

---

# **Automation of Windows Applications with Object Rexx**

---

**Florian Helmecke**

Bachelor of Science in Business Administration

# Table of Content

List of Figures	15
List of Tables	19
List of Object Rexx Scripts	20
List of other Scripts	23
Acronyms	24
Preamble	29
1. Object Rexx	31
2. The Component Object Model	33
2.1. COM Interfaces	33
2.2. The COM Library	35
2.3. Further Items	35
2.3.1. Process	35
2.3.2. Thread	36
2.3.3. Apartment	36
2.3.3.1. Single-Threaded Apartment	36
2.3.3.2. Multi-Threaded Apartment	36
2.3.4. Monikers	36
2.4. COM Clients and Server	37
2.4.1. COM Client Getting a Pointer to an Object	37
2.4.2. The CLSID	38
2.4.2.1. Creation of CLSID with Uuidgen.EXE	39

2.4.2.2.	Creation of CLSID with GUIDGen.EXE	40
2.4.3.	Communication of Objects	41
2.5.	Defining COM Interfaces	41
2.6.	The Registry	42
2.6.1.	The Registry Editor	44
2.6.2.	The OLE/COM Object Viewer	46
2.6.3.	The WindowsRegistry Class	47
2.7.	Distributed COM (DCOM)	49
2.7.1.	COM Components in different Processes	49
2.7.2.	COM Components on different Machines	50
2.7.3.	Features of DCOM	50
2.8.	COM+ (Component Service)	52
2.8.1.	New Features of COM+	52
2.8.2.	Features for Creating Applications	53
3.	ActiveX	55
3.1.	History of OLE/ActiveX	55
3.2.	Object Linking and Embedding	56
3.2.1.	Linking	56
3.2.2.	Embedding	57
3.2.3.	The Class ID (CLSID) of the OLE object	57
3.2.4.	Features of ActiveX	57
3.3.	ActiveX Control	58

3.3.1.	Interfaces	59
3.3.2.	Further Characteristics	61
3.3.2.1.	Licensing	61
3.3.2.1.1.	Design-Time Licensing	61
3.3.2.1.2.	Run-Time Licensing	61
3.3.2.2.	Initialization Security	62
3.3.2.3.	Compression	62
3.3.2.4.	Self-Registration	62
3.3.2.5.	Digital Signature and Certification	63
4.	ActiveX Automation	64
4.1.	ActiveX Client	65
4.2.	ActiveX Object	65
4.3.	Important Interfaces	67
4.4.	Interaction of Objects and Clients	69
4.4.1.	Dual Interface	71
4.4.2.	Object Access with the IDispatch Interface	72
4.4.3.	ID Binding	72
4.4.4.	Late Binding	73
4.4.5.	Object Access with the VTBL	73
4.4.6.	Out-of-Process Servers	73
4.4.7.	In-Process Servers	73
4.5.	Exposing ActiveX Objects	73

4.5.1.	Initializing of exposed Objects	74
4.5.2.	Implementation of the exposed Objects	75
4.5.3.	Implementation of the Class Factory	75
4.5.4.	The Application Object	76
	Registration	76
4.5.6.	Registration of Classes	77
4.5.7.	Releasing of the exposed Objects and OLE	79
4.5.8.	Retrieving of the Objects	79
4.5.9.	The Returning of Objects	80
4.5.10.	Termination of Objects	80
4.6.	Design of an Application which is Automated	80
4.6.1.	IUnknown Interface	80
4.6.2.	IDispatch Interface	80
4.6.3.	Dual Interface	81
4.6.4.	Registration of Interfaces	81
4.6.5.	Creation of a CLSID	82
4.6.6.	IEnumVARIANT Interface	82
4.7.	Type Library	82
4.7.1.	Creation of a Type Library	83
4.7.2.	Registration of a Type Library	84
4.7.3.	Error Handling	85
4.8.	Access of ActiveX Objects	85

5.	How to Get Script Code	86
5.1.	Trial and Error	86
5.2.	Macro Recorder Tool	86
5.3.	Converting Visual Basic Script Code to Object Rexx	89
5.4.	Other Sources	90
6.	OLE and Object Rexx with OLEObject Class	94
6.1.	Methods of the OLEObject Class	95
6.2.	Type Conversion	95
6.3.	Init	97
6.3.1.	Init with ProgID	98
6.3.2.	Init with CLSID	98
6.3.3.	Init with WITHEVENTS	99
6.4.	UnKnown	100
6.4.1.	Unknown without Arguments	100
6.4.2.	Unknown with Arguments	101
6.4.3.	Unknown with identical Method Names	102
6.5.	GetObject	105
6.6.	GetConstant	106
6.6.1.	GetConstant with the Name of the Constant	107
6.6.2.	GetConstant without the Name of the Constant	109
6.7.	GetKnownEvents	110
6.8.	GetKnownMethods	114

6.9. GetOutParameters	118
7. Tools	120
7.1. METHINFO.rex	120
7.2. OLEInfo.rex	120
7.3. RGF_OLEInfo.hta	121
8. Useful Object Rexx Classes	124
8.1. WindowObject	124
8.2. MenuObject	124
8.3. Object Rexx Classes Remoting the User Interface	124
8.3.1. WindowsProgramManager	124
8.3.2. WindowsManager	124
8.3.3. WindowsClipboard	125
8.3.4. WindowsRegistry	125
8.3.5. WindowsEventLog	125
9. Embedding Object Rexx in HTML or XML	126
9.1. Tag	126
9.2. Document Type Definition (DTD)	126
9.3. HTML (Hypertext Markup Language)	126
9.4. XML (eXtensible Markup Language)	127
9.5. Cascading Style Sheets (CSS)	127
9.6. Document Object Model (DOM)	127
9.7. Microsoft Internet Explorer (MSIE)	128

9.7.1.	Embedding an Object Rexx Script in HTML	129
9.7.2.	MSIE and Error	131
10.	WMI	133
10.1.	Win32_Service	133
10.2.	Win32_OperatingSystem	135
10.3.	Win32_DiskPartition	136
10.4.	Win32_LogicalDisk	136
10.5.	Win32_Process	138
10.6.	Win32Shutdown	139
11.	Automation of Microsoft Agent Technology	141
11.1.	Introduction to MS Agent Technology	141
11.2.	Overview of MS Agent Technology	143
11.3.	MS Agent and Events	151
12.	Automation of Microsoft Speech	154
12.1.	Text-To-Speech Synthesis (TTS)	155
12.1.1.	Introduction to MS Speech TTS	156
12.1.2.	MS Speech TTS embedded in HTML	158
12.1.3.	Reading a MS Word Document	162
12.2.	Speech Recognition	165
12.2.1.	Dictation Recognition	166
12.2.2.	Command and Control Recognition	168
12.2.2.1.	C&C Recognition with Configuration File	168



12.2.2.2.	C&C and the Creation of a new Grammar Rule	170
13.	Windows Script Host	173
13.1.	Scripting	174
13.1.1.	Script Basics	174
13.1.2.	Server-Side Scripting	175
13.2.	Object Rexx and Windows Script Host	176
13.2.1.	Basics	176
13.2.2.	COM Interfaces	176
13.3.	Host and Engine	177
13.3.1.	Script Host and Script Engine Basics	177
13.3.2.	Interaction between Scripting Host and Engine	178
13.4.	Types of Script File	179
13.4.1.	WSF File Using Windows Script Files (.wsf)	179
13.4.2.	WSH File	180
13.4.3.	REX File	180
13.4.4.	RXS File	180
13.5.	Running a Script	181
13.5.1.	CScript	181
13.5.2.	WScript	181
13.5.3.	Embedding a Script in a HTML File	182
13.5.4.	Other Possibilities to Run a Script	182
13.6.	Instantiating of Objects	182

13.7.	WSH Object Model	183
13.7.1.	WshArguments Object	184
13.7.2.	WshController	186
13.7.2.1.	WSHController on the local Machine	188
13.7.2.2.	WSHController on multiple Machines	189
13.7.2.3.	WSHController and Events	194
13.7.3.	WshNetwork Object	195
13.7.4.	WshShell	198
13.7.4.1.	Run Method and SendKeys Method	198
13.7.4.2.	Accessing the Registry	200
13.7.4.3.	Creation of Shortcuts	201
13.7.4.3.1.	Creation of a Shortcut	201
13.7.4.3.2.	Creation of an UrlShortcut	202
13.7.4.3.3.	Deletion of a Shortcut	203
13.7.4.4.	WshEnvironment	204
13.7.4.5.	WshScriptExec	205
13.8.	FileSystemObject Object	206
13.8.1.	The AvailableSpace Property	206
13.8.2.	DriveType Property	207
13.8.3.	Creation of a Folder with a WSF File	208
13.8.4.	Creating a Text File	209
13.8.5.	Attribute Property	209

13.8.6.	Copying a File	210
13.8.7.	Deleting Files and Folders	210
13.9.	Dictionary Object	211
13.10.	Security in Windows Script Host	211
13.10.1.	SignFile and VerifyFile Methods	214
13.10.2.	Sign and Verify methods	218
13.11.	Starting Applications with WSH	221
13.12.	Windows Script Components	222
13.12.1.	Windows Script Components Basics	223
13.12.2.	Structure of Windows Script Components	223
13.12.2.1.	The Registration	224
13.12.2.2.	Exposing the Functions	226
13.12.2.3.	The Script Code	227
13.12.2.4.	Accessing the Functions of the Component	229
13.12.2.5.	Remote Instantiation of a Script Component	230
13.12.2.6.	Windows Script Component Wizard	233
14.	MS.NET	234
14.1.	Smart Devices	235
14.2.	Web Services	235
14.2.1.	Simple Object Access Protocol	236
14.2.2.	Web Services Description Language	236
14.2.3.	Universal Discovery Description and Integration	236

14.2.4.	Difference of Web Site and XML Web Service	236
14.3.	NET Framework	237
14.3.1.	Common Language Runtime	237
14.3.2.	The Assembly	239
14.3.3.	Metadata	243
14.3.4.	Cross-Language Interoperability	244
14.3.5.	.NET Framework Class Library	245
14.4.	Programming with the .NET Framework	247
14.4.1.	ADO.NET	247
14.4.1.1.	Objects of ADO.NET	247
14.4.1.2.	ADO.NET Architecture	248
14.4.2.	.NET Remoting	249
14.4.3.	Accessing the Internet	251
14.4.4.	Active Directory-Components	252
14.4.5.	CodeDOM	253
14.4.6.	Components Development	254
14.4.7.	Developing World-Ready Applications	255
14.4.8.	Asynchronous Calls	256
14.4.9.	Creation of Messaging Components	257
14.4.9.1.	Basic Knowledge of Messaging	258
14.4.9.2.	Types of Queues	258
14.4.10.	Windows Management Instrumentation	259

14.4.11.	Processing Transactions	259
14.4.12.	Security	260
14.4.12.1.	Basic Security Terms	260
14.4.12.2.	Access Security	261
14.4.12.3.	Role-based Security	261
14.4.12.4.	Cryptography	261
14.4.12.5.	Security Policy Management	263
14.4.12.6.	Security Tools	263
14.4.13.	System Monitoring Components	263
14.4.14.	Microsoft .NET Passport	264
14.5.	Building Applications	265
14.5.1.	ASP.NET	265
14.5.2.	Windows Service Applications	266
14.5.3.	Windows Forms	267
14.5.4.	Design-Time Support	267
14.6.	Object Rexx and MS.NET	268
14.6.1.	Exposing of .NET Framework Components for Usage with COM	269
14.6.2.	Assembly Registration Tool (Regasm.exe)	270
14.6.3.	COM Interop	273
14.6.3.1.	COM Wrappers	273
14.6.3.2.	COM Callable Wrapper	274
14.6.4.	Conclusion	279

15. Examples of Use	280
15.1. Döner Dome Restaurant	280
15.2. High Value Customers Consultancy	281
15.3. Tourplanning	282
Summary	285
Bibliography	288
MS Library Sources	289
Microsoft Library .NET Framework Sources	300
Other Sources:	307
System Administration Scripting Guide Sources	311
Microsoft Speech SDK 5.1 Documentation Sources	312

## List of Figures

<i>Figure 1:</i>	History of Rexx	32
<i>Figure 2:</i>	Snapshot of the MS-DOS shell with uuidgen.exe	40
<i>Figure 3:</i>	Snapshot of GUIDGen.EXE	40
<i>Figure 4:</i>	Snapshot of the window of Start->Run	45
<i>Figure 5:</i>	Snapshot of the Registry Editor	45
<i>Figure 6:</i>	Snapshot of OLE/COM Object Viewer	46
<i>Figure 7:</i>	Snapshot of the Registry Editor with the new key "TestKey"	48
<i>Figure 8:</i>	COM components in different processes	50
<i>Figure 9:</i>	DCOM: COM components on different machines	50
<i>Figure 10:</i>	Timeline of ActiveX	56
<i>Figure 11:</i>	Relations among ActiveX objects and ActiveX clients	65
<i>Figure 12:</i>	Some objects of MS Excel	66
<i>Figure 13:</i>	VTBL with IUnknown and IDispatch interface.	70
<i>Figure 14:</i>	Dispatch interface is not supported	71
<i>Figure 15:</i>	Accessing an Object Through the IDispatch Interface	72
<i>Figure 16:</i>	Interfaces that should be implemented to expose ActiveX	75
<i>Figure 17:</i>	Interaction of ActiveX components, CLSIDs and ProgIDs.	78
<i>Figure 18:</i>	IEnumVARIANT interface	82
<i>Figure 19:</i>	Macro Recorder tool	87
<i>Figure 20:</i>	MS Word macro, which recorded that a text is typed.	88

---

<i>Figure 21:</i>	Outlook Express 6 with the newsgroup microsoft.public.msagent.	93
<i>Figure 22:</i>	Interaction of OLE object, OLEObject and Script	94
<i>Figure 23:</i>	Snapshot of MS Excel	105
<i>Figure 24:</i>	Snapshot of MS Word	109
<i>Figure 25:</i>	Object Rexx Workbench with the command line	111
<i>Figure 26:</i>	Snapshot of MS Excel	114
<i>Figure 27:</i>	Snapshot of MS Excel	118
<i>Figure 28:</i>	OLE/ActiveX Object Viewer with functions of Qualcomm Eudora	121
<i>Figure 29:</i>	Snapshot of the start page of "rgf_oleinfo.hta"	122
<i>Figure 30:</i>	Snapshot of "RGF_OLEInfo.hta" with compact listing.	123
<i>Figure 31:</i>	Example for DOM	128
<i>Figure 32:</i>	Snapshot of Embedding Object Rexx in HTML.htm.	131
<i>Figure 33:</i>	Error handling with the MSIE.	132
<i>Figure 34:</i>	Snapshot of MSAgents	141
<i>Figure 35:</i>	Snapshot of Merlin	143
<i>Figure 36:</i>	Snapshot of Agent_Overview.rex	151
<i>Figure 37:</i>	Snapshot of the MS Speech SDK 5.1 Help	155
<i>Figure 38:</i>	Speech recognition process flow	156
<i>Figure 39:</i>	Snapshot of MSSpeech_TTS_2.hta	162
<i>Figure 40:</i>	Snapshot of the IBM Object Rexx Workbench with command line.	163
<i>Figure 41:</i>	Speech recognition process	166
<i>Figure 42:</i>	Interaction between Scripting Host and Engine	178



<i>Figure 43:</i>	WSH Object Model	183
<i>Figure 44:</i>	Snapshot of the MS-DOS Shell	185
<i>Figure 45:</i>	Snapshot of the registry with regedit.exe	188
<i>Figure 46:</i>	Snapshot of the Security Policy console.	191
<i>Figure 47:</i>	Snapshot of the start page of the User Accounts	192
<i>Figure 48:</i>	Snapshot of the form for creating a password	193
<i>Figure 49:</i>	WSHNetwork1.JPG: \\Server =Antares\Public=Eigene Daten	196
<i>Figure 50:</i>	WSHNetwork2.JPG: Shows the new network drive" Z:"	196
<i>Figure 51:</i>	WSHNetwork3.JPG: Shows the network printer	197
<i>Figure 52:</i>	Creation of a certificate	212
<i>Figure 53:</i>	Certificate Snap-In	213
<i>Figure 54:</i>	Signed source code of code 50	216
<i>Figure 55:</i>	MSDOS Shell with the command to sign the script	217
<i>Figure 56:</i>	Message box to verify the script.	218
<i>Figure 57:</i>	Dialog box which occurs after the Verify method is invoked.	221
<i>Figure 58:</i>	Snapshot of Registration field.	224
<i>Figure 59:</i>	ITypeInfo Viewer of the OLE/COM Object Viewer	229
<i>Figure 60:</i>	Microsoft Windows Script Component Wizard.	233
<i>Figure 61:</i>	Differences between Web Site and Web Service	237
<i>Figure 62:</i>	All elements are united in a single file.	241
<i>Figure 63:</i>	Multifile assembly	241
<i>Figure 64:</i>	Single-file assembly and multifile assembly	242

