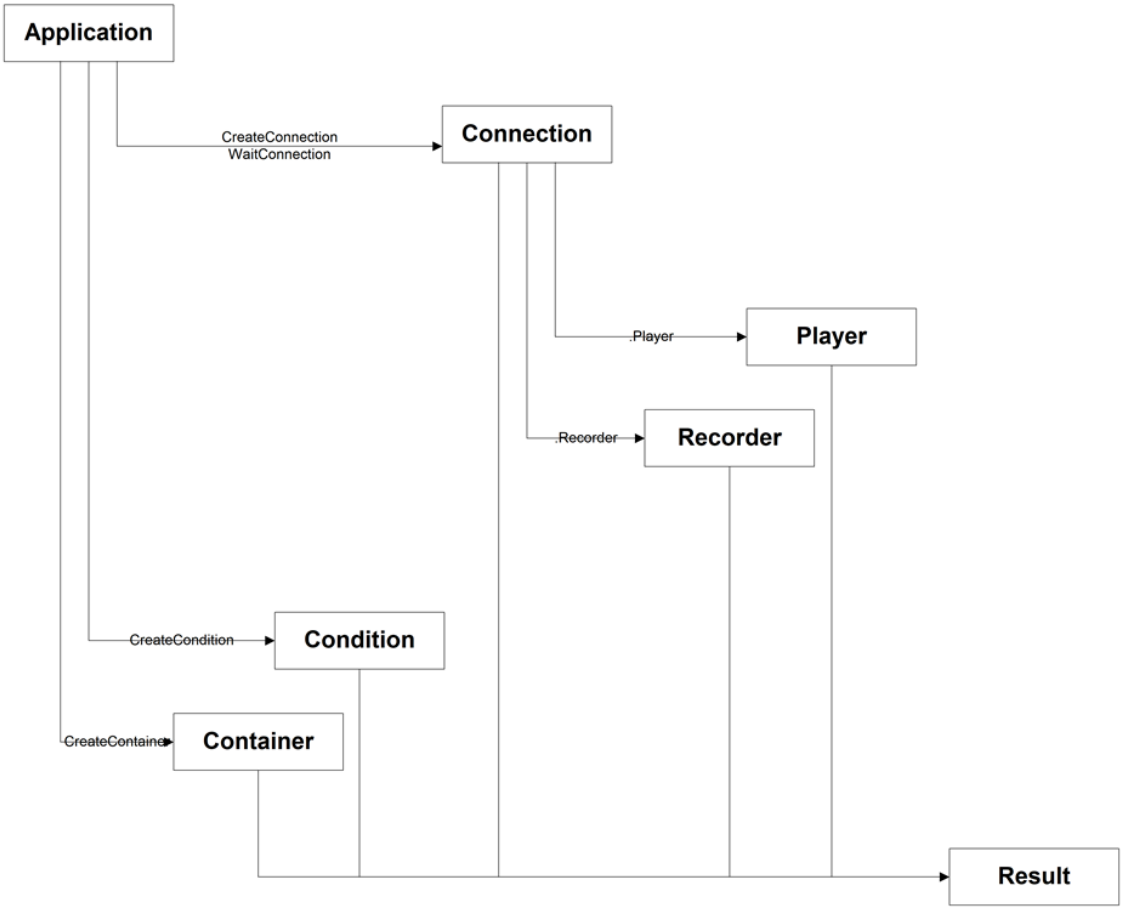
KOFAX

Legal Notice

Table of Contents

Chapter 1

Object Model



Chapter 2

COM Interface

Every Object provided by TCRT uses the Free-Threaded Marshaler. This means, system provided marshaling of interface pointers between different apartments and threads is bypassed and no marshaling is done. The reason for marshaling is to provide thread locking for non-reentrant objects, and to switch to the correct thread for objects bound to a certain thread (STA). These reasons do not apply to TCRT objects. They are all thread safe and provide internal synchronization mechanisms. TCRT objects can be used freely between apartments. TCRT objects are registered using the threading model “Both” allowing to adopt both the Apartment Model and the Free-Threaded Model as used by the application.

In blocking (synchronous) calls (e.g. Play()), TCRT continues to handle windows messages to prevent from dead-locking the application in STA models. Only one blocking call can be active at the same time.

Events are always generated from a specific apartment dedicated to the corresponding connection object. This apartment is initialized as STA. For all events caused by a specific connection the same apartment is used. The apartment’s lifetime is bound to the lifetime of the connection object.

The reference to the connection-interface, used to pass the event into the application, gets marshaled into the connection object’s event-apartment before it is used. This allows safe calls to any apartment’s connection-interface.

Functionality Supported by Every Object

This section provides functionality supported by every object.

LastError Property

Description

Returns the number (long value) of the last error occurred. This is a read-only property.

Syntax

```
Object.LastError
```

read-only

Parameter	Description
Object	Any TCRT object

Remarks

- Reading *LastError()* does not reset the last error to resOk.
- The last error is overwritten by new errors.

LastErrorDescription Property

Description

Returns the last error description(string value) . This is a read-only property.

Error codes and descriptions are always thread, application and object specific. (e.g.: The *.LastError* property of a *Player* object will return the last error that occurred in this *Player* object.)

Syntax

```
Object.LastErrorDescription
```

read-only

Parameter	Description
Object	Any TCRT object

Remarks:

- Only errors are written to the internal error variables.
- Descriptions are full English sentences. The text can be used for trace output, but not for user interaction

Application Property

Description

Returns a reference to the *Application* object the current object belongs to. This is a read-only property. However, there is only one real parent object (see *Object Dependencies*) per process. An object may store references to the *Application* object it corresponds to. Returns 0 on error or if there is no parent object. *Application* objects themselves have no *.Application* property.

Syntax

```
Object.Application
```

read-only

Parameter	Description
Object	Any TCRT object

WhatAmI Property

Description

Every object can be identified using the “WhatAmI” read-only property. It returns the text description of the object(string value). This is especially useful, when using variants to store objects. This is a Read-Only property.

Syntax

```
Object.WhatAmI
```

read-only

Parameter	Description
Object	Any TCRT object

Example:

```
myApplication.WhatAmI has a value of "Application".
```

Trace Method

Description

Writes the given string to the trace file.

Syntax

```
Object.Trace(text)
```

Parameter	Description
Object	Any TCRT object
Text	A string value that will be written into the trace file

Remarks:

- The trace file is always located in c:\tcoss\trace.
- TCLIB32.DLL common functionality is used to restrict traces in size, and to provide special identifiers like thread id, timestamps or similar.

See chapter configuration for trace level options.

Objects

This section describes the objects in TCRT.

Application

This section describes the properties in TCRT.

Language Property

Description

Use this property to set or read the application's default language. Notice that the language can also be set per Connection and even per *Play()*.

Syntax

```
Object.Language = [value]
```

Parameter	Description
Object	TCRT application object
Value	A variant value that defines the language

Setting	Description
01	English
02	German
07	Japanese

Property

Description

This Property is not supported. Invoking it has not effect.

