Using COM to Automate IndiCom

Table of Contents

1 Introduction 1
2 Using the RemoteInterface 2
2.1 Preconditions 2
2.2 Using Functions and Events 2
2.3 Using the RemoteInterface with DCOM 3
2.4 RemoteInterface 4
2.4.1 Functions 4
2.4.2 Properties 5
2.5 RemoteInterface2: 6
2.5.1 Functions 6
2.5.2 Properties 6
2.6 Events 7
2.7 Keys for SetParameter and GetParameter 8
3 IndiCom Remote Errors: 8
4 Changes from IndiCom 1.0 to IndiCom 1.1 and 1.2 9

1 Introduction

IndiCom (from Version 1.1 onwards) is able to act as OLE Automation server. It exposes COM interfaces, which may be used to remote control the software and thus the whole indicating system.
If the controlling test bed is also COM enabled (like AVL PUMA Open), it may directly communicate to IndiCom via DCOM (distributed COM), which is based on a standard TCPIP network connection. For this purpose DCOM must be installed and activated on the IndiCom and the testbed PC.
If the testbed system does not use COM, but prefers other connections e.g. a serial line (like PUMA 5), a remote control program has to be implemented which runs on the IndiCom PC and works as a translator. On the one hand this program has to react to the commands of the testbed system being sent via the specific communication line. On the other hand this task has to remote control IndiCom via the COM Interface described hereafter.
This way IndiCom is completely independent from the type and protocol used for remote control. It only knows its own standardised remote control interface.

2 Using the RemoteInterface

2.1 Preconditions

To be able to use the IndiCom RemoteInterface, the option ‚R' has to be present. If this option is missing, the call to *StartRemote* will fail and all other function will send an *Error* event.

Most of the functions are non-blocking, that means, that the function call will return immediately before the action triggered has finished (e.g. *SaveFile*) . Thus the client has to wait until the correct answer event is fired, which signals the end of the function execution. The call was successful when the correct event (e.g. *SaveFileDone*) is fired or unsuccessful, when the *Error* event id fired.

When using VB to automate IndiCom , the project references have to include “IndiCom Automation”, which uses IndiCom.tlb.

2.2 Using Functions and Events

- The very first function a client calls has to be *StartSession*.
- The very last function has to be *EndSession*. Do not forget to set all Interfaces to Nothing. Otherwise IndiCom cannot shut down and stays in memory.
- *StartMeasurement* and *StartContinuousMeasurement* have to wait for the events *MeasurementStarted* OR *Error*.
- *StopMeasurement* has to wait for the event *MeasurementStopped* OR *Error*. After the event MeasurementStopped ALWAYS occur the *FinalCalculationsDone* or *Error* events, even, when no measurement was performed. Between these two events IndiCom performs a calculation of data and in the mean time no other COM call will be processed. For example the function *IsAlive* will not return until *FinalCalculationsDone* returned. For interrupting the calculation see below.
- When a single measurement stoppes normally, also the events *MeasurementStopped* followed by *FinalCalculationsDone* occure. Between these two events IndiCom performs a calculation of data and in the mean time no other COM call will be processed. For example the function *IsAlive* will not return until *FinalCalculationsDone* returned. For interrupting the calculation see below.
- *Break* has to wait for the events *BreakDone* OR *Error*.
- *LoadParFile* has to wait for the events *LoadParFileDone* OR *Error*.
- *SetParameter* has to wait for the events *SetParameterDone* OR *Error*.
- *Recalculate* has to wait for the events *RecalculateDone* OR *Error*. While waiting for these events, no other COM call will be processed. For example the function *IsAlive* will not return until *RecalculateDone* OR *Error* returned. For interrupting the calculation see below.
- *AverageData* has to wait for the events *AverageDataDone* OR *Error*.

- *SaveFile* has to wait for the events *SaveFileDone* OR *Error*.

- On the event *RemoteOff* (indicates that IndiCom has taken control), *StartRemote* has to be sent before any other function can be called again.

- On the event ShutDown (IndiCom shuts down), the client has to call *EndRemote*, *EndSession* and has to release its Interfaces to IndiCom (set it to nothing). Otherwise IndiCom cannot shut down and stays in memory.

- ExecuteScript has to wait for the events ExecuteScriptDone or Error. For interrupting the script see below.

- Recalculate2ndStep has to wait for the events Recalculate2ndStep or Error. For interrupting the calculation see below.

- To abort a running calculation or script the IndiCom window handle can be received by a call to *GetWindowHandle* (before the calculation was started). The message number 33662 can be posted. On this message IndiCom will break any running calculation or script. Nevertheless the correct events on previous commands will be sent afterwards. (e.g. *FinalCalculationDone*)

2.3 Using the RemoteInterface with DCOM

To be able to use the IndiCom RemoteInterface via DCOM, the option ‚R‘ and the Option ‘X[D]’ have to be present.

When using VB to automate IndiCom via DCOM across the network, the project references have to include “IndiComRemote Type Library”, which uses IndiComRemote.exe.