<i>Figure 65:</i>	ADO.NET architecture	249
<i>Figure 66:</i>	Remotingprocess	251
<i>Figure 67:</i>	Messengerouting between sites	258
<i>Figure 68:</i>	Levels of design-time support	268
<i>Figure 69:</i>	Illustration of a registry entry with a reference to Mscoree.dll	271
<i>Figure 70:</i>	RGF_OLEINFO.HTA with the new created ProgIDs of .NET classes	272
<i>Figure 71:</i>	Principle of RCW and CCW	274
<i>Figure 72:</i>	Access of a CCW	275
<i>Figure 73:</i>	COM interfaces for the CCW	275
<i>Figure 74:</i>	CCW method call	276
<i>Figure 75:</i>	RGF_OLEInfo.hta: New ProgID “AndyMc.CSharpCOMServer”	278
<i>Figure 76:</i>	User interface of the cash registering system of project Döner Dome	281
<i>Figure 77:</i>	User interface of High-Value Customers consultancy	282
<i>Figure 78:</i>	Tourplanning	284
<i>Figure 79:</i>	Microsoft Document Explorer	289
<i>Figure 80:</i>	Microsoft Document Explorer	300
<i>Figure 81:</i>	System Administration Scripting Guide	311
<i>Figure 82:</i>	Microsoft Speech SDK 5.1 Documentation	312

## List of Tables

<i>Table 1:</i>	OLE Control interfaces	60
<i>Table 2:</i>	Important interfaces for OLE Automation.	69
<i>Table 3:</i>	VBScript code to Object Rexx	90
<i>Table 4:</i>	Methods of the OLEObject class	95
<i>Table 5:</i>	Type conversion	97

## List of Object Rexx Scripts

<i>Code 1:</i>	WindowsRegistryClass.REX	48
<i>Code 2:</i>	Init_Instantiation of Word with ProgID.REX	98
<i>Code 3:</i>	Init_Instantiation of Word with CLSID.REX	99
<i>Code 4:</i>	Init_WITHEVENTS.rex	100
<i>Code 5:</i>	UnKnown_without_Arguments.rex	101
<i>Code 6:</i>	UnKnown_with_Arguments.rex	102
<i>Code 7:</i>	UnKnown_Identical_Methodnames.rex	104
<i>Code 8:</i>	GetObject.rex	106
<i>Code 9:</i>	GetConstant_GetConstant with name of Constant.REX	108
<i>Code 10:</i>	GetConstant_GetConstant without name of Constant.REX	110
<i>Code 11:</i>	GetKnownEvents_AllEventsOfAnApplication.rex	114
<i>Code 12:</i>	GetKnownMethods_AllMethodsOfAnApplication.rex	118
<i>Code 13:</i>	Embedding Object Rexx in HTML.htm	130
<i>Code 14:</i>	WMI_ListAllServicesOnTheSystem.rex	135
<i>Code 15:</i>	WMI_Win32_OperatingSystem.rex	136
<i>Code 16:</i>	WMI_Win32_DiskPartition.rex	136
<i>Code 17:</i>	WMI_Win32_LogicalDisk.rex	138
<i>Code 18:</i>	WMI_LaunchANewProcess.rex	139
<i>Code 19:</i>	WMI_Win32Shutdown.rex	140
<i>Code 20:</i>	Agent_Intro.rex	143
<i>Code 21:</i>	Agent_Overview.rex	150

<i>Code 22:</i>	Agent_Events.rex	153
<i>Code 23:</i>	MSSpeech_TTS_1.rex	158
<i>Code 24:</i>	MSSpeech_TTS_2.hta	161
<i>Code 25:</i>	MSSpeech_TTS_3_Word.rex	165
<i>Code 26:</i>	MSSpeech_SR_1_Dictation.rex	168
<i>Code 27:</i>	MSSpeech_SR_2_CommandAndControl.rex	169
<i>Code 28:</i>	MSSpeech_SR_3_CommandAndControl_2.rex	172
<i>Code 29:</i>	WSH_Arg.rxs	186
<i>Code 30:</i>	WSHController_OnTheSameMachine.rex	189
<i>Code 31:</i>	WSHRemoteCalc.rxs	189
<i>Code 32:</i>	WshController_OnMultipleMachines.rxs	194
<i>Code 33:</i>	WshControllerWithEvents.rxs	195
<i>Code 34:</i>	WSHRemoteCalc.rxs	195
<i>Code 35:</i>	WshNetwork.REX	198
<i>Code 36:</i>	RunMethodAndSenkeysMethod.htm	200
<i>Code 37:</i>	WSHRegistry.rex	201
<i>Code 38:</i>	CreationOfAShortcut.rex	202
<i>Code 39:</i>	CreationOfAnUrlShortcut.rex	203
<i>Code 40:</i>	DeletionOfAShortcut.rex	204
<i>Code 41:</i>	WshEnvironment.rex	205
<i>Code 42:</i>	WshScriptExec.rxs	206
<i>Code 43:</i>	FSOAvailableSpace.rex	207

<i>Code 44:</i>	FSODriveType.rex	208
<i>Code 45:</i>	FSOCreationOfANewFolder.wsf	209
<i>Code 46:</i>	FSOCreationOfATextfile.rex	209
<i>Code 47:</i>	FSOFileAttribute.rex	210
<i>Code 48:</i>	FSOCopyingAFileWithANewName.rex	210
<i>Code 49:</i>	FSODeletionOfTheFilesAndFolders.rex	211
<i>Code 50:</i>	DictionaryObject.rex	211
<i>Code 51:</i>	ScriptWhichIsToSign.wsf	214
<i>Code 52:</i>	WSH_Scripting_Signer_SignFile.rex	215
<i>Code 53:</i>	Signer_VerifyFile.rxs	216
<i>Code 54:</i>	WSH_ScriptingSigner_Sign.rex	219
<i>Code 55:</i>	WSH_ScriptingSigner_Verify.rex	220
<i>Code 56:</i>	WSH_EmbeddingAScriptInHTML.htm	222
<i>Code 57:</i>	WSH_WSC_Test.wsc	229
<i>Code 58:</i>	WSH_WSC_TEST.rex	230
<i>Code 59:</i>	WSH_WSC_Remote.wsc	232
<i>Code 60:</i>	MS_NET_System_Random.rex	273
<i>Code 61:</i>	MS_NET_AndyMc_CSharpCOMServer.rex	279

## List of other Scripts

<i>OtherScript 1:</i>	Excel macro for UnKnown_Idetical_Methodnames.rex	103
<i>OtherScript 2:</i>	MS Word macro for code 9	108
<i>OtherScript 3:</i>	VBS script code for Win32_Service	133
<i>OtherScript 4:</i>	VBS script code for Win32_LogicalDisk	137
<i>OtherScript 5:</i>	VBS script code for Win32_Process	138
<i>OtherScript 6:</i>	VBS script for the demonstration of Win32Shutdown	139
<i>OtherScript 7:</i>	Macro for MSSpeech_TTS_3_Word.rex.	164
<i>OtherScript 8:</i>	solx.xml	170
<i>OtherScript 9:</i>	Remote.vbs	194
<i>OtherScript 10:</i>	Sign method with Visual Basic Script	219
<i>OtherScript 11:</i>	Verify method with Visual Basic Script	219
<i>OtherScript 12:</i>	RemoteWSC.vbs	233
<i>OtherScript 13:</i>	testcomserver.cs	277

## Acronyms

ACF	Application Configuration File
ADO	ActiveX Data Object
ADSI	Active Directory Service Interfaces
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
ATL	ActiveX Template Library
BAT	MS DOS batch file extension
CATID	CATegory ID
CCW	COM Callable Wrapper
CLS	Common Language Specification
CLSID	CLaSS IDentifier
CodeDOM	Code Document Object Model
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRL	Certification Revocation List
CSS	Cascading Style Sheets
CTL	Certificate Trust List
DB	DataBase
DCERPC	Distributed Computer Environment Remote Procedure Call
DCOM	Distributed COM
DDE	Dynamic Data Exchange
DEF	module-DEFinition file
DHTML	Dynamical HTML



---

DISPID	DISPatch IDentifier
DLL	Dynamic Linked Library
DNS	Domain Name Service
DOM	Document Object Model
DOS	DOS
DTD	Document Type Definition
EXE	Executable program
FTC	Federal Trade Commission
GUI	Graphical User Interface
GUID	Globally Unique IDentifier
HTA	HTML Application
HTML	Hypertext Markup Language
HTML	HTML file extension
HTTP	HyperText Transfer Protocol
I/O	Input/Output
IBM	International Business Machines
ID	IDentifier
IDL	Interface Definition Language
IID	Interface IDentifier
IIS	Internet Information Services
IPX/SPX	Internet Packed Exchange/Sequenced Packed Exchange
IT	Information Technology
JIT	Just-In-Time
JPEG	Joint Photographic Expert Group
JS	Jscript file extension
L&H	Lernout&Hauspie

---

LAN	Local Area Network
LCID	LoCal IDentifier
LDAP	Lightweight Directory Access Protocol
LPC	Local Procedure Call
LPK	License PackAge file
MDI	Multiple Document Interface
MemberID	Member IDentifier
MFC	Microsoft Foundation Classes
MIDL	Microsoft Interface Definition Language
MS	MicroSoft
MSDN	Microsoft Developer Network
MS-DOS	MicroSoft-Disk Operating System
MSIE	Microsoft Internet Explorer
MSIL	MicroSoft Intermediate Language
MTS	Microsoft Transaction Server
MVP	Most Valuable Professional
NetBIOS	Network Basic Input/Output System
OCX	ActiveX file extension
ODL	Object Description Language
OLE	Object Linking and Embedding
PDF	Portable Document Format
PE	Portable Executable
ProgID	Program IDentifier
RAD	Rapid Application Development
RC	Resource Compiler
RCW	Runtime Callable Wrapper

---

RegDB	Registration DataBase
REX	Object REXx file extension
Rexx	REstructured eXtended eXecutor
ROT	Running Object Table
RPC	Remote Procedure Call
RXS	ObjectRexxScriptFile extension
SAPI	Speech Application Programming Interface
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SP3	Service Pack 3
SQL	Structured Query Language
SR	Speech Recognition
SSI	Single Sign In
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLB	Type Library file extension
TTS	Text-To-Speech
UDDI	Universal Discovery Description and Integration
UDP	User Datagram Protocol
UDT	Uniform Data Transfer
UDT	User-Defined Type
UNIX	UNIplexed Information and Computing System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
US	United States

UUID	Universally Unique ID
VB	Visual Basic
VBA	Visual Basic for Applications
VBS	Visual Basic Script file extension
VBScript	Visual Basic® Scripting Edition
VTBL	Virtual function TaBL
WAN	Wide Area Network
WAV	Wave file extension
Win	Windows
Windows Me	Windows Millennium Edition
Windows NT	Windows New Technology
Windows SE	Windows Standard Edition
Windows XP	Windows eXPerience
WMI	Windows Management Instrumentation
WSC	Windows Script Components
WSC	Windows Script Component file extension
WSDL	Web Services Description Language
WSE	Windows Script Engine
WSF	Windows Script Host file extension
WSH	Windows Script Host
WSH	Windows Script Host file extension
WYSIWIG	What You See Is What You Get
XML	eXtensible Markup Language

## Preamble

This master thesis with the subject “*Automation of Windows Applications with Object Rexx*” is a reference work. It will help in obtaining solutions to Automation problems with Object Rexx. Object Rexx code examples explain many fields of the Automation technology. This paper was written for the Associate Professorship of Computer Science in Economics III and New Media of Prof. Dr. Rony G. Flatscher at the University of Augsburg.

First, there is an introduction to the Component Object Model. After that, the ActiveX Automation technology is explained theoretically and then practically with Object Rexx samples. Often the Object Rexx samples are explained with the source code of another language like Visual Basic Script. It will help solve Automation problems with Object Rexx through the comparative analysis of possible solutions. This is necessary because in the most cases there is only Visual Basic Script and no Object Rexx instruction available. Also included in this paper is a list of sources for more information, a description of tools and an introduction of the embedding of Object Rexx in HTML/XML. The Object Rexx `OLEObject` class is described with examples. The Automation technology is explained by Microsoft Office components, the MS Agent technology, MS Speech technology, Windows Management Instrumentation (WMI) and Windows Script Host (WSH). These technologies are described theoretically and then practically with Object Rexx scripts. Then there is an introduction to MS.NET and a description of access possibilities of MS.NET functions with Object Rexx or other languages via OLE. This is significant because Object Rexx has no language compiler for the .NET Framework. In the end are some examples, which describe how to use this technology in the economical environment.

The terms ActiveX and OLE are used synonym. All Object Rexx scripts have Visual Basic, Visual Basic Script or JScript codes as draft. Each Object Rexx script can also be written with these languages.

The scripts in this master thesis were programmed with Object Rexx version 2.1.2. on a MS Windows XP Home machine (German). New OREXXOLE.DLL files are needed for some chapters. MS Word 2002 (German), MS Excel 2000 (German), and

MS Internet Explorer 6.0 (German) were used. Further software, which is necessary, is described in the sections.

The author says thank-you to Times L. Richardson for marking this paper, and to the development office of the language Object Rexx that supported the author patiently and competently.

Send comments to this address: [Florian.Helmecke@Web.DE](mailto:Florian.Helmecke@Web.DE)

### *Document Conventions:*

In this paper Object Rexx script code is written in a framed field:

Object Rexx script code
-------------------------

Comments have a green font colour.

Object Rexx script code comment
---------------------------------

Text, written between quotation marks, is written in blue font colour.

Object Rexx script code text written between quotation marks
--

Other script codes like Visual Basic Script code is used to explain the Object Rexx code. It is written with a coloured background.

Other script code
-------------------

Folder specifications or commands like method names, which are written in the normal text, are written with `Courier New`.

Sometimes in the lettering of a figure “Snapshot” is written. That means that the content of the display was copied to the clipboard by pressing `Ctrl`, `Alt` and `Print`.

# 1.Object Rexx

In 1979 Rexx (REstructured eXtended eXecutor) was developed by Mike F. Cowlishaw (IBM-Fellow). It was the successor for the cryptical script language EXEC for IBM mainframes. It has a simple and easy to learn syntax.

Since the beginning of the 90s an object-orientated version of Rexx was developed. This language Object Rexx is fully compatible with the classic Rexx and is internally built object-orientated. Procedural commands are internally converted to object-orientated commands. Object Rexx has a powerful object model and is available for a lot of operating systems like OS/2, AIX, Linux or Windows. Object Rexx uses an interpreter and is interactive. [Fla02]

Object Rexx also supports messaging, polymorphism, classes, objects, methods, inheritance and multiple inheritance. [IBM03]

Figure 1 gives a survey of the history of Rexx.



SOFTWARE SOLUTIONS DIVISION

## History of REXX

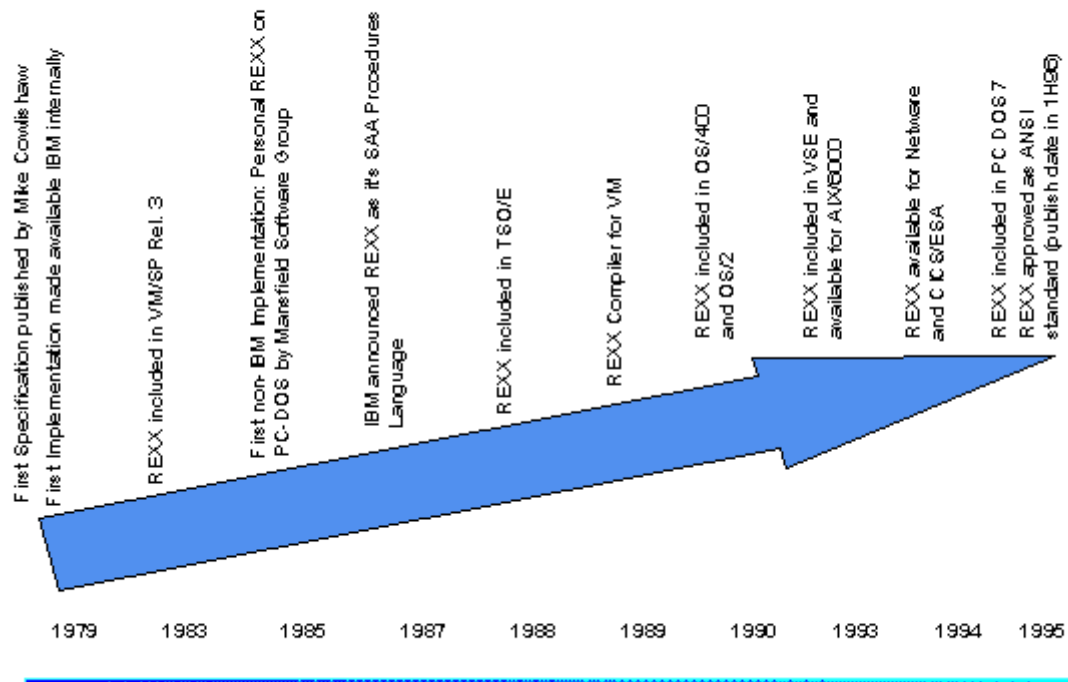


Figure 1: History of REXX<sup>11</sup>

<sup>1</sup> Taken from [IBM03]



## 2.The Component Object Model

This chapter describes the Component Object Model with its interfaces, the COM Library, some basic concepts, COM clients and server, the CLSID and how to get one, the registry and ways to modify it, DCOM and COM+. This chapter concerns elements which are used for technologies described later in this document.