Syntax

```
Object.Property(Parameter [engine])
```

AttPtr2Str Function

Description

This function returns a hexadecimal string representation of a TFC.Attachment or TFC.Message COM object. Note that the function makes an AddRef() to this interface pointer. For details on TFC see the *TFC Reference Manual*.

Syntax

```
Object.AttPtr2Str(Parameter) as String
```

Parameter	Description
Object	TCRT application object
Parameter	a TFC.Attachment or TFC.Message object

Return value

A string value, which represents a pointer to a TFC.Attachment or TFC.Message object

Remarks: Obsolete since TCRT 2.03.05. Use AttPtr2Str of the connection object instead.

CreateCondition Method

Description

This method returns an empty Condition object. This method is provided for compatibility reasons.

Syntax

```
Object.CreateCondition
```

Return value

On success the function returns an empty Condition object.

Remarks: Though conditions are not supported in the connection object to stop Play() or Record(), this method still returns a valid condition object.

CreateConnection Method

Description

This method establishes a connection to the resource specified as *parameter* and returns a connection object. The parameter allows addressing any resource at any engine at any node within the network. If the connection cannot be established *Nothing* is returned.

Syntax

```
Object.CreateConnection(Parameter)
```

Parameter	Description
Object	TCRT Application object
Parameter	A variant value that specify the resource for the connection

Return value

On success the method returns a `Connection` object. On failure the method returns `Nothing`.

Connection Parameter

The TCRT methods CreateConnection(), Play(), Record(), Set() and Transfer() use the same parameter to specify the resource to use for transferring data. This parameter is of type Variant and can be of following subtypes:

IUnknown , IDispatch

Target engine: resource

This parameter type is a reference to an object. Valid objects are TFC.Attachment or TCRT.Container. This parameter implicitly addresses the “resource” engine as target engine. This means the “resource” engine implementation is responsible for accepting the objects as data source and/or as sink.

Current implementation:

The TFC.Attachment parameter is supported both as data source and as data sink i.e. you can play from and record into this object.

The TCRT.Container is supported as data source only.

String

Target engine: any

This parameter allows addressing any resource at any engine at any node within the network. In addition, the direction of the data transfer for later playing or recording can be specified.

Syntax:

```
['['<direction>']']<engine>['@'<node>][':'<address>]
```

or (defaults to “resource” engine)

```
['['<direction>']']'<resource-engine parameter>
```

[…] optional

‘x’ literal

<x> identifier

<engine>	any valid engine name e.g. “sound”, “telephone”, “resource”
<node>	a valid node name e.g. the IP address or workstation name of the server to look for the specified resource. The connection is always routed to the remote engine.
<address>	an address string valid for the resource specified with <engine>. For the “resource” engine this can be <resource-engine par>.
<direction>	‘in’, ‘out’, ‘in,out’ or ‘fdup’(behavior not defined). ‘in’ means that this halfcall receives data.
<resource-engine par>	a string with the syntax as defined by the “resource” engine. The string must contain at least one pair of angle brackets ‘<’…’>’ to be identified as resource engine parameter.

Examples:

```
telephone:66133888
telephone@TCVOICESRV:66133888
FILE:C:\temp\test.wav)
[out]resource:<FILE "c:\recorded.wav">
[out]resource@YOURPC:<FILE "c:\recorded.wav">
resource@YOURPC: <TTS "Here you can enter the text you will hear.">
```

CreateConnectionAsynch Method

Description

The method establishes outbound call in an asynchronous way (non-blocking).

The method call creates the connection object, initiates call establishment and returns immediately.

If the return value was OK – it means that the outbound call is being established (by the telephone engine). So at this moment the application must not call any Play or Record methods on this connection (they would return an error).

In order to guarantee the compatibility with “older” Voice server (not supporting asynchronous call request), new 2nd parameter “BOOL *pAsynchResp” was defined with CreateConnectionAsynch() method.

If the application calls CreateConnectionAsynch() and the call is handled by the “new” Voice server supporting asynchronous call, on return of CreateConnectionAsynch() the pAsynchResp parameter is set to TRUE to indicate that this call is being established asynchronously and therefore the application must wait from MSG_STATUS messages (see below) prior to calling any of Play or Record methods.

If the application calls CreateConnectionAsynch() and the call is handled by the “old” Voice server NOT supporting asynchronous call, on return of CreateConnectionAsynch() the pAsynchResp parameter

is set to FALSE to indicate that this call has been established synchronously and therefore is already established on return of CreateConnectionAsynch() (and NO MSG_STATUS messages will arrive afterwards).

Syntax

```
Object.CreateConnectionAsynch(Parameter, AsynchResp)
```

Parameter	Description
Object	TCRT application object
Parameter	A variant value that specify the resource for the connection
AsynchResp	long value that indicates it the call is established asynchronously

Return value

On success the method returns a `Connection` object. On failure the method returns `Nothing`.

Remarks: requires TCRT 2.03.00

CreateContainer Method

Description

This method returns an empty container object.

Syntax

```
Object.CreateContainer
```

Parameter	Description
Object	TCRT application object

Return value

On success the method returns a `Container` object. On failure the method returns `Nothing`.

InitiateConnection Method

Description

The method InitiateConnection() simulates an incoming call from an engine without any existing connections. This method is called by TCECP if an active call plug-in is configured. For each configured instance this method is called. With this call the INIT event is send to the plug-in.

Syntax

```
Object.InitiateConnection(parameter)
```

Parameter	Description
Object	TCRT Application object
Parameter	A variant value that specify the resource for the connection

Start Method

Description

This method will start all loaded engines. This method is invoked implicitly when trying to use any engine for the first time.

Syntax

```
Object.Start
```

Parameter	Description
Object	TCRT Application object

Return value

A TCRT `Result` object

TestEvent Method

Description

Manually generates an event with the given parameters for this object.

Syntax

```
Object.TestEvent(EventType As Long, EventData As String, con As Connection, Result As
 Long)
```

Parameter	Description
Object	TCRT Application object
EventType	A long value that represents the event type(e.g. msgDTMF)
EventData	A string value that contains event specific data (e.g. for a DTMF event it contains the key)
Con	A valid TCRT Connection object
Result	A long value that indicates success or failure

Return value

If the method succeeds, the value Result is zero.

If the method fails, the value Result is nonzero.

Error codes

```
resOk             =      0
resErrInt         =      1
resOutOfMemory    =      2
resBadParam       =      3
resTimeout        =      4
resNotSupported   =      5
resNotConnected   =      6
resBadChannel     =      7
```

```
resBadChannelType =      8
resBadEngine      =      9
resBusy           =     10
resNoResponse     =     11
resErr            =     12
resInvalidHandle  =     13
resCancelled      =     14
```

Event types

COM Interface	Value
MsgDiscInd	0x80000002
MsgInit	0x80000003
MsgDone	0x80000004
MsgDTMF	0x80000012
MsgPlayEnd	0x80000019
MsgRecordEnd	0x8000001A
MsgTimeout	0x8000001F
msgStatus	0x80000020

Event

Description

Called on application specific events (e.g.: MSG_CONNECT, …), and for any connection events that have no event sink registered.