Use CreateObject(“IndiComRemote.RemoteInterface”, “SERVER”) to access the IndiCom on a PC in the network, whereas “SERVER” is the network name or IP Address of the IndiCom PC, you want to access.

You may the use the IndiComRemote Interface exactly like IndiCom RemoteInterface as described above.

As an alternative you may use the program dcomcnfg.exe to change the location of the Application “IndiComRemote” to the network address of the computer IndiCom is running on:

Open dcomcnfg.exe, find IndiComRemote in the list, click on properties, open the location sheet, clear “Run application on this computer”, check “Run application on the following computer”, and enter the network address of the computer IndiCom is running on.

Exception:
- To abort a running calculation or script, use the function *StopCalculation*. On this command IndiCom will break any running calculation or script. Nevertheless the correct events on previous commands will be sent afterwards. (e.g. *FinalCalculationDone*)

2.4 RemoteInterface

2.4.1 Functions

void StartSession()
Initializes the remote control. This always has to be the very first command.

void EndSession()
Ends the remote control. This always has to be the very last command.

void StartRemote()
Starts the remote session. The text 'REMOTE' will appear in the lower right corner of IndiCom. This function has to be successfully called BEFORE any other function is used. It will fail, if the option 'R' is missing.

void EndRemote()
Ends the remote session.

void LockUserInterface()
Locks the user interface for measurement.

void UnlockUserInterface()
Unlocks the user Interface for measurement

void LoadParFile(String strFilePath)
Loads an IndiCom Parameterfile. If strFilePath is only a filename and not a complete path, the filename will be extended by the directory specified by the entry PumaParPath in the IndiCom.ini.
CAUTION: IndiCom will not perform any other functions (like IsAlive) until LoadParFileDone returned.

void SetParameter(String strKey, String strContent)
Sets a parameter specified by strKey to the value specified by strContent. Numbers have to be converted to strings

String GetParameter(String strKey)
Delivers the parameter specified by strKey. For a complete list if keys see below.

void StartMeasurement()
Starts a single measurement.

void StartContinuousMeasurement()
Starts a continuous measurement.

void StopMeasurement()
Stops a running measurement.

void Break()
Sends a break. This command performs a system reset if necessary.

void Recalculate()
Recalculates all formulae.
CAUTION: IndiCom will not perform any other functions (like IsAlive) until RecalulateDone returned.

void AverageData()
Averages all cycles.

void SaveFile(String strPath, short dataMask)
Saves measurement data. If strPath is only a filename and not a complete path, the filename will be extended by the directory specified by the first entry of PumaDatPath in the IndiCom.ini.
if strPath is empty, the saving presets in IndiCom will be used to save the file in two different locations. (see the IndiCom manual).
The value dataMask specifies what kind of data shall be saved. Possible values:
1 = CA, 2 = TIM, 4 = RTP, 8 = UTC. If more than one datatype shall be saved, the values have to be added. e.g. save CA and RTP -> dataMask = 1 + 4 = 5

String GetLastFilePath()
Delivers the path of the last successfully saved file.

String GetHighestFileName(String strFilePath)
Delivers the filename with the highest extension that exits in the directory of the specified path.

long IsInitialized()
Can be used for determining, if IndiCom started up successfully. The function delivers 0, if the startup process is not complete and 1, if the startup was successful.

void IsAlive()
This function can be used to determine periodically, if IndiCom is still running. In an error occurs IndiCom is not running anymore. In this case the client should shut down.
CAUTION: While IndiCom is calculating formulae or loading a parameter file, this call will have to wait until calculation or loading is finished. In the meantime, the client will be blocked.

String GetDataDir(long nr)
Delivers the either the main directory or the alternate directory specified in the saving presets in IndiCom. If nr equals 0 or 1, the function delivers the main directory, if nr is greater than 1, the function delivers the alternate directory.

String GetParameterDir(long nr)
Delivers the directory specified by the ini-entry PumaParPath and nr. It is possible to add multiple parameter directories seperated by ‘;’
e.g: PumaParPath = c:\; c:\test

2.4.2 Properties

VARIANT ResultNamesDuringMeasurement
This variant contains an array with the names of the datasets defined to be transmitted during a measurement.

VARIANT ResultValuesDuringMeasurement
This variant contains an array with the calculated values of the defined datasets to be transmitted during a measurement.

VARIANT ResultNamesAfterMeasurement
This variant contains an array with the names of the datasets defined to be transmitted after a measurement.

VARIANT ResultValuesAfterMeasurement
This variant contains an array with the calculated values of the defined datasets to be transmitted after a measurement. These values are calculated immediately after every measurement.

2.5 RemoteInterface2:

2.5.1 Functions

void ExecuteScript (String path)
Executes a script specified by the parameter *path.* If the first character of *path* is a '>', the following text will be interpreted as a direct script call. e.g. ExecuteScript("d:\test\test.csf") ExecuteScript(">PrintWindow()")

void GetVersion (long* plMainVers, long* plSubVers, long* plBuildNr, long* plMachineBuildNr)
Delivers the actual version if IndiCom.