The Component Object Model (COM) is a technology from Microsoft that is object-based, RPC-based [Fla03], distributed and platform-independent. COM offers an object model that makes it possible for objects to interact with other objects. The objects could be in the same process, in another process or on a remote machine. COM is also a binary-standard, which means that the objects can be written in different object-orientated languages and differ in their structure. Examples for such languages are C, Basic, Java or Object Rexx. COM is the basis for OLE (Object Linking and Embedding) or later called ActiveX<sup>2</sup>. [MLCOMa]

Since Windows 2000 COM supports asynchronous calling, the control returns without delay after a given command and the client can continue its work while the command is worked on. [MLMMGa]

COM objects can be created with the Windows Script Components of Windows Script Host<sup>3</sup> [IBM02].

### 2.1. COM Interfaces

Interfaces are important because they enable the access of objects [MLCOMb]. Because it is the most important interface, the `Unknown` interface is also contained in this item. [MLCOMe]

Methods are the functions of the interfaces [MLCOMc]. Interface means in this case that it pertains to a group of functions [MLCOMb]. Clients can access the COM objects (also-called COM components [MLCOMa]) with a pointer to an interface. In this way the client can use the methods of the interface [MLCOMb]. The object must support all

---

<sup>2</sup> c.p. 3.

<sup>3</sup> c.p. 13.12.

methods of that interface. Otherwise, there is an error message if a method is used that is not supported by the object [MLCOMc].

The COM interfaces are unchangeable and each of them has its own interface identifier (IID), a subset of GUID (Globally Unique Identifier), which allows checking an object if it supports an interface and makes it unique [MLCOMd].

All interfaces inherit their functions from the `IUnknown` interface. Inheritance means that the following interfaces include all methods of the parent interface.

The `IUnknown` interface has three core methods: `QueryInterface`, `AddRef`, and `Release` [MLCOMe].

The `QueryInterface` method makes it possible for the client to access other interfaces and to navigate in an object [MLCOMf].

There are four conditions for this method:

- It must be transitive, which means that if a query from one interface to a second interface is successful and if a query from that second interface to a third interface is successful then a query from the first interface to the third interface must be also successful.
- The second requirement is that it is reflexive, which means that it must be successful if a pointer is held on an object and queried to that interface.
- It must be symmetric. If a query with a pointer to one interface is successful for another, a query through the obtained pointer must be successful for the first interface.
- It is static. That means if a call to `QueryInterface` for a pointer has success in pointing an interface the first time, it is successful the second time again.

In a decentralized system, it is not always known, if an object is still needed or not because there could be several clients who access an object. To solve this problem COM uses the reference count [MLCOMg].

By an access, the reference count for an interface on an object is raised with the `AddRef` method [MLCOMh]. Otherwise, the `Release` method reduces the reference

count. The memory is freed from the object if the reference count is zero and no client accesses this object [MLCOMi].

## 2.2. The COM Library<sup>4</sup>

The Library is initialized, respectively uninitialized, through each process that uses COM.

The COM library consists of DLL (dynamic linked library) and EXE files and it contains data about the following topics.

- A unique class identifier (CLSID) helps to locate the server that implements a class and where it is located.
- A standard mechanism makes it possible for applications to control the memory of their processes.
- In the library are API (application programming interfaces) functions that enable the programming of COM applications for clients and server.
- Remote procedure calls when an object is running in a remote server or local server.

## 2.3. Further Items<sup>5</sup>

This chapter describes some basic concepts to support the understanding of COM. These concepts are process, thread, apartment and monikers.

### 2.3.1. Process

A process is a collection of virtual data, memory space, code and system resources. The operating system makes it possible for processes to operate with the Microsoft's Remote Procedure Call (RPC) to communicate with each other.

---

<sup>4</sup> This section uses [MLCOMj]

<sup>5</sup> The following definitions refer to [MLCOMk]

## 2.3.2. Thread

A thread is an executed code in a process. Multithreaded applications are in danger of races (one thread has finished faster than another thread which depends on it) and deadlocks (each thread is waiting for another thread).

## 2.3.3. Apartment

A COM object is contained in one apartment. There are two types of apartments.

### 2.3.3.1. Single-Threaded Apartment

COM objects contained in the apartment can only receive method calls from that thread that pertains to the apartment. The method calls are synchronized.

### 2.3.3.2. Multi-Threaded Apartment

COM objects contained in the apartment can receive method calls from one or more threads that pertain to the apartment. The model is called free-threading and the calls are synchronized by the objects.

## 2.3.4. Monikers

A moniker is used to identify an object. It is an object which enables a component to get a pointer to an object and the moniker functions as a name which unambiguously identifies a COM object [MLGLOd]. This is called binding. A moniker is contained in a DLL and implements the `IMoniker` interface. There are moniker provider and moniker clients. The first is a component that supplies monikers identifying its objects to moniker clients and the second is a component that accesses a moniker to obtain a pointer to another object [MLCOMah].

There are the following monikers [MLCOMai]:

- Monikers used for almost any object in any location:
  - *File monikers* are used to identify any object that is contained in its own file [MLCOMaj].
  - *Composite monikers* are monikers that can characterize the relation among other monikers and that are a composition of other monikers [MLCOMak].

- *Item monikers* identify an object that is included in another object [MLCOMal].
- Monikers that are mainly used inside OLE:
  - *Anti-monikers* are used for the generation of new moniker classes [MLCOMam].
  - *Pointer-monikers* are used for the identification of an object that can occur only in the running or active state [MLCOMan].
- A so-called *Class moniker* can identify classes. They bind to the class object of the class [MLCOMao].

## 2.4. COM Clients and Server<sup>6</sup>

This item describes the interaction of COM server and client. Initially this paper explains how to get a pointer to an object. A basic element of COM is the CLSID. The properties of the CLSID and how to create it are explained. Finally the communication of objects is discussed.

A COM server supplies services to clients, which can be demanded with a pointer on the COM interfaces. There are two kinds of servers, the out-of-process and the in-process server. The out-of-process server runs in an EXE file in a remote machine or in a local machine. The in-process server runs in a dynamic linked library and is able to be implemented within an EXE process to use it for remote machines.

COM allows the employment within networks.

### 2.4.1. COM Client Getting a Pointer to an Object<sup>7</sup>

This section shows the possibilities how an object can be instantiated.

A client uses the services of a COM server. This is enabled by the methods of the interface of the server [MLCOMI].

---

<sup>6</sup>This section uses [MLCOMI]

<sup>7</sup>This section uses [MLCOMm]

There are four ways how clients can instantiate an object:

- Objects pass their interface pointer to the client directly with the implemented interface of another object for bidirectional communication.
- An object is called with its CLSID (CLaSS IDentifier).
- An API function in the COM Library is called to create an object of a predetermined type
- The method of an interface is called to create another object. Then an interface pointer on that other object is returned

### **2.4.2. The CLSID**

The CLSID is an important basic concept that is used very often in further chapters. For this reason an explanation of how to create a CLSID, is included.

The CLSID (Class identifier) is a globally unique identifier (GUID). It is compounded with an OLE class object. For a server application it is recommended that the CLSID be registered if a class object generates more than one instance of the object. [MLCOMn]

COM enables a client to launch a server and to have access to the interfaces methods through its CLSID. The server is like a COM class. This class is an implementation of a group of interfaces and can be used by different applications. The code is stored in DLL's or in executable files. The CLSID identifies the COM class and contains information of the location of the DLL or EXE code. If the server and the client are on the same system, the CLSID is the only thing needed. On distributed machines, a registry helps the server to be used by a client. A server signs on its location in the registry and can be called with the CLSID [MLCOMo].

The server is responsible for implementing the code for a class object. He is also responsible for registering its CLSID and for security [MLCOMp].

A class can be registered in the registry in the following ways: [MLCOMq]

- Registering at installation,

- self-registration,
- registering of objects in the ROT (running object table),
- installing as a user account or as a Win 32 service,
- registering a running EXE server.

A CLSID is a GUID (Globally Unique Identifier). That means that there is no other class with the same CLSID and in this way there cannot be any software collisions. As well if there are the same names for a class, the CLSID's are different [MLCOMr].

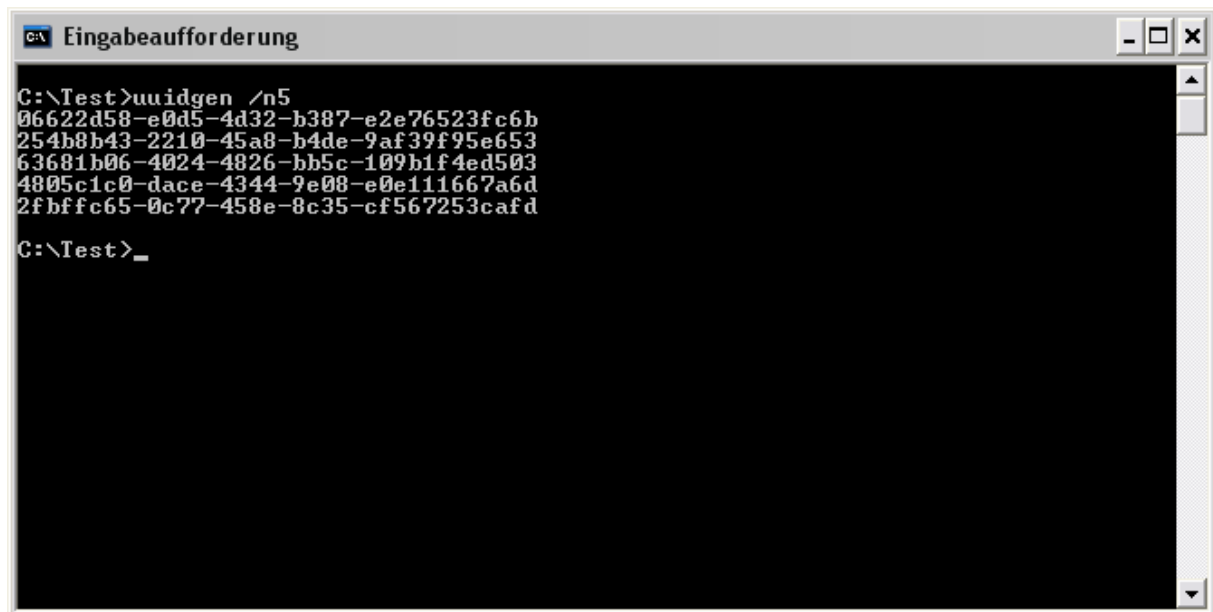
#### **2.4.2.1. Creation of CLSID with Uuidgen.EXE<sup>8</sup>**

This utility is part of the Visual Studio and helps to create unique CLSID's to prevent name collisions. The command `uuidgen /n5` in the shell prompt<sup>9</sup> like in figure 2 offers five CLSIDs.

---

<sup>8</sup> This section uses [MLCOMr]

<sup>9</sup> Start->Run->Command



**Figure 2:** Snapshot of the MS-DOS shell<sup>10</sup> with uuidgen.exe<sup>11</sup>

#### 2.4.2.2. Creation of CLSID with GUIDGen.EXE

Another tool to generate CLSID's is GUIDGen.EXE which is part of the Visual Studio. Figure 3 shows the user interface of GUIDGen.EXE creating a new CLSID [MLMMGb].



**Figure 3:** Snapshot of GUIDGen.EXE<sup>12</sup>

<sup>10</sup> Part of MS WindowsXP

<sup>11</sup> Part of MS Visual Studio



### 2.4.3. Communication of Objects<sup>13</sup>

If a client wants to call an object this object could be in-process or out-of-process.

In the case of in-process the client reaches the object directly. In the other case, out-of-process, it reaches only a proxy object supplied by the original object or COM. This proxy object implements the procedure call for the original object with all necessary parameters including the pointers. This procedure is called marshalling (coding [Fla03]). Marshalling means the packaging and the sending of interface method calls across process or thread boundaries [MLGLOa].

In the case of an in-process, the caller for an object is for the server the client. In the case of an out-of-process situation, the caller is a stub. This stub gets its instruction from the proxy. It unmarshals (decoding [Fla03]) the information and calls the object.

## 2.5. Defining COM Interfaces<sup>14</sup>

COM supplies many interfaces the developer can use. But if the supplied interfaces do not satisfy the requirements of an application the developer has to program its interface by himself. This chapter shows the prerequisites for programming an interface and which properties an interface must have.

The `IUnknown` interface is the interface from which all interfaces are deducible. After defining an interface it is described in MIDL (Microsoft Interface Definition Language), compiled and registered. To define interfaces the following things are required: MS Windows NT version 4.0 or later or Windows 95. A 32-bit C/C++ compiler with the MS Platform Software Development Kit is also required.

*The following steps are necessary to program an interface:*

- Decision about a type-library-driven marshalling or a proxy/stub DLL.
- Write a description of the interface in an interface definition language (IDL) file. In an application configuration file (ACF) specific details about the interface are

---

<sup>12</sup> Part of MS Visual Studio

<sup>13</sup> This section uses [MLCOMs]

<sup>14</sup> This section uses [MLCOMt]

written. In the case of a type-library-driven marshalling a library statement is added.

- With the MIDL compiler type library and header file, or interface identifier file, DLL data file and header file and proxy/stub files are generated.
- Dependent on the chosen marshalling method, a module definition (DEF) file is written. The MIDL-generated files are compiled and linked into a single proxy DLL. The interface is registered in the system registry or the type library is registered.

*Interfaces must have the following features:* [MLCOMu]

- They must be unchangeable. That means that after creation no part of the interface might be mutated.
- HRESULT must be returned for all methods. HRESULT shows if an action was a success or a failure<sup>15</sup>.
- The interfaces have a unique interface identifier (IID)
- The data types have to be remotable. If this isn't possible, marshalling and unmarshalling routines have to be created. If needed, a pointer to the `IUnknown` interface is created.
- The string parameters in interface methods have to be Unicode.

Remotable interfaces are important because of the distributed COM. MIDL makes it possible that the interfaces are used outside of the machine, process or the thread.

[MLCOMv]

Important for an efficient interface is the quantity of the data that is transferred in a method call and the rate of method calls across the interface border. [MLCOMv]

## 2.6. The Registry

The registry is the system database for the operating system with data about a lot of areas of the system. First it is briefly described and then several access possibilities

---

<sup>15</sup> c.p. 4.7.3.

are shown with tools like the Registry Editor, the OLE/COM Object Viewer or via an Object Rexx script.

The registry of Windows contains information about the users of a system, software and hardware configuration. The registry is searched by a client for information about components. Any application can read from and insert data to the registry [MLCOMw].

There is also information about the COM objects. When the registry is asked for a ProgID (Program Identifier) or the CLSID, the location of the EXE or DLL is given back. Then the server is either loaded into the process space of the client application in case of in-process components or in case of remote or local servers the server is started in its own process space. The server returns a reference to one interface of the components after he created an instance of this component [MLCOMw].

The hierarchy of the registry is structured as named values or single default values, subkeys and keys. The keys get the name by backslash-delimited strings and can have one or more values consisting of binary data integral values or strings [MLCOMx]. The registry of Windows XP has the following root keys: HKEY\_CLASSES\_ROOT, HKEY\_CURRENT\_USER, HKEY\_LOCAL\_MACHINE, HKEY\_USERS and HKEY\_CURRENT\_CONFIG which can be seen with the Registry Editor and the root key HKEY\_PERFORMANCE\_DATA (access of performance data) which can't be seen with the Registry Editor [MLWSla].