Syntax

```
Function App_Event(ByVal EventNum As Long, ByVal EventDescr As Variant, ByVal Conn As
 Connection, Result As Variant) As Long
```

Parameter	Description
EventNum	A long value that identifies the event
EventDescr	A variant value for the event description. With the current version EventDesc is used only for the msgDTMF event and for msgDiscInd with an asynch. call .
Conn	A TCRT Connection object
Result	The parameter Result has no function in the current release of TCRT. For the first event handler it is initialized to a value of 0. The event handler may change this parameter and pass the new value back. The new value is passed to the next event handler. TCRT does not interpret this value.

Return value

By default the event handler return value is resOk. Thus once the event was passed to a handler function it is not automatically passed to another. To forward a specific event to other handlers, the handler function must return a non-zero value (e.g. resErr).

Register Method

Description

This method is provided for compatibility. It has no functionality. The result is OK.

Syntax

```
Function Register(name As String) As Result
```

UnRegister Method

Description

This method is provided for compatibility. It has no functionality. The result is OK.

Syntax

```
Function UnRegister(name As String) As Result
```

WaitConnection Method

Description

This method is provided for compatibility. It has no functionality. It returns Nothing.

Syntax

```
Function WaitConnection([ConnectionType As String = "*"], [timeout As Long = -1]) As
 Connection
```

Connection

This section describes the connection.

Property

Description

Returns or sets a connection property

Syntax

```
Object.Property(parameter As String, [engine as Variant]) [=varPropertyValue]
```

Parameter	Description
Parameter	the name of the property
Engine	the name of the engine

Return value

The return value is the value of the engine property.

Supported property names:

Name		Result	Description	Example
CallerID	R/O	String	source address	"Phone:4312345", "VoIP:voice.tc.com:1234"
CalledID	R/O	String	destination address	"telephone:12345", "Client:192.168.1.1:12" "telephone: dxxxPri1Tr12"
CallerEngine	R/O	String	local engine generating or forwarding the call	telephone
CalledEngine	R/O	String	target resource	telephone
Calltransfer	R/W	String	Transfer number	
CancelTransfer	R/W	Long		
Faxcall				
LineID	R/O	String	Line ID	"telephone: dxxxPri1Tr12"
Prompt:…	R/O	String		"text:<number>"
BlockSize	R/W	Long	Compatibility	
Size	R/W	Long	duration of the last record in milliseconds(Compatibility)	
Position	R/O	Long	the current position in ms for *Play()*, *Record()* or *Start()*.	
Status	R/O	Long	0 if there is no active data transfer	
Volume	R/W	Long	volume for sound or telephone engine in % (0 – 100)	
Silencetimeout	R/W	Long	sets the silence timeout in milliseconds	
Lengthtimeout	R/W	Long	sets the maximum record length in milliseconds	
TTSSpeakerList	R/O	String	GUIDs of the installed TTS speakers	
Mailboxdetection	W/O	String	sets the parameters for mailbox detection	[DetTime=a,Toff1=b,DetEnergy=c,BlkLen=d,DetDelay=e,]Fq1= values in [] are optional, sequence Fq1,Fq2, OnTime may come max. 16 times The FFT mode is invoked by setting "Fq1=-1,Fq2=0,OnTime=t" DetDelay: in ms, delay the sending of MSG_STATUS after beep recognition DetEnergy: in percent of the total energy, default is 70% Toff1: in ms, the minimum length of silence before the beep tone

Remarks: You must create a connection to the resource engine with parameter TTSProperties to get the TTSSpeakerList property. e.g. `resource@VOICESERVERNAME:TTSProperties`

The mailbox detection is currently supported only for LS1, it is not supported for H.323

Example:

```
Rem reading a property
Variant result
result = connect.Property("CallerID")
If IsEmpty(result) Then
   print "A stranger is calling you"
Else
   print "You have been called by " + result
End If

Rem reading the TTS speaker list
Set tcrtCon = tcrtApp.CreateConnection("resource@DEMOVOICE:TTSProperties"
vSpeakers = tcrtConn.Property("TTSSpeakerList")
Rem vSpeakers is a string that contains all TTS speakers that are installed on
Rem the voice server
```

Syntax of the TTS speaker list

TTSGUID=TTSLanguageID,TTSPitch,TTSRate,TTSVolume,TTSSpeakerName;

- TTS speakers are delimited with a semicolon
- Parameters of a TTS speaker are delimited with a comma

Speaker list example:

```
c77c5170-2867-11d0-847b-444553540000=0409,20,82,200,Gerhard (German);c77c5170-28
67-11d0-847b-444553540000=0409,20,82,200,Roger (British English);c77c51c0-2867-1
1d0-847b-444553540000=0409,67,58,99,Mike (for Telephone);c77c51d0-2867-11d0-847b
-444553540000=0409,66,58,99,Mary (for Telephone)
```

Cookie Property

Description

This property used to assign connection-related information to the connection object. It assigns an arbitrary variant to the connection. Use this property to retrieve this variant again.

Syntax

```
Object.Cookie [=varCookie]
```

LanguageProperty

Description

Returns or sets the connection's language.

Syntax

```
Object.Language [= varLanguageIdentifier]
```

Remarks: The language can also be set per Play().

LastActionTime Property

Description

This property has no functionality. It always returns 0.

Syntax

```
Object.LastActionTime
read-only
```

LastEvent Property

Description

This property is used to get the last event from the connection object. Use this property if your application does not support the event handlers of the connection.

Syntax

```
varEvent = Object.LastEvent
```

Player Property

Description

Use this property to the directly access the connection's Player object. You may safely assume that this object will always exist.

Syntax

```
Object.Player
read-only
```

Return values

Returns a TCRT player object

Position Property

Description

This property returns the current play or record position in milliseconds. When no play or record is active the last play or record position is returned. When assigning new values to this property, the resource engine will seek to the given position.

Syntax

```
Object.Position
read-only
```

Return values

This property holds the current play or record position in milliseconds.

Recorder Property

Description

Use this property to the directly access the connection's Recorder object. You may safely assume that this object will always exist.

Syntax

```
Object.Recorder
read-only
```

Return values

Returns a TCRT recorder object

ConnectionHandle Property

Description

This property returns the host engine handle of the connection. The handle may be used as a unique ID for the connection.

Syntax

```
Object.ConnectionHandle
read-only
```

Return values

The host engine handle(long value) of the connection.

Accept Function

Description

This function accepts an incoming connection. After accepting the connection the state of it is as it were created with CreateConnection().

The connection is fully functional and all methods and properties can be called except Transfer().

Note There are two types of incoming connections. One type is the true incoming connection as generated by e.g. the telephone engine. This type of connection can be accepted with Accept().

The other type is the forwarded connection. The forwarded connection is simply an outgoing connection of a different node being forwarded to the local node via the remote engine. This type of incoming connection can only be transferred using Transfer().

Currently the only way to distinguish between these types is to query the connection property “CallerEngine” and to compare for “remote”. Connections with “remote” are forwarded.

The behavior of the connection is undefined if Accept() or Transfer() are applied to connections of the wrong type.

Syntax

```
Accept() as result
```

Return values

This function returns a TCRT result object.