String GetOptionString ()
Delivers the actual options.

void ShowIndiCom (long *show***)**
Shows (show = 1) or hides (show = 0) IndiCom.

long GetWindowHandle ()
Delivers the window handle of IndiCom as a long value. This value can be used to send window messages to IndiCom.

void Recalculate2ndStep ()
Forces a recalculation or the 2nd Step result output values.

void StopCalculation ()
Stops any running calculation. This function will only work if used together with IndiComRemote.exe (DCOM).

2.5.2 Properties

VARIANT ResultNames2ndStep
This variant contains an array with the names of the datasets defined to be transmitted after meas 2.

VARIANT ResultValues2ndStep
This variant contains an array with the names of the datasets defined to be transmitted after meas 2. These values are calculated every time when *Recalculate2ndStep* is called.

VARIANT ResultUnitsDuringMeas
This variant contains an array with the units of the datasets defined to be transmitted during a measurement.

VARIANT ResultUnitsAfterMeas
This variant contains an array with the units of the datasets defined to be transmitted after a measurement.

VARIANT ResultUnits2ndStep
This variant contains an array with the units of the datasets defined to be transmitted during a after meas 2.

2.6 Events

void MeasurementStarted()
Indicates that a measurement or a continuous measurement has been successfully started.

void MeasurementRunning()
Indicates that a measurement or a continuous measurement is running. This event occurs each time that IndiCom has finished painting a new picture.

void MeasurementStopped()
Indicates that a measurement or a continuous measurement has stopped.

void FinalCalculationDone()
Indicates that the final calculations after a measurement have been calculated. This event ALWAYS occurs after the event *MeasurementStopped*

void SaveFileDone()
Indicates that the file has been successfully saved.

void RemoteOff()
Indicates that IndiCom has taken control. In this case a StartRemote has to be sent before any other function can be called.

void ShutDown()
Indicates that IndiCom shuts down. In this case, the client has to call EndRemote, EndSession and to release its Interface to IndiCom (set it to nothing). Otherwise IndiCom will stay in memory.

void Error(long errorCode)
Indicates that an error has occurred. See below for possible error codes.

void ResultNamesChanged()
Indicates that the result names in IndiCom have changed.

void AverageDataDone()
Indicates that the call to AverageData was successful.

void BreakDone()
Indicates that a Break was successfully done.

void LoadParFileDone()
Indicates that a parameter file was successfully loaded.

void RecalculateDone()
Indicates that the recalculation has successfully ended.

void SetParameterDone()
Indicates that a parameter was successfully set.

void ExecuteScriptDone ()

Indicates that a script was successfully executed.

void Recalculate2ndStepDone ()
Indicates that the 2nd step results were successfully calculated.

2.7 Keys for SetParameter and GetParameter

ENAM -> Engine Name
HUBF -> Stroke
PLEU -> Conrod
BORE -> Bore
EPSI -> Compression
DESAXI -> Pinoff1
DESAXI2 -> Pinoff2 (only IndiSet)
NACY -> Measurement duration
SCOPEBUFFER -> Scopebuffer
COMMENT -> IFile comment
AVERAGE -> Averaging of Rawdata after a measurement
PREF->Set reference value of all cylinder pressure signals
PREFx->Set reference value of cylinder pressure signal x (e.g: PREF2)
MODE_RAWDATA -> Turn Rawdata measurement on
MODE_RTP -> Turn RTP measurement on
MODE_KNOCK -> Turn knock measurement on (only IndiSet)
MODE_TIME -> Turn time measurement on
OP_P01, OP_P02 .. OP_P99 -> Operating Parameters
%xxx -> User Variable

3 IndiCom Remote Errors:

0… No Error
16… SaveFile: Drive not ready
17… SaveFile :File not accessable
19… SaveFile: Read/Write Error
22… SaveFile: Disk full
23… SaveFile: File Exists
30… LoadParFile: File not found
31… LoadParFile: No file specified
32… Crank angle mark error
33… ADC Time Out
34… ADC Range violation
35… no DRQ Data present

1000…Critical Error
2000… Non-Critical Error
3000… StartRemote not called

4 Changes from IndiCom 1.0 to IndiCom 1.1 and higher

There are a few differences in the COM Interface between IndiCom 1.0 and IndiCom1.1/1.2

- The progId changed
- The typelibrary changed from “ConcApp.tlb” to “IndiCom.tlb”
- The Interface name changed from “ConcApp.RemoteInterface” to “IndiCom.RemoteInterface”

A few things have to be done in programs that use the IndiCom RemoteInterface via COM when IndiCom is updated from Version 1.0 to Version 1.1 or 1.2:

1. Change the program so that it uses the IndiCom.tlb instead of the ConcApp.tlb that means in VB that the reference to “Concerto Automation” has to be changed to “IndiCom Automation”
2. Change every ConcApp.RemoteInterface to IndiCom.RemoteInterface
3. Recompile the program.