Before information can be added to the registry a key must be opened. For this, it is necessary to offer a handle to a key that is open. There are, through the system, predefined keys. It is possible that the usage of the handles differ from platform to platform. [MLWSla]

The HKEY\_CLASSES\_ROOT contains HKEY that is responsible to switch to a registry key. The whole phrase allows the configuration of the CLSID by the user in the path HKEY\_CLASSES\_ROOT\CLSID. The information that is stored under this key, is used by Shell and COM. Another key is HKEY\_CURRENT\_USER. With this key, the preferences of the user can be changed. These preferences could be colors, network connections, application preferences or environment variables. HKEY\_LOCAL\_MACHINE sets the physical state of the computer including the current configuration data and system information. HKEY\_USERS describes the default user

configuration for the current user and new users. The `HKEY_CURRENT_CONFIG` key is an alias for `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Hardware Profiles\Current` and offers information of the current hardware profile. [MLWSIa]

COM supports the self-registration with the functions `DllRegisterServer` and `DllUnregisterServer` for the DLL. [MLCOMy]

Each time an application is loaded, it should control whether the CLSID and the application's CLSID are present in the registry. If this is not the case, it should be registered as the original setup. It is also checked if the path containing server entries and points to the location where the application is installed are correct. [MLCOMz]

Container applications that permit linking to embedded objects, server applications and container/server applications installed on the system must sign on their data into the registry. [MLCOMaa]

It is very time-consuming to check each component if it supports the interface. To solve that problem component categories are installed which are allocated with a GUID named CATID (Category ID). Components that support each interface of a component category could register themselves as member of that category. It is also possible that a component supports various categories [MLCOMab].

### **2.6.1. The Registry Editor<sup>16</sup>**

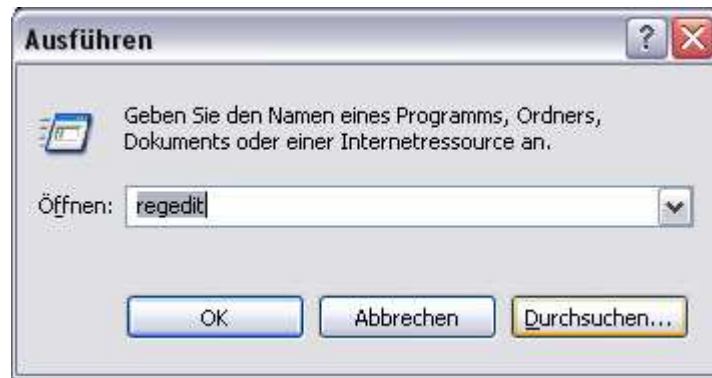
With the Registry Editor it is possible to edit and view the registry.

Dependent on the operating system the file `regedt32.exe` (Windows NT) or `regedit.exe` (Windows 95, Windows 98 and Windows XP) is used. Open the Start menu, select Run and insert `Regedit` on a Windows XP machine to launch the Registry Editor. The editor contains information about the registry and its organization including the CLSID's. The CLSID's are presented in the form `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx}`.

Figure 4 shows the "Run" window of a Windows XP machine where the file name of the Registry Editor is inserted.

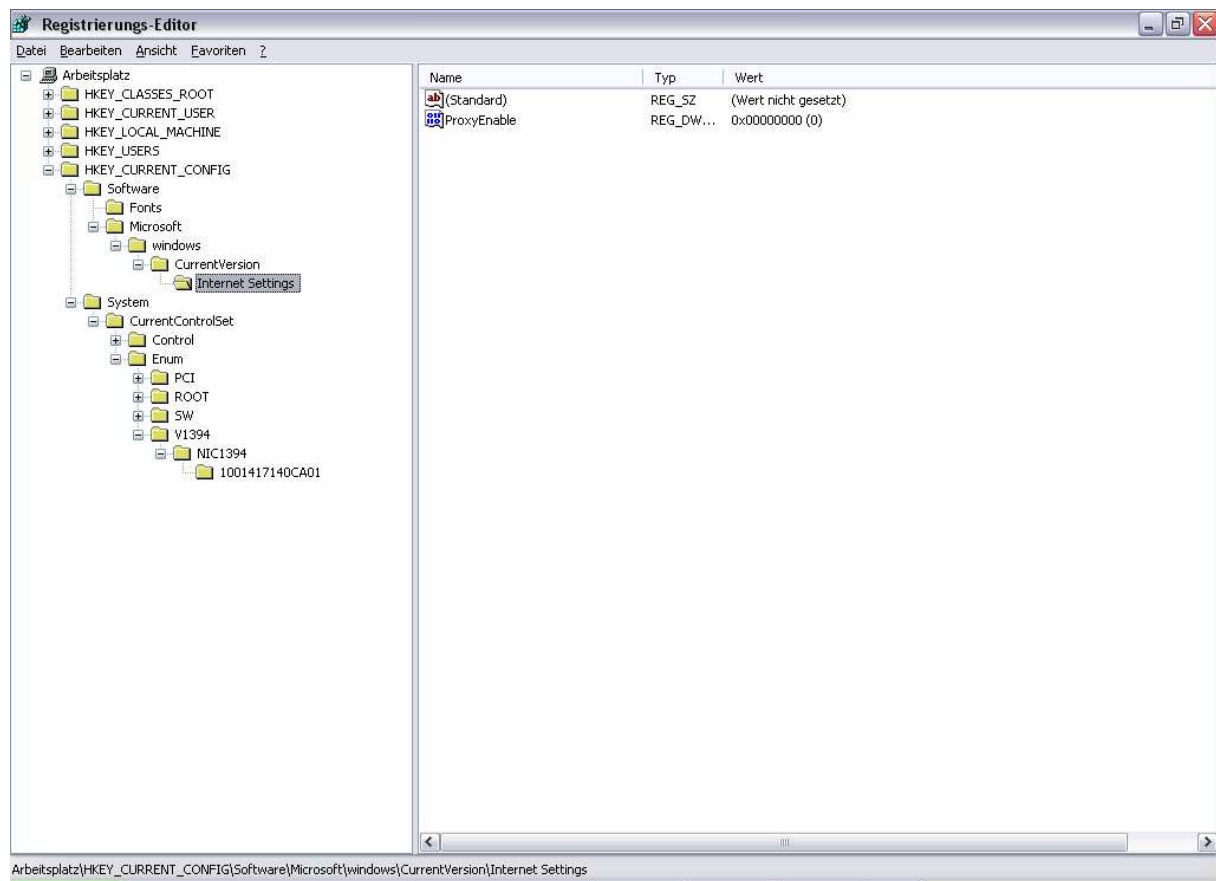
---

<sup>16</sup>This section uses [MLCOMac]



**Figure 4:** Snapshot of the window of Start->Run<sup>17</sup>

The figure 5 contains a snapshot of the Registry Editor with some opened keys.



**Figure 5:** Snapshot of the Registry Editor<sup>18</sup>

<sup>17</sup> Part of MS Windows XP

<sup>18</sup> Part of MS Windows XP

## 2.6.2. The OLE/COM Object Viewer<sup>19</sup>

Another more comfortable possibility to watch the registry is the OLE/COM Object Viewer. It shows the CLSIDs and the COM classes in the registry on the machine.

COM classes and other settings can be configured including DCOM and security settings. The COM classes can also be tested, activated remotely or locally and the supported interfaces can be shown. The content of the type library can be seen. The OLE/COM Object Viewer can be downloaded from the Microsoft Homepage<sup>20</sup> and installed on the machine. The figure 6 shows the OLE/COM Object Viewer with the `_Workbook` interface of Excel with interface, CLSID and Type Library information.

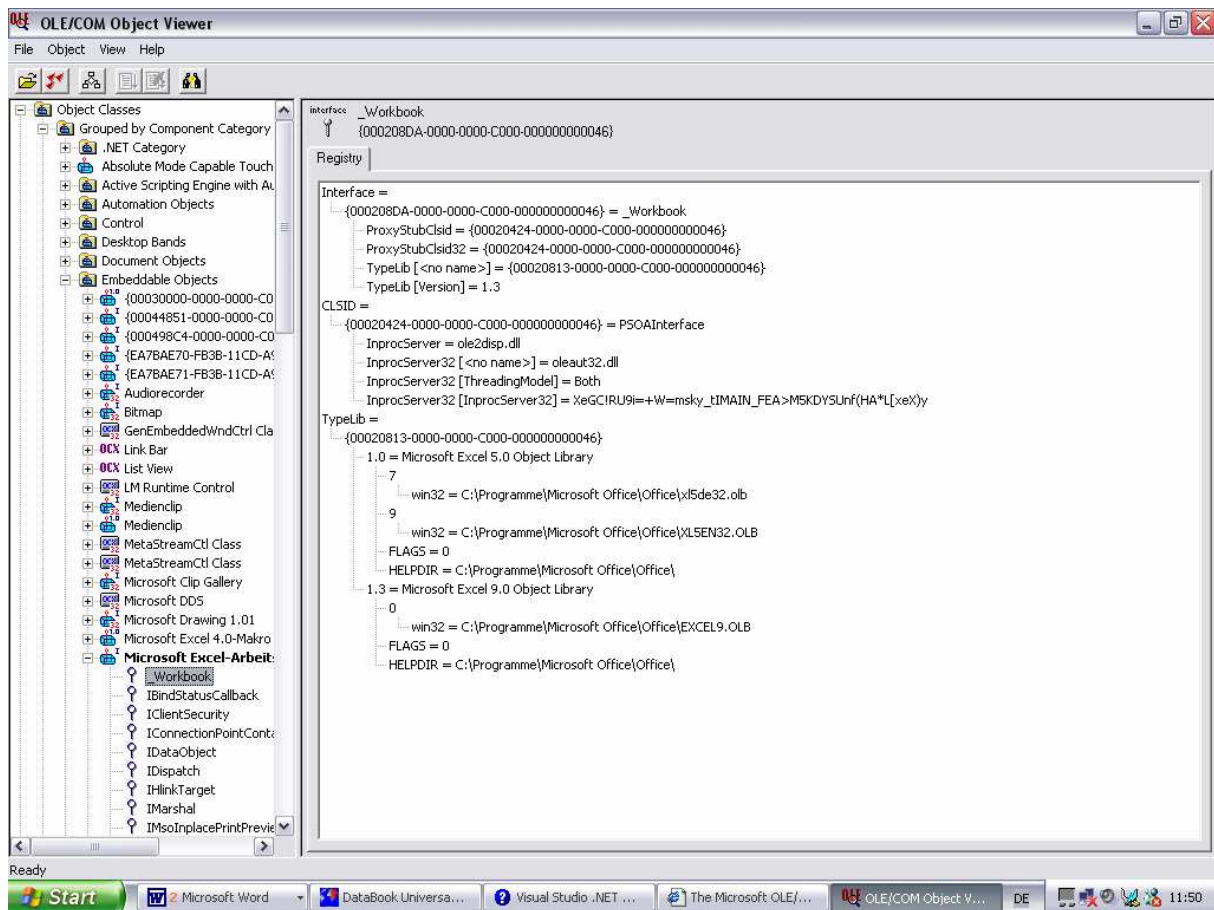


Figure 6: Snapshot of OLE/COM Object Viewer<sup>21</sup>

<sup>19</sup> This section uses [MS98]

<sup>20</sup> <http://www.microsoft.com/com/resources/oleview.asp>

<sup>21</sup> Can be downloaded from the Microsoft Homepage

### 2.6.3. The WindowsRegistry Class<sup>2223</sup>

The WindowsRegistry class enables Object Rexx to modify and query the Windows registry and to delete or add items. Code 1 demonstrates some features of the WindowsRegistry Class.

First the Registry object is created. Then the handles of the root key and of the current key are shown. The key "TestKey" is generated in the key HKEY\_CURRENT\_USER with the Create method and some values and types are defined with the SetValue method. After that these values and types are taken with the GetValue method and its suffixes Data and Type. The DeleteValue method deletes the named value "Name1" and the Delete method erases the whole key "Testkey". At least the directive `::REQUIRES Winsystem.cls` is needed because the WindowsRegistry class is not a built-in class and the WindowsRegistry class information is therein contained.

```
-----
-- WindowsRegistryClass.REX --
-----

reg = .WindowsRegistry~New -- Creation of the registry object
  SAY "The handle of the root key is: " reg~CLASSES_ROOT
  SAY "The handle of the current key is: " reg~CURRENT_KEY
  SAY "-----"
reg~Create(reg~Current_User,"TestKey") -- The key "TestKey" is created
reg~Setvalue(,"","Keyvalue") -- Default value
  -- Other values are added
reg~Setvalue(,"Name1","0","BINARY")
reg~Setvalue(,"Name2","1234","NUMBER")
reg~Setvalue(,"Name3","Value3","EXPAND")
  -- MessageBox
CALL RxMessageBox "Start the Registration Editor to watch the " -
  "created entries!", "Information", "OK", "INFORMATION"
  -- Show the values and types
Stem. = reg~Getvalue(,"")
  SAY "Default value:" stem.data
  SAY "Default type: " stem.type
Stem. = reg~Getvalue(,"Name1")
  SAY "Name1 value:" stem.data
  SAY "Name1 type: " stem.type
```

<sup>22</sup> This section uses [IBM01, p249ff]

<sup>23</sup> This section uses [IBM01a]

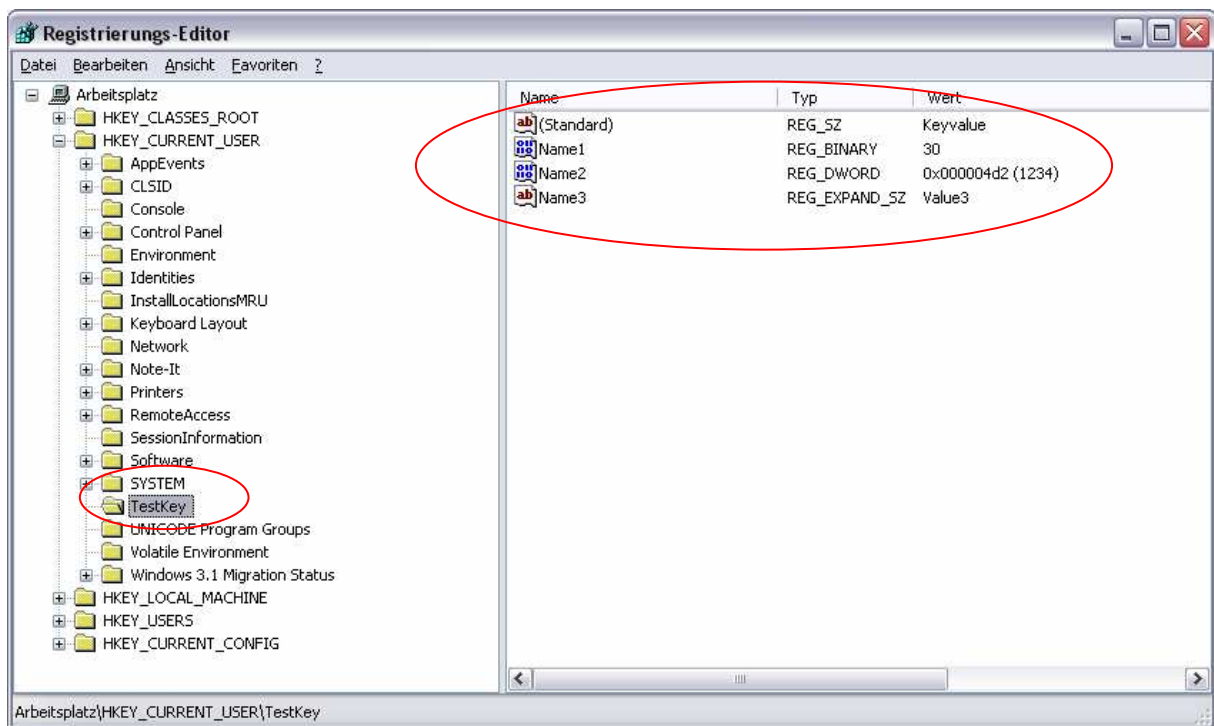
```

Stem. = reg~Getvalue(,"Name2")
  SAY "Name2 value:" stem.data
  SAY "Name2 type:" stem.type
Stem. = reg~Getvalue(,"Name3")
  SAY "Name3 value:" stem.data
  SAY "Name3 type:" stem.type
reg~DeleteValue(,"Name1") -- Name1 is deleted
  -- MessageBox
CALL RxMessageBox "Name1 is deleted!", "Information", "OK", "INFORMATION"
reg~Delete(reg~Current_User,"TestKey") -- The Key "TestKey" is deleted
  -- MessageBox
CALL RxMessageBox "The Key TestKey is deleted!", "Information", "OK", -
  "INFORMATION"
::REQUIRES "winsystem.cls" -- Loads the WindowsRegistry class definition

```

**Code 1: WindowsRegistryClass.REX<sup>24</sup>**

Figure 7 shows the Registry Editor with the new created key “Testkey” (small circle) and its values (big circle).