If the function succeeds, the Ok property of the result object is TRUE.

AttPtr2Str Function

Description

This function converts a reference to a TFC. Attachment object into a string representation. If parameter is not TFC. Attachment object or any other error occurs an empty string is returned.

Syntax

```
AttPtr2Str(parameter) as String
```

Parameter	Description
Object	TCRT application object
Parameter	a TFC.Attachment or TFC.Message object

Return value

A string value, which represents a pointer to a TFC. Attachment or TFC. Message object

Close Method

Description

This method closes the connection. Closing the connection means closing all established resource connections and to put it into a passive state. Methods or properties of the connection object and of the connection's player and recorder objects do not work any longer.If there is a Play(), Record() or Start() outstanding, it returns with resOk or resCancelled depending on whether end-of-data was encountered. An outstanding non-blocking call may still generate an event after Close() was called.

The event msgDone appears as very last event and indicates that all connections are now closed and all resources have been freed.

The application may free its resources immediately after Close() or after msgDone. If the application relies on msgEndPlay or msgEndRecord, it should perform its cleanup within the msgDone event.

Syntax

```
Close
```

Connect Function

Description

The purpose of this method is to establish the first real-time half call (half call index 0). For non-active calls as those from Voice Access the first half call is already established when creating the connection object.

For faked connections (active calls) method Connection.Connect() has to be called successfully before any function such as Set(), Play(), Record() or Start() can be called.

Parameter	Description
parameter	a connection string (see the CreateConnection method of the Application object)
asynchronous	TRUE for non blocking connecting

Syntax

```
Connect(parameter, [asynchronous As Long]) As result
```

Return values

This function returns a TCRT result object.

If the function succeeds, the Ok property of the result object is TRUE.

Disconnect Function

Description

This method destroys any half call within a connection. This leaves the connection object in a state where the method Connect() has to be called again before Play(), Set() etc. can be called.

Syntax

```
Function Disconnect() As result
```

Return values

This function returns a TCRT result object.

If the function succeeds, the Ok property of the result object is TRUE.

Flush Method

Description

This method is provided for compatibility. It has no functionality.

Syntax

```
Flush()
```

Play Function

Description

This method establishes a connection to the resource specified as parameter and starts streaming data from this resource to the resource specified at the call to CreateConnection().

For a description of parameter refer to chapter Connection Parameter.

This method can be blocking or non-blocking. By default it blocks execution until all data was transferred or it was cancelled by any other event like Stop() or a transmission error.

For non-blocking operation the parameter bAsync has to be set to True. In this case the method returns immediately after initiating the data transmission and does not wait for the end of data. In non-blocking operation an msgPlayEnd Event is generated if the data transmission automatically stops because the sender encounters an end-of-data condition.

If the data transmission was cancelled by Stop() or Close(), no event will be generated. Stop() returns an error if a data transmission was already finished and an event has been generated.

There may be only one active Play(), Record() or Start(). Starting a new data transmission does not automatically pre-empt any other. It is not possible to both play and record at the same time.

During blocking and non-blocking Play() the methods Seek() and Pause() are allowed and do not cancel playing.

After playing is finished the connection to the resources remain established and can be used by calls to Seek() and Start() to efficiently re-play parts of the current resource.

Calls to Set(), Play() or Record() terminate the connection to the resource previously established with Play() and establish a new connection to the resource specified as parameter to one of those calls.

Syntax

```
Function Play(parameter, [asynchronous As Long]) As result
```

Parameter	Description
parameter	refer to chapter Connection Parameter
asynchronous	TRUE for non blocking play

Return values:

resOk	blocking: data was successfully transferred, non-blocking: data transfer was successfully initiated.
resBusy	There is already a Play() or Record() active.
resCancel	a blocking Play() was cancelled by Stop() and the data was not played to the end.
Other	other error codes may be generated while the connection to the resource gets established or be caused by errors during transmission.

Record Function

Description

This method establishes a connection to the resource specified as parameter and starts streaming data from the resource specified at the call to CreateConnection() to this resource.

For a description of parameter refer to chapter Connection Parameter.

This method can be blocking or non-blocking. By default it blocks execution until all data was transferred or it was cancelled by any other event like Stop() or a transmission error.

For non-blocking operation the parameter bAsync has to be set to True. In this case the method returns immediately after initiating the data transmission and does not wait for the end of data. In non-blocking operation an msgRecordEnd Event is generated if the data transmission stops because the sender encounters an end-of-data condition.

If the data transmission was cancelled by Stop() or Close(), no event will be generated. Stop() returns an error if a data transmission was already finished and an event has been generated.

Events like telephone hang-up or silence-timeouts should be considered as end-of-data condition within the engine generating the data. End-of-data then causes a msgRecordEnd event.

There may be only one active Play(), Record() or Start(). Starting a new data transmission does not automatically pre-empt any other. It is not possible to both play and record at the same time.

During blocking and non-blocking Record() the methods Seek() and Pause() are allowed and do not cancel recording. Though Seek() is allowed it may be silently ignored in this case.

After playing is finished the connection to the resources remain established and can be used by calls to Start() to efficiently append to the current resource.

Calls to Set(), Play() or Record() terminate the connection to the resource previously established with Record() and establish a new connection to the resource specified as parameter to one of those calls.

Syntax

```
Function Record(parameter, [asynchronous As Long]) As result
```

Return values

resOk	blocking: data was successfully transferred, non-blocking: data transfer was successfully initiated.
resBusy	There is already a Play() or Record() active.
resCancel	a blocking Record() was cancelled by Stop() and the sender did not encounter end-of-data.
Other	other error codes may be generated while the connection to the resource gets established or be caused by errors during transmission.

Reject Function

Description

This method rejects incoming connections. This is the same as calling Close(). The event msgDone is generated.

Syntax

```
Function Reject() As result
```

Return values:

resOk	success.
Other	any error occurred while rejecting, the connection is rejected anyway

Set Function

Description

Closes any resource connections previously established by Set(), Play() or Record() and establishes a new resource connection. This method just establishes a connection and does not start the data transfer. Set() does not pre-empt an active data transfer it returns with resBusy in this case.

Refer to chapter Connection Parameter for a description of parameter.

If parameter is invalid, i.e. no connection can be established to the resource specified with this value, the method returns an error and the previous connection remains terminated.

Use this method with an invalid parameter to close resource connections left open after Play(), Record() or Start().

Syntax

```
Function Set(parameter) As result
```

Return values:

resOk	a resource connection was successfully established.
resBusy	There is already a Play() or Record() active.
Other	other error codes may be generated while the connection to the resource gets established. Previous connections remain terminated.

SetCondition Function

Description

This function is provided for compatibility. It has no functionality.

Syntax

```
Function SetCondition(pICondition As condition) As Long
```

Start Function

Description

This method starts a data transfer on already established resource connections. The resource connection must have been established previously with either Set(), Play() or Record(). Start() does not pre-empt an active data transfer it returns with resBusy in this case.

Like with Play() and Record(), Start() can be started in blocking or non-blocking mode. By default the call is blocking. The same rules apply as with Play() or Record().

The parameter position can be used to start data transfer from a specific position. Dependent on the implementation of the hardware engines this parameter may be silently ignored. By now the behavior of position is defined only for the direction of the data flow from a non-real-time resource (“resource” engine) to a real-time resource (“sound” or “telephone” engine).

If position is not specified, Start() uses the position previously set by Seek().

The parameter direction defines the direction of the data flow. A value of 0 defines a data flow from the resource as specified at the call to CreateConnection(), to the resource defined with Set(), Play() or Record(). In fact, the direction 0 is the same as Record().

The value 1 specifies the other direction, in fact, Play().

A value of –1 specifies full duplex operation. The hardware engines or the host engine may not support full duplex operation.

All parameters are optional.

Syntax

```
Function Start([bAsync As Long], [position As Long = -1], [direction As Long = 1]) As
 result
```

Return values:

resOk	blocking: data was successfully transferred, non-blocking: data transfer was successfully initiated.
resBusy	There is already a Play() or Record() active.
resCancel	a blocking Start() was cancelled by Stop() and the sender did not encounter end-of-data.
Other	other error codes may be generated by errors during transmission or during transmission initiation.

Stop Function

Description

Cancels a currently active data transfer. If the data transfer was blocking, the blocking call returns with error code resCancelled.

Stop() suppresses msgEndPlay or msgEndRecord events if the data transfer was interrupted before the data source encountered end-of-data.

Stop() does not close any resource connections.

Syntax

```
Function Stop() As result
```

Return values:

resOk	an active data transmission was interrupted before end-of-data condition.
resErr	there was no data transmission active

TestEvent Method

Description

Manually generates an event.

Syntax

```
Sub TestEvent(EventNum As Long, EventDescr As String, result As Long)
```

Timer Function

Description

On success the method returns a unique, non-zero identifier. This identifier will be included with msgTimeout as correlation information.

msTimeout sets the period in milliseconds. The value 0 stops the timer.

Setting Mode to 1 activates a periodic timer. Only a single timer message will be generated when setting Mode to 0 value.

Syntax

```
Function Timer(msTime As Long, [Mode As Long]) As Long
```

Transfer Function

Description

This method is used to transfer incoming connections to any other resource. Transferring means the application calling transfer gives away the control over this connection and directly connects the incoming call to a other resource. After transferring the application does not get events other than msgDone. The application should not call any other methods or properties.

If the transfer fails, the incoming connection must be transferred to any other resource or rejected with Reject().

Parameter is described in chapter Connection Parameter.

Syntax

```
Function Transfer([parameter]) As result
```

Return values:

resOk	Transfer was successful.
Other	Transfer failed

WaitForAction Function

Description

This function is provided for compatibility. It has no functionality.

Syntax

```
Function WaitForAction([timeout As Long = -1], [asynchronous As Long]) As result
```

Event

Description

TCRT passes events to every handler in the event handler chain until one successfully handles it. Successfully handling means:

1. The event handler exists
2. The event handler returns 0 (resOk).

By default the event handler return value is resOk. Thus once the event was passed to a handler function it is not automatically passed to another. To forward a specific event to other handlers, the handler function must return a non-zero value (e.g. resErr).

The handler chain is walked in the following order:

1. 1. Connection event handler
2. 2. Application event handler
3. 3. Other Application event handlers
4. 4. Default handler

Note For incoming calls, there is never a connection event handler. In this case always the application handlers are called.

If there are multiple application event handlers only one is called by default. It is not defined which application handler is the first to be called.

Usually there are no more than one application objects. Only in the case of multiple in-process COM servers (e.g. Player.OCX) each instantiating the application object independently from each other, there may be more than one application objects (in fact, more than one references to the same object). The default behavior prevents from calling the same handler (one for each application object) multiple times. If the event is to be passed to other handlers, this has to be indicated with a non-zero return value.

Except for incoming calls (msgInit), the default handler has no function. In the case of an incoming call the default handler rejects the call and closes the connection.

Event handlers are defined as shown:

```
Dim WithEvents Connection AS TCRT.Connection
Function Connection_Event ( ByVal EventNum as Long, ByVal EventDesc as Variant, ByVal
 Conn As Connection, ByRef Result as Variant) as Long
```

With the current version EventDesc is used only for the msgDTMF event and for msgDiscInd with an asynch. call .

The parameter Result has no function in the current release of TCRT. For the first event handler it is initialized to a Long value of 0. The event handler may change this parameter and pass the new value back. The new value is passed to the next event handler. TCRT does not interpret this value.

Note It is not allowed to call Play(), Record() or Start() in blocking mode from within event handlers. This will lead to a deadlock!

Syntax

```
Event Event(EventNum As Long, EventDescr, connection As connection, result)
```

Container

Description

A container is an abstraction for storage space. You can record to and play from a container. Following items can be added to a container:

- String
- TFC.Attachment
- TFC.Text
- TFC.Message

More information about TFC objects can be found in the *TFC Reference Manual*.

Count Property

Syntax

```
Object.Count As Long
read only
```

Description

This property returns the number of items in the container.

Item Property

Description

This property returns a specific member of a Container object by position.

Syntax

```
Object.Item([Index as Variant])  [=VarObject]
```

Add Method

Description

This method adds a member to a container object.

Syntax

```
Object.Add(Value as Variant)
```

Insert Method

Description

This method inserts a member to a container object at position Index.

Syntax

```
Object.Insert(Index as Variant, Value as Variant)
```

Remove Method

Description

This method removes a member from a container object at position Index.

Syntax

```
Object.Remove(Index as Variant)
```

Condition

Not supported

Note From Kofax Communication Server 10.3.0, the type name of the interface to this class has been changed from ICondition to ITfcCondition.

Player

This section describes the player functions.

Pause Function

Description

Pauses the current data transfer. Unlike Stop() the data transfer is only halted for some time and not cancelled. Blocking Play(), Record() or Start() methods are not interrupted and do not return.

In the paused state no new Play(), Record(), Set() or Start() calls are possible.

Data transfer can be continued with Pause(False) or cancelled with Stop(). Pausing the data transfer modifies the internal seek position.

Currently the behavior of a Pause() for data transfers other than a Play() from a non-real-time engine (“resource” engine) to a real-time engine (“telephone” “sound” engine) is not defined. In any other case, the engines may start anew.

Syntax

```
Function Pause(yes As Long) As result
```

Return values:

resOk	success
Other	any error occurred while pausing

Play Function

Description

Same as Connection.Play()

Syntax

```
Function Play(parameter, [asynchronous As Long]) As result
```

PlayAttachment Function

Description

Same as Play() but accepts only a TFC.Attachment object as parameter.

Syntax

```
Function PlayAttachment(attachment, [bAsync As Long]) As result
```

PlayBeep Function

Description

Same as Play() but plays only a single tone.

Syntax

```
Function PlayBeep([frequency As Long = -1], [duration As Long = -1], [bAsync As Long])
 As result
```

PlayDTMF Function

Description

This function plays DTMF tones at the permanent connection. This function may not be implemented at the engine level.