**Figure 7: Snapshot of the Registry Editor with the new key “TestKey”<sup>25</sup>**

<sup>24</sup> Modelled after [IBM01a]

<sup>25</sup> Part of MS Windows XP



## 2.7. Distributed COM (DCOM)<sup>26</sup>

DCOM is an extension of COM. It enables data transfer between objects which are located on machines which are distributed in a LAN, WAN or over the Internet. This section contains comments on DCOM, CORBA, and COM components in different processes, to COM components on different machines and to the features of DCOM.

Distributed COM supports the communication of objects in networks like the Internet, WAN or LAN. It adds on COM and extends it. DCOM is available for Windows, Apple Macintosh and all major UNIX platforms. The ActiveX Consortium manages DCOM.

DCE RPC (Distributed Computer Environment – Remote Procedure Call) is the fundament of DCOM. In this way, DCOM is able to be adapted to other DCE RPC platforms. Virtual machine environments like Java or platform-neutral development frameworks can be integrated with DCOM .

### *CORBA and DCOM*

The Common Object Request Broker Architecture (CORBA) is a further standard for distributed object computing. It has an abstract object model with components and interfaces. Like DCOM it has standard mappings from the abstract object definition to programming languages.

The so-called Object Request Broker of CORBA transmits and supervises messages between the objects. CORBA is unlike DCOM platform-independent. Both models can be used as server-side component model [Br02].

### 2.7.1. COM Components in different Processes<sup>27</sup>

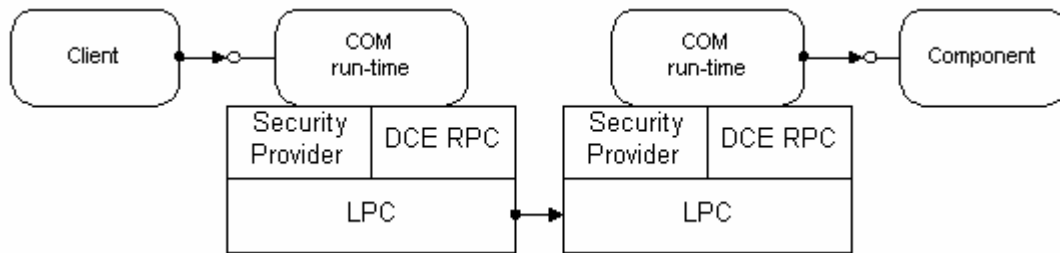
If a client wants to communicate with a component in another process, COM intercepts the call and passes it on to the other process according to the security provider and DCE RPC with a LPC (Local Procedure Call).

---

<sup>26</sup> This section uses [MLDCOa]

<sup>27</sup> This section uses [MLDCOa]

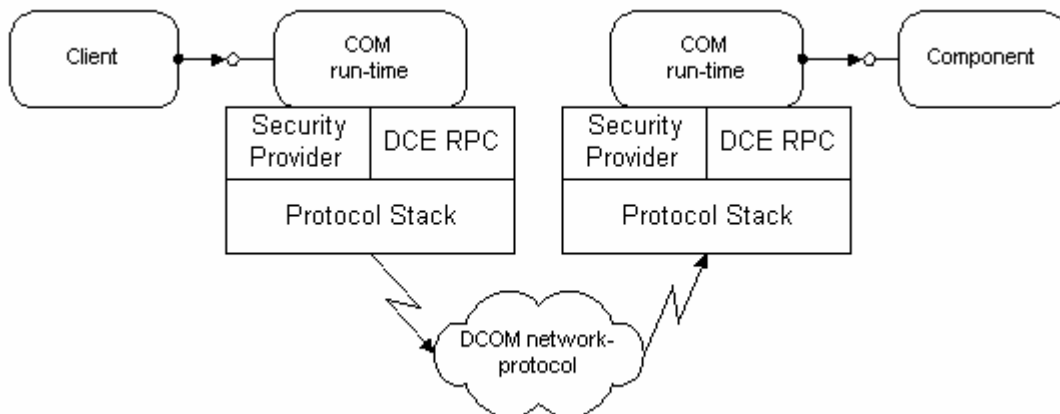
This is illustrated in figure 8.



**Figure 8:** COM components in different processes<sup>28</sup>

### 2.7.2. COM Components on different Machines<sup>29</sup>

In the case of different machines for the client and the component, DCOM communicates with a network protocol by using a protocol stack. This is illustrated in figure 9.



**Figure 9:** DCOM: COM components on different machines<sup>30</sup>

### 2.7.3. Features of DCOM<sup>31</sup>

COM tools and components can be used for DCOM and reduce the expense on development. It is also possible that components developed for distributed use can be reused in the future. This section describes the features of DCOM.

<sup>28</sup> Taken from [MLDCOa]

<sup>29</sup> This section uses [MLDCOa]

<sup>30</sup> Taken from [MLDCOa]

<sup>31</sup> This section uses [MLDCOa]

DCOM ensures that a client connects to a component and it does not matter if the component is in the same process or anywhere in the Internet. There are no changes in the source code and it is not necessary that the program is recompiled.

Languages like Java, C, Delphi or Basic can be used for DCOM. Unfortunately DCOM is not supported by Object Rexx [IBM01b,p154] and there is actually (December 2002) no planning to change this [Doe02].

To check if a client is still active, DCOM uses a pinging protocol. The machine of the client sends a periodic message and if there are three ping periods without receiving a ping message the connection is interrupted. The reference count is reduced to zero and the component is freed.

DCOM enables unidirectional and interactive symmetric communication between peers and between servers and clients.

DCOM offers scalability, which means the possibility for a distributed application to grow if data and the number of users grow. If the number of users and the amount of data is reduced, the distributed application could be small.

DCOM uses UDP (User Datagram Protocol) as transport protocol, which is part of the TCP/IP (Transmission Control Protocol/Internet Protocol). There are also other protocols supported like IPX/SPX (Internet Packed Exchange/Sequenced Packed Exchange) or NetBIOS. To reduce the number of network round trips DCOM supports batching to bundle several method calls.

If the number of users and data grows, possibly it is necessary to distribute the load among multiple server machines. That is called load balancing. It is called static load balancing if there are always the same users who run on the same machine the same application. DCOM's location independence makes it possible that different servers are chosen for different users. Another kind of load balancing is the dynamic load balancing. That means that the referral component uses information about statistics, network topology and server load to allocate transparently the client to the most adequate server.

## 2.8. COM+ (Component Service)<sup>32</sup>

COM+ is a development of the Microsoft Transaction Server (MTS) (technology for distributed applications [MLMTSa]) and the Component Object Model. Content of this section are the new features of COM+ and the new features with regard to the creation of applications.

COM+ is based on MS Windows 2000 and able to program distributed, enterprise-wide and mission-critical applications, which can also run on Windows 95, Windows 98 and Windows NT systems. DCOM is flowed into COM+ and both are then not supported by Object Rexx and there is actually no planning to change this [Doe02].

### 2.8.1. New Features of COM+<sup>33</sup>

With installation of COM+ new features are offered which are listed below:

- COM+ library or server applications can be enabled and disabled and COM+ server applications can be paused and resumed.
- COM+ supports COM+ partitions. With this technique, it is possible that various versions of COM+ applications are configured and installed on the same machine.
- Private components can be created so that private applications cannot be accessed from outside the application. Nevertheless, they still take part in all COM+ services.
- Application Recycling enables the automatically shutting down of a process and restarting it if there are problems with this process.
- COM+ enables components to be moved and copied, which means that a single physical implementation of a component can be configured for many different times.
- COM+ application recycling integrates with COM+ application pooling service and adds scalability for single-threaded processes.

---

<sup>32</sup> This section uses [MLCO+a]

<sup>33</sup> This section uses [MLCO+b]

- Corresponding to performance, scalability, helping to increase concurrency and need, an application's isolation level can be configured.
- COM+ allows its applications to be implemented as NT service. In this way the application's dependent services can be started automatically, the COM+ application can run as local system and the server can be automatically started or restarted.
- The Component Services checks if there is enough memory before creating an object to improve the reliability.
- Process dumping allows an administrator to dump a process without ending the process.

### **2.8.2. Features for Creating Applications<sup>34</sup>**

COM+ offers new features for creating distributed, component-based applications.

- COM+ Events is a system with so-called subscribers and publishers. The subscribers are COM+ objects that run the methods on the event interface. The publishers are COM+ objects calling an event object.
- With COM+ transactional multi-tier applications can be created.
- COM+ allows execution of objects on any thread type. This new model is called neutral threading.
- All Microsoft Transaction Server (MTS) 2.0 semantics are supported by COM+.
- With object pooling objects can be pooled and generated by an application according to applications requirements.
- Instead of the system registry, the registration database (RegDB) is installed. The scriptable and transactional interface COM+ catalog accesses the RegDB.

---

<sup>34</sup> This section uses [MLCO+c]

- With the COM+ queued components service components can be executed instantaneously if server and client are linked or the implementation is suspended until there is a link.
- Process access permission security and role-based security are supported by COM+

## 3.ActiveX

This section discusses ActiveX with its history, the meaning of linking and embedding, the features of ActiveX and ActiveX Controls. The basics of Object Linking and Embedding, which is the basis of OLE Automation, are explained.

ActiveX is another name for OLE (Object Linking and Embedding) [En01,p.7]. ActiveX is based on COM. COM and OLE offer the possibility for interacting of different objects programmed by different people. OLE is an object-based technology [MLOLEa].

### 3.1. History of OLE/ActiveX<sup>35</sup>

In the 1980s Microsoft developed the dynamic data exchange (DDE) protocol to ease the creation of compound documents. A compound document contains data of different formats like spreadsheets, bitmaps, text or sound clips, generated by different applications. Later in 1991 the DDE was extended and became to OLE version 1.0. The objects were linked or embedded to reference them. The objects are a kind of software component that can be integrated in an application to enlarge its functionality. With version 2.0 (1993) OLE was improved and was given a huge infrastructure to sustain the component software. In 1996 ActiveX was introduced to use interactive software that is Internet-enabled [MS96]. The figure 10 contains a timeline of ActiveX.

---

<sup>35</sup> [MLOLEa]

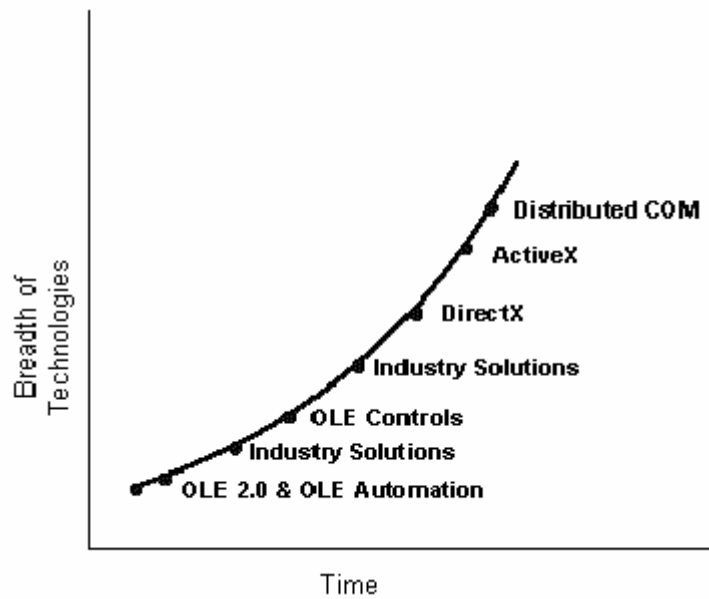


Figure 10: Timeline of ActiveX<sup>36</sup>

## 3.2. Object Linking and Embedding<sup>37</sup>

OLE makes it possible that different elements and different programs are linked in a single document. This could be elements like text, pictures, tables or sounds. The sections 3.2.1. to 3.2.3. were taken from [He02,p5f].

There are two methods: Embedding and Linking.

### 3.2.1. Linking<sup>38</sup>

Information can be linked with other files. All changes in the source document can be transferred automatically to the target document.

Disadvantage: There is a gap in the target document if data in the source file is erased.

Advantage: Changes of data in the source file are immediately transferred to the target document.

<sup>36</sup> Taken from [MLOLEa]

<sup>37</sup> Taken from [He02,p5]

<sup>38</sup> Taken from [He02,p5]



### 3.2.2. Embedding<sup>3940</sup>

An embedded object is a copy from information of the source file that is inserted into the target document. Data and document are linked fix. The application which contains the embedded data can start the source application for working with the data via OLE. Afterwards the worked data is saved [Fla03f].

Advantage: Data is part of the target document.

### 3.2.3. The Class ID (CLSID) of the OLE object<sup>41</sup>

Each application supporting OLE can be identified with its class ID. For example the CLSID of the Microsoft Internet Explorer (Version 5.00.2014.0216) is:

"{0002DF01-0000-0000-C000-000000000046}"

Alternatively, the Program ID can be used:

ProgID of Microsoft Internet Explorer: "InternetExplorer.Application".

### 3.2.4. Features of ActiveX

An application that can embody linked or embedded objects into its own document is called a container application [MLGLOb]. A container application is used to access and store compound documents [MLAUTa]. A component or server application is used to generate OLE document components for container applications [MS03].

OLE allows data transfer with the clipboard's Copy&Paste and with drag and drop [MLOLEb]. The Uniform Data Transfer (UDT) is the presupposition for drag and drop, clipboard and Automation and replaced DDE [MLGLOc].

Copy&Paste or drag and drop are used to insert data from a server application to a container application. Server and container communicate through the OLE system dynamic-link library (DLL). It is also possible that an application is a container and a server at the same time [MS03].

---

<sup>39</sup> Taken from [He02,p5]

<sup>40</sup> Section uses [Ar03]

<sup>41</sup> Taken from [He02,p6]

There are three primary type specifications: [MLOLEa]

- Creation and management of custom controls is described with OLE Controls.
- For scripting and programmability, there is the OLE Automation specification. Automation controller and Automation object are there explained.
- For the management and the generating of compound documents, there are OLE Documents. The creation of containers and embeddable or linkable objects is there explained.

With OLE Automation an application can be driven by another application. OLE enables development, design and deployment of component software and asynchronous and decentralized innovation [MLOLEa].

### **3.3. ActiveX Control**

This chapter gives an overview of ActiveX Control. Thereby the interfaces are listed and explained and features like digital signature and certification, design-time licensing, run-time licensing, initialization security, compression, self-registration and licensing are discussed.

An ActiveX Control is an OLE control or respectively a former OCX control with extra characteristics [MLAXCa]. Integrated in a Microsoft Internet Explorer an ActiveX Control can improve Web pages with features like text boxes or buttons. HTML pages can be automated with the properties, events and methods provided by controls [MLAXCb]. With the HTML `<OBJECT>` tag ActiveX Controls can be inserted to web pages [MLAXCa].

ActiveX Controls can also be installed in applications programmed in many languages. In this way it is possible to reuse packaged functionality. With Visual C++ and one of the ActiveX Control frameworks like BaseCtl framework, the ActiveX Template Library (ATL) or Microsoft Foundation Class Library (MFC), controls can be written [MLAXCa].

### 3.3.1. Interfaces<sup>42</sup>

This part provides an overview over important interfaces that are used for ActiveX controls. A table gives a short description about the interfaces.

An OLE control is a COM object. In this way it supports the interfaces `IUnknown`, `IClassFactory` and the `IClassFactory2`. The `IClassFactory` interface is used to register a class in the system registry and must be implemented [MLCOMad]. The `IClassFactory2` interface is optional and makes it possible that object creation can be controlled by a class factory object through licensing. It is an extension to the `IClassFactory` interface [MLCOMae]. Normally, an OLE control supports other interfaces that enable, for example, the support for a user interface for the control, the support of Automation or the writing of persisting information to disk.