Syntax

```
Function PlayDTMF(tones As String, [toneDuration As Long = -1], [gabDuration As Long =
 -1]) As result
```

PlayFile Function

Description

Same as Play() but accepts only a valid filename as parameter

Syntax

```
Function PlayFile(filename As String, [bAsync As Long]) As result
```

PlayPrompt Function

Description

Same as Play() but accepts only a text string as parameter which is interpreted as prompt. Refer to the TCE_RES.DLL description for details on the prompt syntax.

Syntax

```
Function PlayPrompt(prompt As String, param As String, [bAsync As Long]) As result
```

PlayText Function

Description

Same as Play() but accepts only a text string as parameter which is converted to speech.

Syntax

```
Function PlayText(text As String, [bAsync As Long]) As result
```

Seek Function

Description

This function sets the new position for the data transfer. If there is a data transfer currently active, Seek() sets the engines to the new positions and continues data transfer. If no data transfer is active the method stores the new position but returns resWarn. A successive call to Start() uses the current seek position. The current seek position is reset to 0 on calls to Stop() and Set(). Pause() changes the seek position to the position where the data transfer was paused.

Currently the behavior of a Seek() for data transfers other than a Play() from a non-real-time engine (“resource” engine) to a real-time engine (“telephone” “sound” engine) is not defined. In any other case, the engines may silently ignore Seek() or even start anew.

The parameter units is optional. The default value is unitsMSec which is the only valid value.

The parameter relative is optional. It specifies if the distance is a value relative to the current position or if it is an absolute position. The default value is True (relative distance).

Syntax

```
Function Seek(distance As Long, [units As Long = -1], [relative As Long = -1]) As
 result
```

Return values:

resOk	success, repositioned active data transfer
resWarn	success, no active data transfer but internal seek position changed.
Other	any error occurred while seeking

Set Function

Description

Same as Connection.Set()

Syntax

```
Function Set(parameter) As result
```

Start Function

Description

Same as Connection.Start() but with limited parameter list. The direction is always 1 (play)

Syntax

```
Function Start(bAsync As Long) As result
```

Stop Function

Description

Same as Connection.Stop()

Syntax

```
Function Stop() As result
```

Recorder

This section describes the **Recorder** functions.

Record Function

Description

Same as Connection.Record()

Syntax

```
Function Record(parameter, [asynchronous As Long]) As result
```

Stop Function

Description

Same as Connection.Stop()

Syntax

```
Function Stop() As result
```

Result

Description

TCRT methods that are not constructors (.*CreateConnection()*, …) return an object of this class.

The Result class may represent function results (success, error, warning, …) or result tokens.

See .WaitForAction() on how to use the latter.

Use the provided properties to access the method's result.

Long Property

Description

This property contains the numeric representation of the result. It contains the result token only if .Error is false.

Syntax

```
R/O: Property Long As Long
```

Ok Property

Description

True if this result is neither an error nor a warning.

Syntax

```
R/O: Property Ok As Boolean
```

Error Property

Description

True if this result is an error.

Syntax

```
R/O: Property Error As Boolean
```

Warning Property

Description

True if this result is a warning.

Syntax

```
R/O: Property Warning As Boolean
```

Started Property

Description

Check this flag if your use asynchronous calls.

True if the call that returned the Result was started. You can tell if the call was successful, when the .Ok, .Error, or .Warning property becomes true. Or the termination event arrives.

Syntax

```
R/O: Property Started As Boolean
```

String Property

Description

This property contains the string representation of the result.

Syntax

```
R/O: Property String As String
```

Details Property

Description

May contain additional descriptions for errors and warnings. Contains the condition that matched in case of *.WaitForAction()*, *.Play()*, *.Record()*, …

Syntax

```
R/O: Property Details As String
```

Events

COM Interface	Value
MsgDiscInd	0x80000002
MsgInit	0x80000003
MsgDone	0x80000004
MsgDTMF	0x80000012
MsgPlayEnd	0x80000019
MsgRecordEnd	0x8000001A
MsgTimeout	0x8000001F
msgStatus	0x80000020

msgInit

Indicates a new incoming connection. This event is the first event of the new connection. It is not generated if the connection is created with *Application.CreateConnection()*.

The incoming connection has to be accepted with *Connection.Accept()*, transferred with *Connection.Transfer()* or rejected with *Connection.Reject()*. If a connection can be accepted or transferred depends on the type of connection. Forwarded connections can only be transferred while all other connections can only be accepted. Forwarded connections always have a the property “CallerEngine” set to “remote”.

Incoming connections should always be accepted, transferred or rejected within this event. It is not allowed to let incoming connections pending.

The application should keep a reference to the connection object. Otherwise the object will be destroyed and the connection closed after the event handler is left.

msgDiscInd

This event indicates that one side of the connection went passive. This happens when for example the user hangs up the telephone.

If in this case the application is recording from the telephone, the data will be fully transferred to the receiver and the pending *Record()* call properly stops (with the corresponding event if non-blocking).

If in this case the application is playing to the telephone, the data can not be fully played and the transfer is cancelled. If a blocking *Play()* was active, it stops with *resCancelled*.

If the call was non-blocking, there is currently no notification event.

Future releases may accept new *Play()*, *Record()* and *Set()* commands to release the passive side of the connection and re-establish a new resource connection. The current behavior is undefined if one of this methods is called.

Currently the only possible reaction to this event is to close the connection with *Close()*.

Remarks: If any error occurs during asynchronous call establishment (like “busy”), the telephone engine sends immediately the MSG_DISC_IND with new error value RT_INT_ISDN_ERROR set to the ISDN cause value for this error (for a full list of ISDN cause values please see TCOSS System manual).

This error code is delivered as event description to the application.

msgDone

This is the very last event generated by a connection. It is caused by a call to *Connection.Close()*.

The application should perform its cleanup in this event. Any reference to the connection object should be freed after this event.

msgPlayEnd

This event is generated when a non-blocking call to *Play()* or *Start(direction=1)* automatically ends by a end-of-data condition. If the play was cancelled by *Stop()*, this event does not occur.

msgRecordEnd

This event is generated when a non-blocking call to *Recor()* or *Start(direction=0)* automatically ends by a end-of-data condition. If the play was cancelled by *Stop()*, this event does not occur.

msgDTMF

A telephone engine generates this event whenever the user presses a button. This event is not dependent on *Play()*, *Record()* or other TCRT calls. It may appear any time.

msgTimeout

An application can return a negative value in it's Main function to define the timeout value. If no other event occurs in this time period (e.g. a DTMF event), the event msgTimeout is sent to the `Main` function of the application.

msgStatus

In order to inform the calling application on the call establishment progress, new message MSG_STATUS has been defined.

This message is sent by the telephone engine as soon as call progress status changes.

The call progress is indicated in the RT_INT_STATUS_VALUE parameter of the MSG_STATUS message. This parameter is delivered as event description to the application.

Following call progress parameter values are defined.

statConnecting = 0 :	currently not used
statDialling = 1 :	telephone engine has just initiated call establishment (CALL_PROC received)
statAlerting = 2 :	telephone engine has received alerting indication (ringing)
statConnected = 3 :	the call is connected
statConnReqTimeout = 4 :	the call request timeout expired

As soon as the application receives the MSG_STATUS(statConnected), the call is connected.

Chapter 3

Examples

This section describes examples.

VBScript

Requirements to run the scripts

- Windows Script Host 5.6
- Visual Basic® Script Edition (VBScript.) Version 5.6
- TC/CP 5.07.04 with TCPlayer installed
- TFC.DLL for the sample script that uses the TFC attachment object

TTS Speakers on the Voice Server

```
Rem With this sample script you can hear the TTS speakers of the voice server
Rem This sample script runs with the CScript.exe host executable file only !!!
Const TXT_TTSSPEAKERGUID = "TTSSpeakerGUID"
Const TXT_TTSPITCH = "TTSPitch":
Const TXT_TTSRATE = "TTSRate":
Const TXT_TTSVOLUME = "TTSVolume":
Class CSpeaker
...Dim mGuid
...Dim mPitch
...Dim mSpeed
...Dim mName
End Class
DIM tcrtApp
Dim vServer
Dim lSpeakers()
Set tcrtApp = CreateObject("TCRT.Application")
vServer = "DEMOVOICE"
If not tcrtApp is nothing then
...doMenu
Else
...WScript.Echo "Cannot create TCRT Application"
End If
Set tcrtApp = Nothing
Quit
Sub doMenu
Dim tcrtConn
Dim szConnStr
Dim vSpeakers
Dim i
Dim iSpeakerIDX
Dim cKey, cNumber
Dim vTmp
WScript.Echo "-> doMenu"
```

```
WScript.Echo "get installed TTS speakers from the voice server"
szConnStr = "resource@" & CStr(vServer) & ":TTSProperties"
Set tcrtConn = tcrtApp.CreateConnection(szConnStr)
If not tcrtConn is Nothing Then
...vSpeakers = tcrtConn.Property("TTSSpeakerList")
...WScript.Echo "closing Connection"
...tcrtConn.Close
...Set tcrtConn = Nothing
...iSpeakerIDX=-1
...If ParseSpeakerList(vSpeakers) > 0 Then
......Do
.........WScript.Echo Chr(10) & Chr(13) &
 "======================================================"
.........For i=0 to UBound(lSpeakers)
............If iSpeakerIDX = i Then
..............vTmp = "(selected)"
............Else
..............vTmp = ""
............End If
............WScript.Echo CStr(i) & " - " & lSpeakers(i).mName & " " & vTmp
.........Next
.........WScript.Echo Chr(13) & Chr(10)
.........WScript.Echo "V - Voice server name " & vServer
.........WScript.Echo "N - Telephone numer (" & cNumber & ")"
.........WScript.Echo "C - Call"
.........WScript.Echo "X - Exit"
.........WScript.Echo Chr(13) & Chr(10)
.........WScript.Echo "Enter command and press the ENTER key"
.........WScript.Echo "======================================================"
.........WScript.Echo Chr(13) & Chr(10)
.........cKey=GetInput()
.........Select Case CStr(cKey)
............Case "v","V"
..............vServer = GetInput
............Case "x","X"
..............Exit Do
............Case "0" , "1", "2", "3", "4", "5", "6", "7", "8", "9"
..............i = CInt(cKey)
..............If i > UBound(lSpeakers) Then
................WScript.Echo "invalid speaker index"
..............Else
................iSpeakerIDX = i
..............End If
................
............Case "n","N"
..............WScript.Echo "enter telephone number:"
..............cNumber = GetInput()
............Case "c","C"...
..............If iSpeakerIDX < 0 Then
................WScript.Echo "You must specify a TTS speaker"
..............End If
..............If Len(cNumber) = 0 Then
................WScript.Echo "You must specify a telephone number"
..............End If
..............If (Len(cNumber) > 0) And (iSpeakerIDX>=0) Then
................WScript.Echo "Number = " & cNumber
................DoCall cNumber,iSpeakerIDX
..............End If
............Case Else
..............WScript.Echo "Invalid command"
.........End Select
......Loop
...End If
...Err.Raise 5
```

```
...Quit
...
Else
...WScript.Echo "CreateConnection(" & szConnStr & "failed"
End If
WScript.Echo "<- doMenu"
End Sub
Sub DoCall(vNumber,vIndex)
...Rem this method does the telephone call and plays the TTS text
...Dim szConnStr,tcrtConn
...Dim TTSGuid, TTSTxt,TTSPitch,TTSSpeed,TTSParam,TTSVolume
...Dim vResult
...WScript.Echo "Index = " & CStr(vIndex)
...szConnStr = "telephone@" & vServer & ":" & cstr(vNumber)
...WScript.Echo "creating connection to " & szConnStr
...Set tcrtConn = tcrtApp.CreateConnection(CStr(szConnStr))
...If not tcrtConn is Nothing Then
......TTSGuid = "<" & TXT_TTSSPEAKERGUID & " """ & lSpeakers(CInt(vIndex)).mGuid &
 """>"
......TTSTxt = "<TTS ""This is a test message for text to speech.""">"
......TTSPitch = "<" & TXT_TTSPITCH & " """ & CStr(lSpeakers(CInt(vIndex)).mPitch) &
 """ > "
       TTSSpeed = "<" & TXT_TTSRATE & " """ & CStr(lSpeakers(CInt(vIndex)).mSpeed) &
 """ > "
       TTSVolume = "<" & TXT_TTSVOLUME & " ""100""">"
       TTSParam = CStr("resource@" & vServer & ":" & TTSGuid & TTSPitch & TTSSpeed  &
 TTSVolume & TTSTxt)
         ...WScript.Echo TTSParam
......Set vResult = tcrtConn.Play(TTSParam)
......If vResult.OK = 1 Then
.........WScript.Echo "success Result=" & vResult.String
......Else
.........WScript.Echo "failed.Error=" & vResult.String
......End If
...End If
End Sub
Function GetInput
...Rem This function gets input from the keyboard
...Dim cInput,cKey
...cKey=""
...Do
......cInput = WScript.StdIn.Read(1)
......If  Chr(10) =  cInput Then
.........Exit Do
......Else
.........If (Asc(cInput) > 31) Then
............cKey = cKey & cInput
.........End If
......End If
...Loop
...GetInput = cKey
End Function
Function ParseSpeakerList(szSpeakerList)
...Rem this function parses the string szSpeakerList to
...Rem an array that contains CSpeaker elements
...Dim iResult,i,vTmp
...WScript.Echo "->ParseSpeakerList"
...WScript.Echo  szSpeakerList
...arSpeakers = Split(szSpeakerList,";")
...iResult = UBound(arSpeakers)
...If iResult > 0 Then
......ParseSpeakerList = iResult
......For i=0 to iResult
.........WScript.Echo "Speaker:" & arSpeakers(i)
```

```
.........vTmp = Split(arSpeakers(i),",")
.........If UBound(vTmp) > 0 Then
............ReDim Preserve lSpeakers(iResult)
............Set lSpeakers(i) = new CSpeaker
............lSpeakers(i).mGuid = vTmp(0)
............If InStr(lSpeakers(i).mGuid ,"=") > 0 Then
..............lSpeakers(i).mGuid = Left(lSpeakers(i).mGuid ,
 InStr(lSpeakers(i).mGuid ,"=")-1)
............End If
............lSpeakers(i).mPitch =  vTmp(1)
............lSpeakers(i).mSpeed = 100
............lSpeakers(i).mName = vTmp(4)
.........End If
......Next
...Else
......ParseSpeakerList = -1
...End If
...WScript.Echo "<-ParseSpeakerList"
End Function
```