The table 1 lists the interfaces that are also supported by an OLE control [MLAXCb]:

<code>IConnectionPointContainer</code>	Responsible for connection points for connectable objects [MLCOMaf].
<code>IDataObject</code>	Notification of changes in data and data transfer [MLCOMag].
<code>IDispatch</code>	Access to the control's methods and properties.
<code>IExternalConnection</code>	Support of external connections.
<code>IoleCache2</code>	In order to cache a control's data, this interface supports the functions that a container calls.
<code>IoleControl</code>	This interface is used to support ambient properties and mnemonics.
<code>IoleInPlaceObject</code>	Responsible for the in-place activation. In-place activation means the skill to activate an object from within an OLE control and to link a verb

---

<sup>42</sup> [MLAXCb]

	like <code>Edit</code> or <code>Play</code> with that activation. [MLAUTb]
<code>IOleObject</code>	This interface is used for the communication between control and container.
<code>IPerPropertyBrowsing</code>	Access of the data in the property pages provided by an object.
<code>IPersist*</code>	Provide six interfaces that allow that a control can write or read its persistent data to stream, file or storage. The “*” is a place-marker for the six different <code>IPersist</code> interfaces.
<code>IProvideClassInfo2</code>	Allows getting a pointer to the type information of the control.
<code>IRunnableObject</code>	Determination whether a control is in a "loaded" and a "running" state.
<code>ISpecifyPropertyPages</code>	Offers a list of property page CLSID's supported by the object [MLAUTbf].
<code>IViewObject2</code>	Allows a container to render a control.

**Table 1: OLE Control interfaces<sup>43</sup>**

COM objects which can save their internal state are in a so-called “persistent state”. Objects with a “persistent state” must implement one `IPersist*` interface. Either `IPersistStreamInit` or `IPersistStream` must be implemented.

The first interface is employed when a control wants to know when it is generated new as opposed to reloaded from an existing persistent state. The second interface has not this “generated new” ability [MLAXCc].

Controls do not need properties or methods and in this case, they do not need to implement the `IDispatch` interface. If a control does have any methods or

<sup>43</sup> Modeled after [MLAXCb]

properties, then there are no requirements for which methods or properties a control must expose [MLAXCd].

It is not necessary for controls to have events. If there are not any events then it is not necessary to implement the `IConnectionPointContainer` interface [MLAXCe].

### **3.3.2. Further Characteristics**

This chapter discusses further characteristics of ActiveX Controls. These characteristics are licensing, initialization security, digital signature and certification, compression and self-registration.

#### **3.3.2.1. Licensing<sup>44</sup>**

There are two kinds of licensing for the most ActiveX Controls, run-time licensing and design-time licensing.

##### **3.3.2.1.1. Design-Time Licensing**

Design-time licensing guarantees that the application or web page is built with a legally acquired control. Control containers like Microsoft Access or Visual Basic verify controls by calling `IClassFactory2::CreateInstanceLic` [MLAXCf] and allow a developer to place a control in an application or web page after the control is licensed.

##### **3.3.2.1.2. Run-Time Licensing**

Run-time licensing guarantees that the application or a web page contains a legally acquired control. Control containers also call functions in the control to confirm the license. Therefore the `IClassFactory2` interface is needed. Run-time licensing can be used with a HTML page by using the `Object` object to include the so-called license package file (LPK).

---

<sup>44</sup> This section uses [MLAXCb]

### **3.3.2.2. Initialization Security<sup>45</sup>**

There is a potential security hazard because a control can receive data from an untrusted source. That is because it is possible that a control can obtain data from any `IPersist*` interface if the control is initialized.

There are two possibilities to ensure that an ActiveX Control is safe for initialization.

The first possibility is to work with the interface `IObjectSafety`. If the control includes the `IObjectSafety` interface then the Internet Explorer calls a method named `IObjectSafety::SetInterfaceSafetyOptions` before loading the control to check the security of the initialization.

The second possibility is to use the Component Categories Manager to create the correct entries in the system registry. The registry is probed by the Internet Explorer before loading the control, whether these entries exist.

Another problem solved in the same way is the scripting security. The problem is that a control that is known to be safe is not safe when a non-reliable script automates it. An example is the MS Office with its Automation model that can be abused to delete files on a machine.

### **3.3.2.3. Compression<sup>46</sup>**

ActiveX Controls can be downloaded faster over an intranet or the Internet with data-compression. Therefore the `.cab` file format is used. Foundation for this non-proprietary compression format is the Lempel-Ziv-compression algorithm.

### **3.3.2.4. Self-Registration<sup>47</sup>**

An ActiveX Control is also a Component Object Model (COM) object. In this way it supports the `IUnknown` interface and is self-registering. With the functions `DllRegisterServer` and `DllUnregisterServers` ActiveX Controls support

---

<sup>45</sup> [MLAXCb]

<sup>46</sup> This section uses [MLAXCb]

<sup>47</sup> This section uses [MLAXCb]

self-registration. To register as control the Component Categories API (application programming interfaces) must be used [MLAPIa].

### **3.3.2.5. Digital Signature and Certification<sup>48</sup>**

Microsoft Authenticode Technology allows digital signatures and digital certification for ActiveX Controls. With a digital signature a unique public key and the software vendor's name are linked with the file which contains an ActiveX object.

---

<sup>48</sup> This section uses [MLXCB]

## 4.ActiveX Automation

This part discusses the technical background of the Automation technology that is used in sections 6 and 8 to 13. Here are individualized the ActiveX client, the ActiveX object, the interaction of objects and clients, the exposing of ActiveX objects, the design of an application which is automated and the access of ActiveX objects.

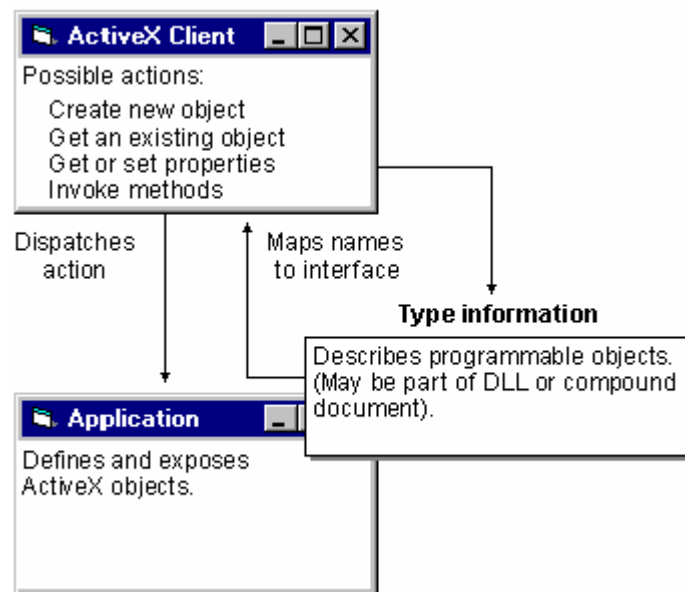
With ActiveX Automation it is possible for applications to expose their functionality to interpreted and scripting languages. It can be used on Windows 95, Windows 98, Windows SE, Windows Me, Windows 2000 and Windows XP systems [MLAUTc].

Automation allows the changing and creation of objects that are exposed in one application from another application, the creation of tools that can contain compilers, object browser, external programming tools and macro languages to change and access objects, and the creation of programming tools and applications that expose objects [MLAUTd].

- *ActiveX objects* are exposed objects of programming tools or applications.
- *ActiveX clients* are programming tools or applications that use the ActiveX objects.
- *ActiveX components* are e.g. DLL or EXE files which contain classes, which define the ActiveX objects. The exposed objects are described by type information. Type information can be accessed either at run time or at compile time by the ActiveX components [MLAUTd].



Figure 11 illustrates the relation among ActiveX client, the application and type information.



**Figure 11:** Relations among ActiveX objects and ActiveX clients<sup>49</sup>

## 4.1. ActiveX Client

ActiveX clients are programming tools or applications that access and use ActiveX objects or create new ActiveX objects.

The client implements properties and methods of the object and generates new instances of the object. It is possible that the object occurs in another or in the same application. Clients can be generated by writing a new application capable of Automation, by redesigning a present programming tool to extend it for Automation or by writing code in an application that allows implementation of another application's objects through Automation. An ActiveX client is for example a programming tool like Visual Basic or Object Rexx [MLAUTe].

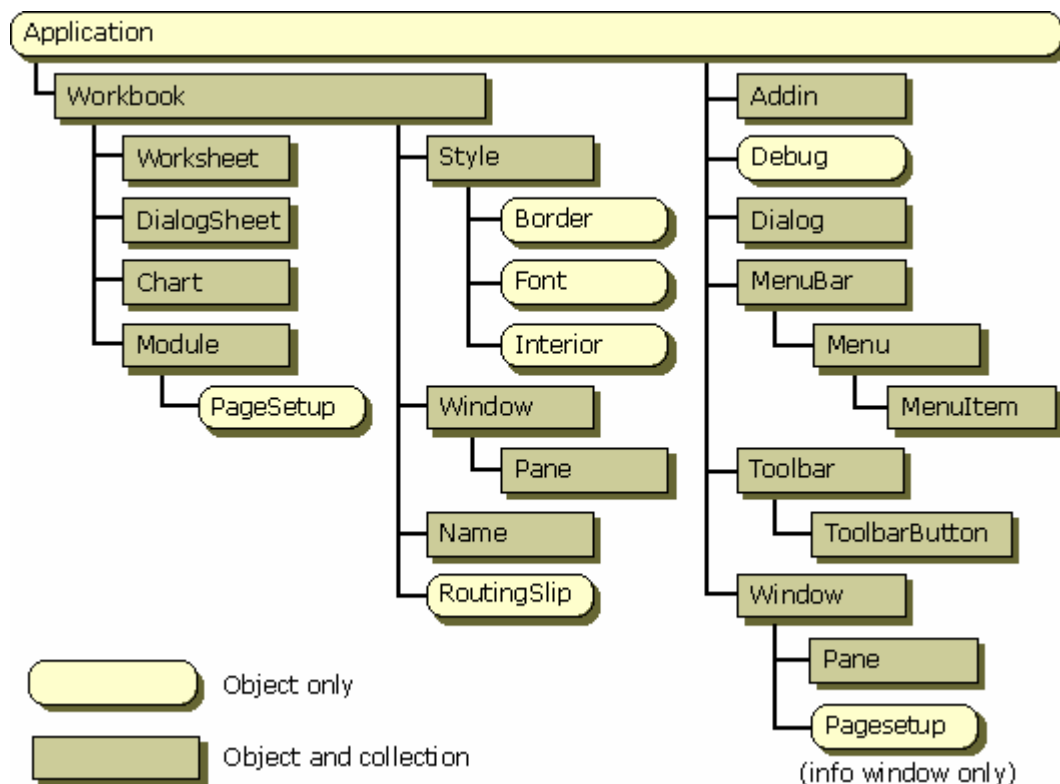
## 4.2. ActiveX Object

The member functions events, properties and methods are exposed to ActiveX clients by an ActiveX object.

<sup>49</sup> Taken from [MLAUTd]

ActiveX objects support COM. The member functions of an ActiveX object make the object programmable by ActiveX clients. Activities that an object can execute are called *methods*. *Properties* allow using data about the state of an object. Activities acknowledged by an object are *Events*. Several instances of an object are together a *collection object*. If there are several instances of an object, all instances can be addressed with the collection object.

Figure 11 shows a part of the object model of Microsoft Excel [MLAUTf].



**Figure 12:** Some objects of MS Excel<sup>50</sup>

The object names can remain consistent in the further versions of an application. The objects can be used from any macro language or programming tool that implements Automation. In this way, it is possible for system integrators to select a suited macro language or programming tool. Exposed objects from many applications are available for developers to create solutions that extend over applications [MLAUTg].

<sup>50</sup> Taken from [MLAUTf]

## 4.3. Important Interfaces

Table 2 lists important interfaces for OLE Automation.

IClassFactory	This interface must be implemented for every class that is registered in the system registry and to which a CLSID is assigned, so objects of that class can be created. Taken from [MLAUTaw]
ICreateErrorInfo	Error information is set with this interface. Taken from [MLAUTav]
ICreateTypeInfo	This type information interface provides the tools for creating and administering the type information defined through the type description. Taken from [MLAUTau]
ICreateTypeInfo2	This is a type information interface which adds methods for erasing items that have been added with ICreateTypeInfo. Taken from [MLAUTau]
ICreateTypeLib	Supplies methods for managing and creating the component or file that includes type information. Taken from [MLAUTau]
ICreateTypeLib2	Used for the administration and creation of type descriptions and type libraries. Taken from [MLAUTau]
IDataObject	Used for specification of methods that enable data transfer and notification of changes in data. Taken from [MLAUTax]
IDispatch	Supplies a late-bound mechanism to access and retrieve information about the properties and methods of an object.

	Taken from [MLAUTae]
IEnumVariant	Provides a standard way for ActiveX clients to iterate over collection objects <sup>51</sup> . Taken from [MLAUTaj]
IErrorInfo	Gives back information from an error object. Taken from [MLAUTav]
IFontDisp	Exposes a font object's properties through Automation. Taken form [MLAUTbd]
IOleClientSite	Primary means by which an embedded object gets information about the extent and location of its moniker, its display site, its user interface, and other resources which are provided by its container. Taken from [MLAUTbc]
IOleContainer	Enumerates objects in a compound document or locks a container in the running state. Taken from [MLAUTbb]
IPicture	Manages a picture object and its properties. Taken from [MLAUTba]
IPictureDisp	Exposes the picture object's properties through Automation. Taken from [MLAUTaz]
IRecordInfo	Describes the structure of a particular UDT (user-defined type). Taken from [MLAUTbe]
IServiceProvider	Generic access mechanism to locate a globally unique identifier (GUID) identified service. Taken from [MLAUTay]
ISupportErrorInfo	Identifies an object as supporting the IErrorInfo interface. Taken from [MLAUTav]
ITypeComp	Provides type lookup and binding methods. Taken from [MLAUTat]

---

<sup>51</sup> c.p. 4.6.6.

ITypeInfo	This interface reads the type information within the type library. Taken from [MLAUTat]
ITypeInfo2	This interface offers additional type information retrieval actions. Taken from [MLAUTat]
ITypeLib	Gets back information about a type library. Taken from [MLAUTat]
ITypeLib2	Supplies further type library retrieval actions. Taken from [MLAUTat]
IUnknown	The IUnknown interface defines three member functions (QueryInterface, AddRef and Release) that must be implemented for each object that is exposed. Taken from [MLAUTad]

**Table 2:** Important interfaces for OLE Automation.

## 4.4. Interaction of Objects and Clients<sup>52</sup>

The interaction of objects and clients is a notable area of Automation. Part of this area is different items like VTBL, dual interface, object access with the IDispatch<sup>53</sup> interface, ID binding, late binding, object access with the VTBL, in-process Servers and out-of-process Servers.

ActiveX clients in two ways can use objects. They can be accessed by implementing the member functions like the properties and methods directly in the *virtual function table* (VTBL) of the object. They can also be accessed with the IDispatch interface that derives from the IUnknown interface. Additionally there is the possibility to use both procedures together. This is called a *dual interface*.

So-called *Custom interfaces* are user-defined interfaces and they are COM interfaces that are not defined as part of OLE.

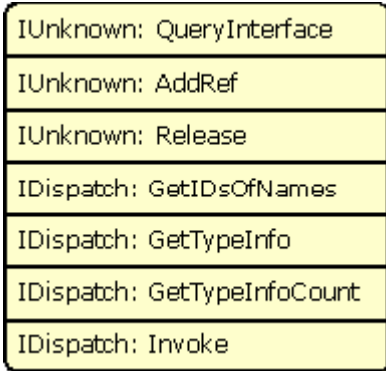
<sup>52</sup> [MLAUTH]

<sup>53</sup> c.p. 4.3.

With a type library or an `IDispatch` interface, the interfaces, with its members, an object can use, can be seen by a programming tool or an ActiveX client.

All methods and properties of an object and the supported interfaces are itemized in the VTBL. The three member functions of the `IUnknown` interface<sup>54</sup> are the first three entries in the VTBL. Afterward the member functions of the other interfaces follow [MLAUTh].

The figure 13 illustrates a VTBL with `IUnknown` and `IDispatch` interface.



IUnknown: QueryInterface
IUnknown: AddRef
IUnknown: Release
IDispatch: GetIDsOfNames
IDispatch: GetTypeInfo
IDispatch: GetTypeInfoCount
IDispatch: Invoke

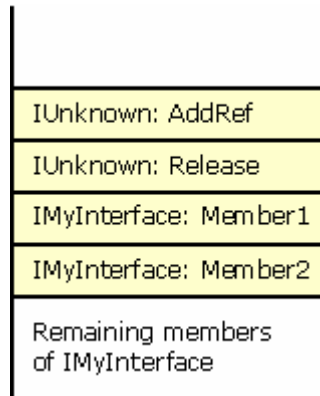
**Figure 13:** VTBL with `IUnknown` and `IDispatch` interface.<sup>55</sup>

---

<sup>54</sup> c.p. 2.1.

<sup>55</sup> Taken from [MLAUTh]

The member functions of the custom interface, in this case called `IMyInterface`, follow the `IUnknown` shown in the figure 14 if an `IDispatch` interface is not supported.