Play a Sound File via the Telephone

```
Rem This sample script play a sound file via the telephone
Rem This sample script runs with the CScript.exe host executable file only !!!
Dim tcrtApp
Dim tcrtConn
Dim vNumber
Dim vFileName
Dim vVoiceServer
Dim vResult
vVoiceServer = "DEMOVOICE"
WScript.Echo "Enter telephone number "
vNumber = GetInput()
WScript.Echo "Enter file name "
vFileName = GetInput
Set tcrtApp = CreateObject("TCRT.Application")
If not tcrtApp is Nothing then
...Set tcrtConn = tcrtApp.CreateConnection("telephone@" & vVoiceServer & ":" & vNumber)
...If not tcrtConn is Nothing Then
......vResult = tcrtConn.Play("<File """ & vFileName & """>")
...End If
...tcrtConn.Close
End If
Set tcrtConn = Nothing
Set tcrtApp = Nothing
Function GetInput
Dim cInput,cKey
cKey=""
Do
...cInput = WScript.StdIn.Read(1)
...If  Chr(10) =  cInput Then
......Exit Do
...Else
......If (Asc(cInput) > 31) Then
.........cKey = cKey & cInput
......End If
...End If
Loop
GetInput = cKey
End Function
```

Record to a File via the Telephone

```
Rem This sample script records from the telephone to file
Rem This sample script runs with the CScript.exe host executable file only !!!
Dim tfcApp
Dim tcrtApp
Dim tcrtConn
Dim vNumber
Dim vFileName
Dim vAttachment
Dim vVoiceServer
Dim vResult
vVoiceServer = "DEMOVOICE"
Set tfcApp = CreateObject("TFC.Application")
Set tcrtApp = CreateObject("TCRT.Application")
If not tcrtApp is Nothing Then
...Set vAttachment = tfcApp.CreateAttachment
...WScript.Echo "Enter telephone number"
...vNumber = GetInput()
...If Len(CStr(vNumber)) > 0 Then
......WScript.Echo "Enter file name"
......vFileName = GetInput()
......If Len(CStr(vFileName)) > 0 Then
.........Set tcrtConn = tcrtApp.CreateConnection("telephone@" & vVoiceServer & ":" &
 vNumber)
.........If not tcrtConn is Nothing Then
............WScript.Echo "Recording ..."
............WScript.Echo "Hang up the phone to stop recording"
............Set vResult = tcrtConn.Record("<File """ & vFileName & """>")...
............If vResult.OK  Then
..............WScript.Echo "Recording has stopped"
............End If
.........End If
......End If
...End If
End if
Set tcrtConn = Nothing
Set tcrtApp = Nothing
Set vAttachment = Nothing
Set tfcApp = Nothing
Function GetInput
Dim cInput,cKey
cKey=""
Do
...cInput = WScript.StdIn.Read(1)
...If  Chr(10) =  cInput Then
......Exit Do
...Else
......If (Asc(cInput) > 31) Then
.........cKey = cKey & cInput
......End If
...End If
Loop
GetInput = cKey
End Function
```

Record to a File via the Telephone Using a TFC Attachment

```
Rem This sample script records from the telephone to a TFC attachment
Rem When recording has stopped, the attachment can be saved to disk
Rem This sample script requires TCRT.DLL and TFC.DLL
```

```
Rem This sample script runs with the CScript.exe host executable file only !!!
Dim tfcApp
Dim tcrtApp
Dim tcrtConn
Dim vNumber
Dim vFileName
Dim vAttachment
Dim vVoiceServer
Dim vResult
vVoiceServer = "DEMOVOICE"
Set tfcApp = CreateObject("TFC.Application")
Set tcrtApp = CreateObject("TCRT.Application")
If not tcrtApp is Nothing Then
...Set vAttachment = tfcApp.CreateAttachment
...WScript.Echo "Enter telephone number"
...vNumber = GetInput()
...Set tcrtConn = tcrtApp.CreateConnection("telephone@" & vVoiceServer & ":" & vNumber)
...If not tcrtConn is Nothing Then
......WScript.Echo "Recording ..."
......WScript.Echo "Hang up the phone to stop recording"
......Set vResult = tcrtConn.Record(vAttachment)
......If vResult.OK  Then
.........WScript.Echo "Recording has stopped. Enter file name to save"
.........vFileName = GetInput()
.........If Len(CStr(vFileName)) > 0 Then
............If vAttachment.Detach(CStr(vFileName)) = True Then
..............WScript.Echo vFileName & " saved"
............Else
..............WScript.Echo "Saving of " & vFileName & " failed"
............End If
.........End If
......End If
...End If
End if
Set tcrtConn = Nothing
Set tcrtApp = Nothing
Set vAttachment = Nothing
Set tfcApp = Nothing
Function GetInput
Dim cInput,cKey
cKey=""
Do
...cInput = WScript.StdIn.Read(1)
...If  Chr(10) =  cInput Then
......Exit Do
...Else
......If (Asc(cInput) > 31) Then
.........cKey = cKey & cInput
......End If
...End If
Loop
GetInput = cKey
End Function
```

Chapter 4

TCP Ports

The real-time object consists of a real-time core and a number of plug-ins, called engines. The core is hardware independent. Its main purpose is to provide the basic COM interfaces to the application and to provide a routing mechanism to connect the different data sources and sinks. The plugged-in engines are the actual data sources and sinks. They are hardware dependent. The hardware available on the local system determines if an engine is loaded or not. In addition, loading of specific engines may be dependent on the type of application.

The remote engine is responsible for connecting different instances of applications. The remote engine has knowledge of all other remote engines in the network. Additionally the remote engine knows all engines plugged into the local real-time core. The information about the local engines can be retrieved by all other remote engines to learn about the resources available in the system.

If an application connects to a resource not locally available, without specifying the target server, the remote engine automatically takes the choice and connects to a different remote engine with the appropriate resources.

An application should have at least the remote engine plugged into the real-time core. Otherwise, it is invisible to other parts of the system.

Port 135 – RPC Locator Service

The remote engine uses the Windows NT browser service for RPC (RPCLocator) to collect information about all other machines within the network containing TC/ECP applications.

Service	TCP	UDP	Notes
RPC endpoint mapper	135	135	* registered as "epmap - DCE endpoint resolution". Used by Microsoft for RPC locator service.

TCP Endpoints

A TCP endpoint is a combination of the IP address and the TCP port used. The remote engine dynamically selects endpoints for the RPC server within every node (a node is one instance of a remote engine - i.e. one single instance of an application using the remote engine). The range of endpoints used can be specified in the registry (for details see the Remote Engine Manual). If no range of endpoints is configured, then the system chooses the endpoint.