**Figure 14:** Dispatch interface is not supported<sup>56</sup>

#### 4.4.1. Dual Interface<sup>57</sup>

The first three items in a VTBL of a dual interface are functions of `IUnknown`<sup>58</sup>, the next four items are members of `IDispatch`<sup>59</sup> and the following entries are members of the dual interface. Here the dual interface is called `IMyInterface`. Figure 15 demonstrates a VTBL for an object with dual interface.

<sup>56</sup> Taken from [MLAUTH]

<sup>57</sup> [MLAUTH]

<sup>58</sup> c.p. 2.1.

<sup>59</sup> c.p. 4.3.

IUnknown: QueryInterface
IUnknown: AddRef
IUnknown: Release
IDispatch: GetIDsOfNames
IDispatch: GetTypeInfo
IDispatch: GetTypeInfoCount
IDispatch: Invoke
IMyInterface: Member1
IMyInterface: Member2
Remaining members of IMyInterface

**Figure 15:** Accessing an Object Through the IDispatch Interface<sup>60</sup>

#### 4.4.2. Object Access with the IDispatch Interface<sup>61</sup>

To approach an object with the `IDispatch` interface, the object must first be generated. After that, the `IUnknown` interface is queried for a pointer to the `IDispatch` interface.

With the `DISPID` (dispatch identifier) parameters, data members, methods and properties are approached internally [MLAUTbg].

The function `IDispatch::Invoke` enables the use of member functions. Thereby the parameters of the member functions are bundled into the `IDispatch::Invoke` parameters. After that, the parameters are unbundled by the object's implementation of `IDispatch::Invoke`, the member function is executed and errors are managed. Through an `IDispatch::Invoke` parameter, the return value of the object is given back to the client.

#### 4.4.3. ID Binding

*ID binding* is another possibility to obtain the `DISPID`. Thereby the `DISPID` is obtained from the type library during compilation [MLAUTi].

<sup>60</sup> Taken from [MLAUTH]

<sup>61</sup> [MLAUTi]



#### **4.4.4. Late Binding**

Late binding enables the client to obtain the DISPIDs at run time with the `IDispatch::GetIDsOfNames` function. It is only half as fast as ID binding [MLAUTj].

#### **4.4.5. Object Access with the VTBL**

During the compile time the client gets the type information from the type library and calls the functions and methods directly. Because the approach to the member functions is without using the `IDispatch` interface, the VTBL binding is faster than late binding and ID binding [MLAUTj].

#### **4.4.6. Out-of-Process Servers**

Out-of-process servers are executed in a separate process space. They are implemented in an EXE file [MLAUTk].

#### **4.4.7. In-Process Servers**

In-process servers are executed in the process space of their controller. They are stored in a dynamic-link library. It is faster as an out-of-process server [MLAUTk].

### **4.5. Exposing ActiveX Objects**

To access ActiveX objects it is necessary to expose them. This section lists all necessary steps for exposing ActiveX objects. Exposed objects can be accessed by other programming tools and applications [MLAUTj].

ActiveX objects are exposed in several steps. The first is to initialize the objects. Thereby OLE is initialized, the class factories of the exposed objects are registered and the active object is registered. Then the exposed objects are implemented by using the virtual function (VTBL), `IUnknown` and `IDispatch` interfaces and by implementing the methods and properties of the object. When the application ends, OLE is released by cancelling the active object and the registration of the class factories, and by uninitializing of OLE. Therewith other users can use the active objects, an IDL or ODL file with information about the methods and properties of the objects is generated with the Interface Definition Language (IDL), respectively, the

Object Description Language (ODL). Both files are compiled with the Microsoft Interface Definition Language (MIDL) compiler. It is also a registration file generated for the application [MLAUTm].

#### **4.5.1. Initializing of exposed Objects<sup>62</sup>**

To initialize the exposed objects and OLE the functions `OleInitialize`, `CoRegisterClassObject`, and `RegisterActiveObject`, are used.

With `OleInitialize`, the COM library is initialized on the present apartment and the concurrency model is identified as single thread apartment [MLAUTo]. `CoRegisterClassObject`, registers the class factory of the object. Then it can be used by other applications to generate new objects. `RegisterActiveObject` allows other applications to join to a present object by registering the active object.

The object is scheduled in the ROT (running object table) [MLAUTp]. A moniker<sup>63</sup> enables the identification of the objects [MLAUTq]. Thereby it identifies a COM object in the same kind, as a file is identified by a path in the file system [MLGLOd].<sup>64</sup>

---

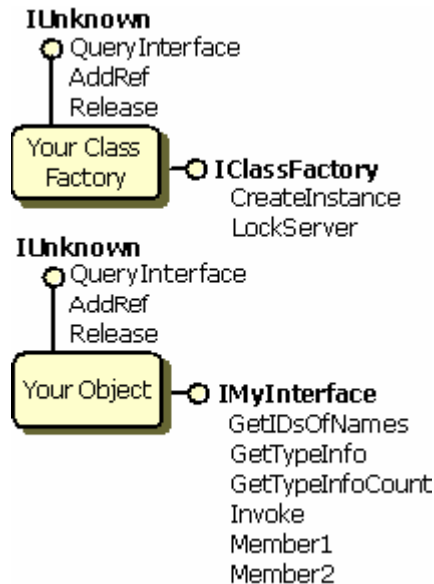
<sup>62</sup> [MLAUTn]

<sup>63</sup> c.p. 2.3.4.

<sup>64</sup> c.p. 2.3.4.

### 4.5.2. Implementation of the exposed Objects

The figure 16 shows the interfaces with their member functions that should be implemented to expose ActiveX objects [MLAUTr].



**Figure 16:** Interfaces that should be implemented to expose ActiveX<sup>65</sup>

### 4.5.3. Implementation of the Class Factory

The class factory of the object is a COM object and it is necessary to generate one or more instances of an object. Therefore, the CLSID and the **IClassFactory** interface are used [MLGLOe].

The **IClassFactory** interface has the two member functions `CreateInstance` and `LockServer` that provide services for OLE API functions. With the `CreateInstance` method an instance of the object's class is generated. The existing instance, listed in the running object table, of the Application object is given back. The `LockServer` method avoids the shutting down of the object's server when the last instance of the object is released. The class factory uses apart from the **IClassFactory** interface also the **IUnknown** interface. In this way it is possible for the client to check what interfaces the object can use [MLAUTs].

<sup>65</sup> Taken from [MLAUTr]

It is possible to generate instances of a class with the class factory. The interfaces of the object are the `IUnknown` and `IMyInterface` interfaces. The `IMyInterface` allows early binding with the VTBL and late binding (`IDispatch`) with the functions `GetIDsOfNames`, `GetTypeInfo`, `GetTypeInfoCount`, and `Invoke`. In this way the dual interface `IMyInterface` offers two kinds of using the member functions. `Member1` and `Member2` can be used to implement the object directly through the VTBL or they can also be used with the `IDispatch::Invoke` function. To handle errors of exposed objects in the case of a dual interface the `IErrorInfo` interface is used [MLAUTs].

Implementation of objects requires also a registrations file (location information for OLE and the operating system) and a type library. The object is described in the library section of an `.idl` file or an `.odl` file is generated. The `.odl` file is compiled with the MIDL compiler. A `.tlb` file (type library) and a header (`.h`) file are generated [MLAUTs]. The header file includes function declarations and type definitions that are based on the interface definition in the IDL file [MLMIDL<sub>a</sub>].

#### **4.5.4. The Application Object**

When a user-interactive, document-based, ActiveX objects exposing application runs, the first object that is initialized should be the top-level `Application` object. With this root object it is possible for the ActiveX client to connect to the application's exposed objects [MLAUT<sub>t</sub>].

#### **4.5.5. Registration**

OLE objects have to be registered with the user's system registration database. This enables the location of remoting code for the interfaces, the location of the type libraries for Automation tools and it enables the generation of instances of the objects with `CoCreateInstance`.

The `DLLRegisterServer` method signs on the type library, the ProgID for each application and the CLSID for each object in the DLL [MLAUT<sub>u</sub>].

The registration connects the ProgID of the application with a CLSID. It is possible to generate instances of the application by name [MLAUT<sub>v</sub>].

*Syntax of the registration file* [MLAUTv]:

```
\AppName.ObjectName[.VersionNumber] = human_readable_string
```

```
\AppName.ObjectName\CLSID = {UUID}
```

AppName describes the name of the application, ObjectName includes the name of the object which should be registered, VersionNumber is not obligatory and contains the number of the version of the object, human\_readable\_string offers a short instruction of the application and UUID contains the universally unique identifier created with Guidgen.exe<sup>66</sup> [MLAUTv].

#### **4.5.6. Registration of Classes<sup>67</sup>**

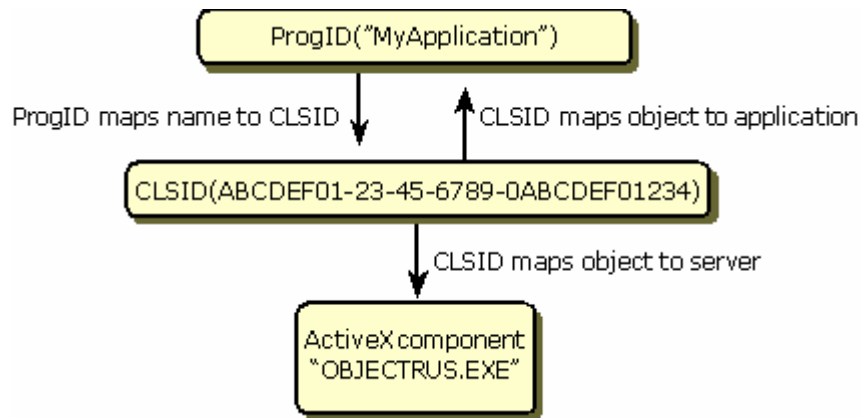
Objects generated with CoCreateInstance must be registered by transferring a CLSID to the Automation component file.

The CLSID maps the ActiveX object to the ProgID and application. Figure 17 shows an ActiveX component with its ProgID and CLSID.

---

<sup>66</sup> c.p. 2.4.2.2.

<sup>67</sup> This section uses [MLAUTw]



**Figure 17: Interaction of ActiveX components, CLSIDs and ProgIDs.**<sup>68</sup>

*Syntax and description of the Registry Entries:*

This `human_readable_string` offers a short instruction of the application. The `human_readable_string` is here `Hello 2.0 Application`.

`HKEY_CLASSES_ROOT\CLSID\F37C8061-4AD5-101B-B826-00DD01103DE1` = `Hello 2.0 Application`

The ProgID is written in the form `AppName.ObjectName.VersionNumber`. Its single parts are described above<sup>69</sup>.

`HKEY_CLASSES_ROOT\CLSID\F37C8061-4AD5-101B-B826-00DD01103DE1\ProgID` = `Hello.Application.2`

The `VersionIndependentProgID` is written in the form `AppName.ObjectName`. The single parts are described above.

`HKEY_CLASSES_ROOT\CLSID\F37C8061-4AD5-101B-B826-00DD01103DE1\VersionIndependentProgID` = `Hello.Application`

`LocalServer[32]` means the usage of an EXE file as ActiveX component which is executed in another process as the ActiveX client. "32" means the usage on 32-bit Windows platforms and is not obligatory. This entry has the syntax `filepath[/Automation]` with name and path of the file which hosts the object. `/Automation` is not obligatory and means that it can be used for Automation. If the ActiveX component is executed in the same process as the ActiveX client and if it is a DLL, instead of `LocalServer[32]` `InProcServer[32]` is used.

<sup>68</sup> Taken from [MLAUTw]

<sup>69</sup> c.p. 4.5.5.

HKEY\_CLASSES\_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\LocalServer32 = hello.exe /Automation

This entry offers the CLSID of the type library:

HKEY\_CLASSES\_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\TypeLib = {F37C8060-4AD5-101B-B826-00DD01103DE1}

This entry shows that it is an ActiveX component:

HKEY\_CLASSES\_ROOT\CLSID\{F37C8061-4AD5-101B-B826-00DD01103DE1}\Programmable

### **4.5.7. Releasing of the exposed Objects and OLE<sup>70</sup>**

The functions that are described in this section are used to release exposed objects and OLE.

`RevokeActiveObject` terminates the active status of an object. OLE is informed with the `CoRevokeClassObject` function that a class factory cannot be longer accessed by other applications. `OLEUninitialize` releases OLE by ending any class factory, the COM library, other COM objects or servers and make RPC impracticable on the apartment [MLAUTy].

### **4.5.8. Retrieving of the Objects**

The following functions enable the retrieving of the objects.

With `RegisterActiveObject` the active object for an application is set when the application is launched. At the termination of an application, the function `RevokeActiveObject` cancels the active object. A pointer to the active object is returned with the function `GetActiveObject`.

The `Application` object is always listed as active and it is possible that there is more than one active object in an application. An active object requires a class factory and that class factory is identified by a `ProgID` in the system registry, registered with the `RegisterActiveObject` function [MLAUTz].

---

<sup>70</sup> This section uses [MLAUTx]

### 4.5.9. The Returning of Objects

The application returns a pointer to the `IDispatch` interface, to return an object from a method or property. The data type is `VT_DISPATCH` in the case if `IDispatch` can be used; otherwise, the data type is `VT_UNKNOWN` [MLAUTaa].

### 4.5.10. Termination of Objects

Here is described how to shut down an object.

An application that is controlled by an ActiveX client and that is visible becomes invisible if the user ends it. Nevertheless, it is still possible to control the object until the application is ended when there is no further external reference to the object. In the case that the application is visible the object ends if the ActiveX client or if the user gives command to shut down. In the case, if the application is invisible the application ends if the last external reference is closed [MLAUTab].

## 4.6. Design of an Application which is Automated

If an application is created which is automated, several items must be taken in consideration. Significant are the `IUnknown` interface, the `IDispatch` interface, the dual interface, the registration of interfaces, the creation of CLSID's and the `IEnumVARIANT` interface.

Generating a programmable interface with its events, methods and properties is like an object-orientated framework for the application [MLAUTac].

### 4.6.1. IUnknown Interface<sup>71</sup>

The prototype for the member functions of the `IUnknown` interface is in the header file `Ole2.h`. With this fundamental interface, it is possible to use objects [MLAUTad].

### 4.6.2. IDispatch Interface

The `IDispatch` interface uses a late-bound mechanism to use information of the member functions of an object. The following functions should also be used:

---

<sup>71</sup> c.p. 2.1.



`GetTypeInfoCount` gives back the amount of type descriptions for the object. `TypeInfo` details the programmable interface. The `GetIDsOfNames` function maps the name of a function to a DISPID. It is possible to use a member function of an object with the function `Invoke` [MLAUTae].

### 4.6.3. Dual Interface<sup>72</sup>

Dual interfaces have advantages over VTBL-only or `IDispatch`-only interfaces. There is better performance for ActiveX clients who use the VTBL interface. Clients using the `IDispatch` interface may carry on their activity. It is possible to bind during run time (`IDispatch`) or during compile time (VTBL) [MLAUTaf].

### 4.6.4. Registration of Interfaces<sup>73</sup>

So that OLE can locate the appropriate remoting code for interprocess communication, an interface must be registered. This data can be watched with the OLE/COM Object Viewer<sup>74</sup>.

*Syntax of the registered information:*

This term describes the universally unique ID and the name of the interface:

```
\Interface\{UUID} = InterfaceName
```

This phrase offers the universally unique ID for the type library with the interface description:

```
\Interface\{UUID}\Typelib = LIBID
```

This entry connects an IID (Interface Identifier) to a CLSID in a 32-bit proxy DLL [MLAUTah]:

```
\Interface\{UUID}\ProxyStubClsid[32] = CLSID
```

---

<sup>72</sup> c.p. 4.4.1.

<sup>73</sup> [MLAUTag]

<sup>74</sup> c.p. 2.6.2.

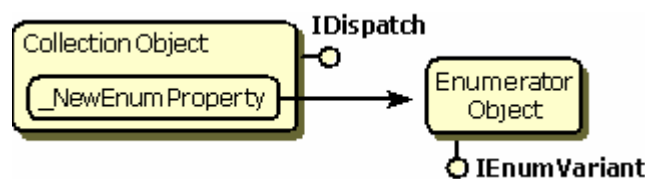
### 4.6.5. Creation of a CLSID

CLSIDs are so-called universally unique identifiers (UUIDs). UUIDs are created with GUIDGen.exe<sup>75</sup>. The CLSID is registered by installing an application. Each object which is exposed for creation has to have a unique CLSID [MLAUTaj].

### 4.6.6. IEnumVARIANT Interface

ActiveX clients can iterate over collection objects with the `IEnumVARIANT` interface.

With the `_NewEnum` property, which sends back an enumeration object that supports the `IEnumVARIANT` interface, the object signals that iteration can be used. With the member function `Skip` of the `IEnumVARIANT` interface one or more parts in a collection are skipped, `Reset` enables that the actual element is reset to the first element in the collection, one or more elements in a collection are retrieved with the `Next` function and the current state of the enumeration is duplicated with `Clone` [MLAUTaj]. The `IEnumVARIANT` interface is illustrated in figure 18.



**Figure 18:** `IEnumVARIANT` interface<sup>76</sup>

The `_NewEnum` property must give back a pointer to the `IUnknown` interface of the enumerator object, it must include `DISPID = DISPID_NEWENUM(-4)` and it has to have the name `_NewEnum` and it is not allowed to be localized [MLAUTak].

## 4.7. Type Library

Type libraries are an essential part for the Automation technology because they store information about one or more ActiveX objects, their characteristics and their interfaces with member functions.

<sup>75</sup> c.p. 2.4.2.2.

<sup>76</sup> Taken from [MLAUTaj]

In this way the objects are available to other programmers. The creation of a type library, the registration of a type library and the error handling are explained.

The objects are written in an object description language file (.odl), and then compiled with the MIDL<sup>77</sup> compiler [MLAUTa].

Exposed objects that use VTBL binding<sup>78</sup> have to be described in a type library, because VTBL references are bound at compile time. Each set of exposed objects must be described in a type library.

The main characteristic of type libraries is its ability to check the type during the compile time. An interface is described with type information. DISPIDs can be stored at compile time to enhance the run time capability by ActiveX clients that cannot use VTBL. Local server access is possible. With the `RegisterTypeLib` function exposed objects can be listed in the registration database. To uninstall an application the `UnRegisterTypeLib` function is used. The library can be seen with type browsers [MLAUTam].

### **4.7.1. Creation of a Type Library**

This section provides information about the creation of a Type Library.

Type libraries are usable at run time and at compile time. They contain function-specific documentation strings, help file names and contexts as well as type information like the description of the methods, properties and objects. To generate a type library an object description script is written first in the ODL format. After that, a class description header (.h) file and a type library file (.tlb) is created with a MIDL compiler.

A type library can be built-in in an EXE or DLL file or as a stand-alone file (.tlb file) [MLAUTan].

---

<sup>77</sup> c.p. 2.5.

<sup>78</sup> c.p. 4.4.

With the following command a type library with the name `output.tlb` and the header file `output.h` is generated from the description script `inscript.odl`:

```
MIDL /TLB output.tlb /H output.h inscript.odl
```

In order for Automation to locate the type library it is necessary to register the library in the application's registration file. A type library is built by adding the header file in the project. The whole project is compiled. It is also possible to join the type library with the compiled project thanks to the Resource Compiler (RC) [MLAUTao].

### 4.7.2. Registration of a Type Library<sup>79</sup>

The information that is exposed by applications and tools must be registered. Then the information can be used by programming tools and type browser.

With the `RegisterTypeLib` function the registration entries for the type library are created.

The data about the type library is stored in the following manner:

- The universally unique ID of the type library is written as follows.

```
\TypeLib\{libUUID}
```

- `Major.minor` is the version number of the type library. A changing of the major version number requires a recompilation of code that was compiled against the type library. If only the minor version number rises, it results in support of all characteristics of the prior type library. `human_readable_string` offers a short instruction of the type library.

```
\TypeLib\{libUUID}\major.minor = human_readable_string
```

- The `helpfile_path` offers the location of the Help file of the type library.

```
\TypeLib\{libUUID}\major.minor\HELPDIR = [helpfile_path]
```

---

<sup>79</sup> [MLAUTap]

- `typelib_flags` is a hexadecimal description of the type library flags.

```
\TypeLib\{libUUID}\major.minor\Flags = typelib_flags
```

- `lcid` means local identifier which is described as a hexadecimal string. With `platform` the target operating system platform is presented. `localized_typelib_filename` is the name of the localized type library.

```
\TypeLib\{libUUID}\major.minor\lcid\platform = localized_typelib_filename.
```

### 4.7.3. Error Handling

If there is an error, ActiveX objects give back a message of the error, an error number and the location of a Help file. Otherwise, it is possible to give back an `HRESULT` (return value with severity code, context information, facility code and status code [MLAUTaq]) with data of the error [MLAUTar].

## 4.8. Access of ActiveX Objects

To use ActiveX objects it is necessary to initialize OLE and to generate an instance of the object. After obtaining information about the member functions of the object the functions are called. When the application ends the active object is withdrawn and OLE is uninitialized [MLAUTas].

## 5. How to Get Script Code

For a developer there is often a problem to get the script code. In the most cases, there is no explanation of an Automation issue for Object Rexx. Here some ways are demonstrated to get the script code. This section is also interesting for non-Object Rexx developer. The Automation technology is a technology from Microsoft and in this way many examples are available for Automation. If you want to program an Object Rexx script, first try it with a Visual Basic Script, if it works, and then convert it to Object Rexx script code. Proceedings like trial and error or the macro recorder tool are included in this section as well as a table that help to convert Visual Basic Script code to Object Rexx code and many further information sources like Internet links.

### 5.1. Trial and Error

Trial and Error is an important hint. Often there is no documentation available and then the developer has to understand the system by trial and error. Attention should also be paid to “small” things like a dot. The author took a long time to program the `Verify` method of the `Scripting.Signer` object with Object Rexx but it did not work<sup>80</sup>. Sometime the author tried to invoke this method with the tool `OLEInfo.rex`<sup>81</sup> and recognized that there was small dot before the phrase “.true”. This was the solution to the problem. It is very important to be tenacious.

### 5.2. Macro Recorder Tool

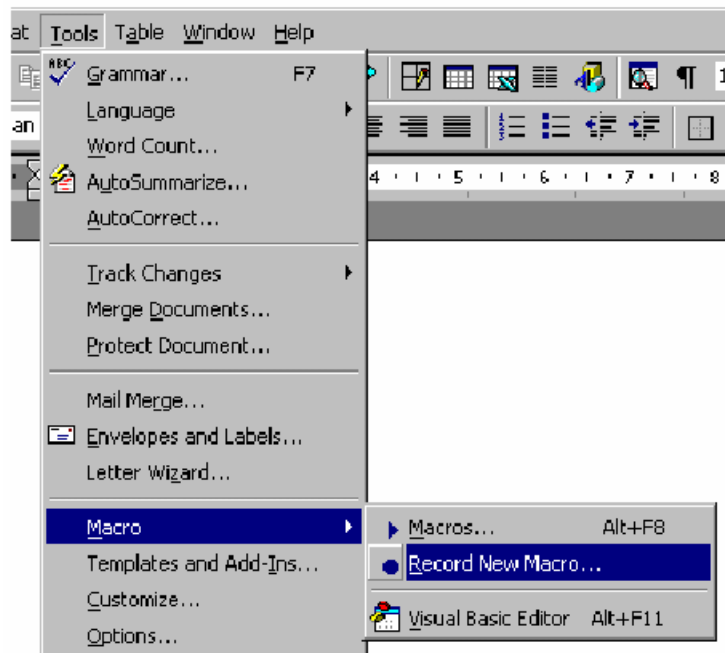
A helpful tool to get the source code for a script, is the "Macro Recorder Tool" of the Microsoft Office applications, that creates Visual Basic code of the actions that are performed manually by the user [IBM02a,p1].

---

<sup>80</sup> c.p. 13.10.2.

<sup>81</sup> c.p. 7.2.

Figure 19 illustrates the macro recorder tool of MS Word.

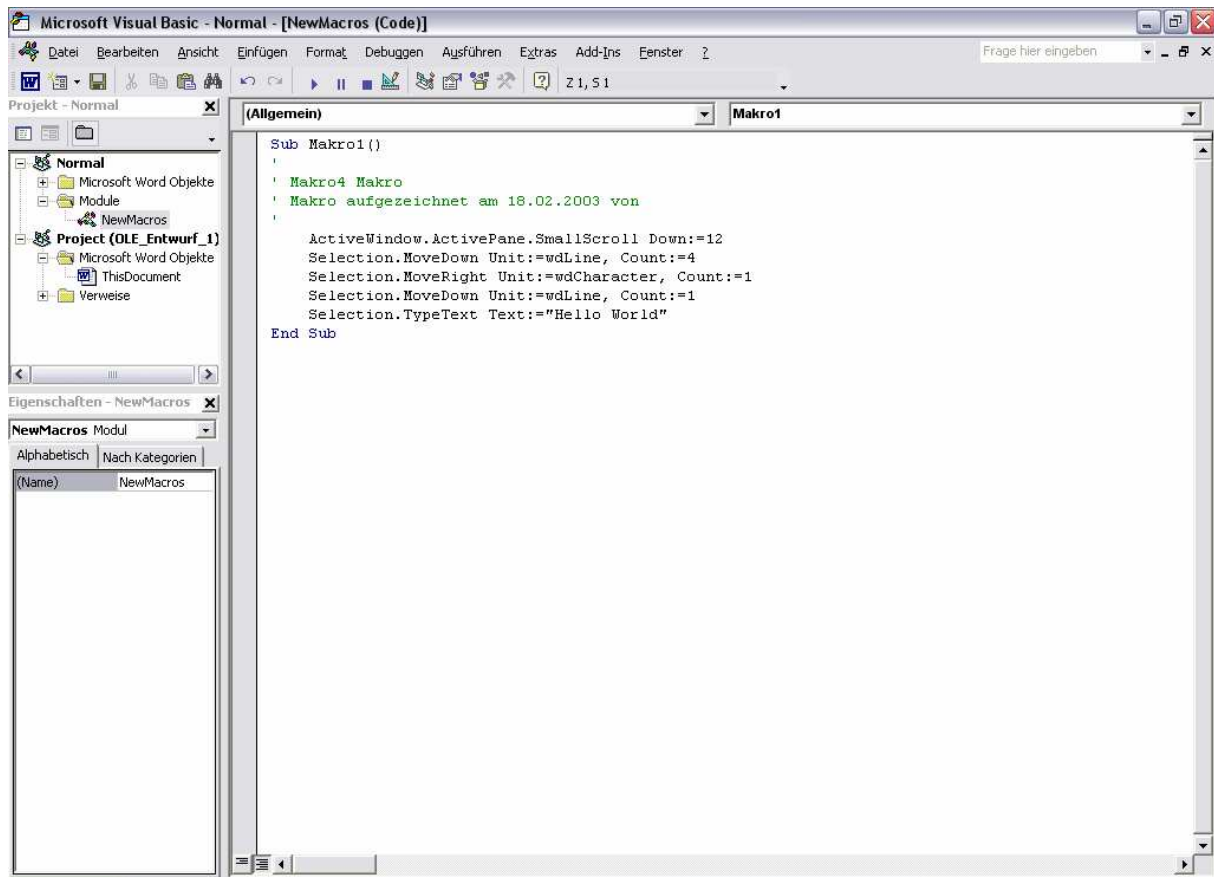


**Figure 19: Macro Recorder tool<sup>82</sup>**

Microsoft Word, Microsoft Power Point or Microsoft Excel have such a macro recorder. To use it look up in the menu `Tools /Macro...`. Then the actions are made manually. After that, the recording is stopped, the macro is selected and the Visual Basic Script code can be seen by pressing the `Edit code` button like in figure 20 [IBM02a,p1].

A macro recorder tool is also available for some non-Microsoft applications.

<sup>82</sup> Taken from [IBM02a,p1]



**Figure 20:** MS Word macro, which recorded that a text is typed.<sup>83</sup>

A Visual Basic Script macro also contains information that is not required for the Object Rexx source code.

All of the information is not contained in the macro which is needed to program such a script with Object Rexx like the `Visible` property of MS Word. More information for automating MS Office can be obtained in the Microsoft Library VBA Language Reference<sup>84</sup>.

<sup>83</sup> Part of MS Word 2002

<sup>84</sup> <http://msdn.microsoft.com/nhp/default.asp?contentid=28000550>



### 5.3. Converting Visual Basic Script Code to Object REXX

Table 3 helps to translate Visual Basic Script code to Object REXX code.

	<b>Visual Basic Script Edition</b>	<b>Object REXX</b>
<i>Method operator</i>	. (point)	~ (twiddle)
<i>Continuation character</i>	_ (underscore)	, (comma) or - (hyphen)
<i>String concatenation</i>	& (ampersand)	(two vertical lines) or (two blank characters)
<i>Definition of variables</i>	DIM var_namen	Give any name
<i>Line comments</i>	` (apostrophe) REM (if no statement before) : REM (with statement before)	-- (two hyphens)
<i>Multiple line comments</i>		/*...*/ (can span several lines and can be nested)
<i>Procedure calling</i>	CALL proc1(a1, a2, a3) or proc1 a1, a2, a3	CALL proc1 a1, a2, a3
<i>Function calling</i>	a=proc1(a1, a2, a3)	a=proc1(a1, a2, a3) or CALL proc1 a1, a2, a3 a=result
<i>Function calling</i>	a=proc1(a1, a2, a3) Calling with named arguments, e.g.: a=proc1( a3 := "The 3. Argument!" )	a=proc1( , , a3) or CALL proc1 , , a3 a=result
<i>Defining a procedure</i>	Sub proc1(a1, a2, a3) MsgBox "a1=" & a1 _ " a2=" & a2 _ " a3=" & a3 End Sub	proc1: procedure parse a1, a2, a3 say "a1="a1 "a2="a2 - "a3="a3 return or ::routine proc1 say "a1="arg(1) - "a2="arg(2) - "a3="arg(3)
<i>Defining a function</i>	Func proc1(a1, a2, a3) proc1=a1 & a2 & a3 End Func	proc1: procedure parse a1, a2, a3 return a1    a2    a3 or ::routine proc1 return arg(1)    - arg(2)    arg(3)

<i>With command</i>	With MyLabel .Height = 2000 .Width = 2000 .Caption = "MyLabel" End With	MyLabel~Height = 2000 MyLabel~Width = 2000 MyLabel~Caption="MyLabel "
---------------------	---	---

**Table 3: VBScript code to Object Rexx<sup>85</sup>**

## 5.4. Other Sources

There are many other information sources, which are here listed. These are Object Rexx, Microsoft, general and other sources. There are some links with a short description of newsgroups provided.

*Object Rexx:*

- This link contains a detailed tutorial of Object Rexx from Prof. Flatscher (German): <http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/>
- The samples in the path `ObjREXX/Samples` contain some examples scripts for OLE and WSH that could be helpful.
- In the folder `...\Windows\Version 2.1\books` on the Object Rexx CD are some e-books. To emphasize are `RexxRef.pdf` and `RexxPg.pdf`.
- In the section 7 are some valuable tools described which show interesting information like the member functions of an automated application.
- The IBM Object Rexx site offers some suitable links: <http://www-3.ibm.com/software/ad/obj-rexx/>
- This IBM Object Rexx support site contains some helpful hints and links: <http://www-3.ibm.com/software/ad/obj-rexx/support.html>
- There is a link to the developer team of Object Rexx: <http://www.ibm.com/software/ad/obj-rexx/service-orexx.html>
- Homepage of the Rexx Language Association: <http://www.RexxLA.org/>

---

<sup>85</sup> Taken from [Fla02b,p21ff]

- This site offers a lot of links to Rexx sites: <http://www2.hursley.ibm.com/rexx/>
- Site with information about NetRexx: <http://www2.hursley.ibm.com/netrexx/>

*Microsoft:*

- The MSDN Library offers a lot of documents for Microsoft technologies: <http://msdn.microsoft.com/library/>
- Here is an introduction to DHTML: <http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/dhtml.asp>

*General:*

- A good search engine is Google (<http://www.google.com>) or MetaGer (<http://www.metager.de>)
- Here is a reference book about many items of information technology (German): <http://www.glossar.de>

*Newsgroups:*

- The Dbforums (<http://dbforums.com>) is a starting page for many newsgroups. To emphasize is the forum `comp.lang.rexx` (<http://dbforums.com/f135/>) for Object Rexx. A forum for scripting is `microsoft.public.scripting` (<http://dbforums.com/f194/>) with two forums for JScript and VBScript. It is not important in which of these two forums a thread is posted. There is also a forum for Microsoft.NET (<http://dbforums.com/f235/>). But there are very less postings in the MS.NET forum.
- The Developersindex (<http://www.developersdex.com/newsgroups.asp>) offers many newsgroups. There are several newsgroups for scripting ([microsoft.public.scripting.jscript](http://microsoft.public.scripting.jscript); [microsoft.public.scripting.vbscript](http://microsoft.public.scripting.vbscript); [microsoft.public.scripting.wsh](http://microsoft.public.scripting.wsh)). For MS.NET many newsgroups are provided.

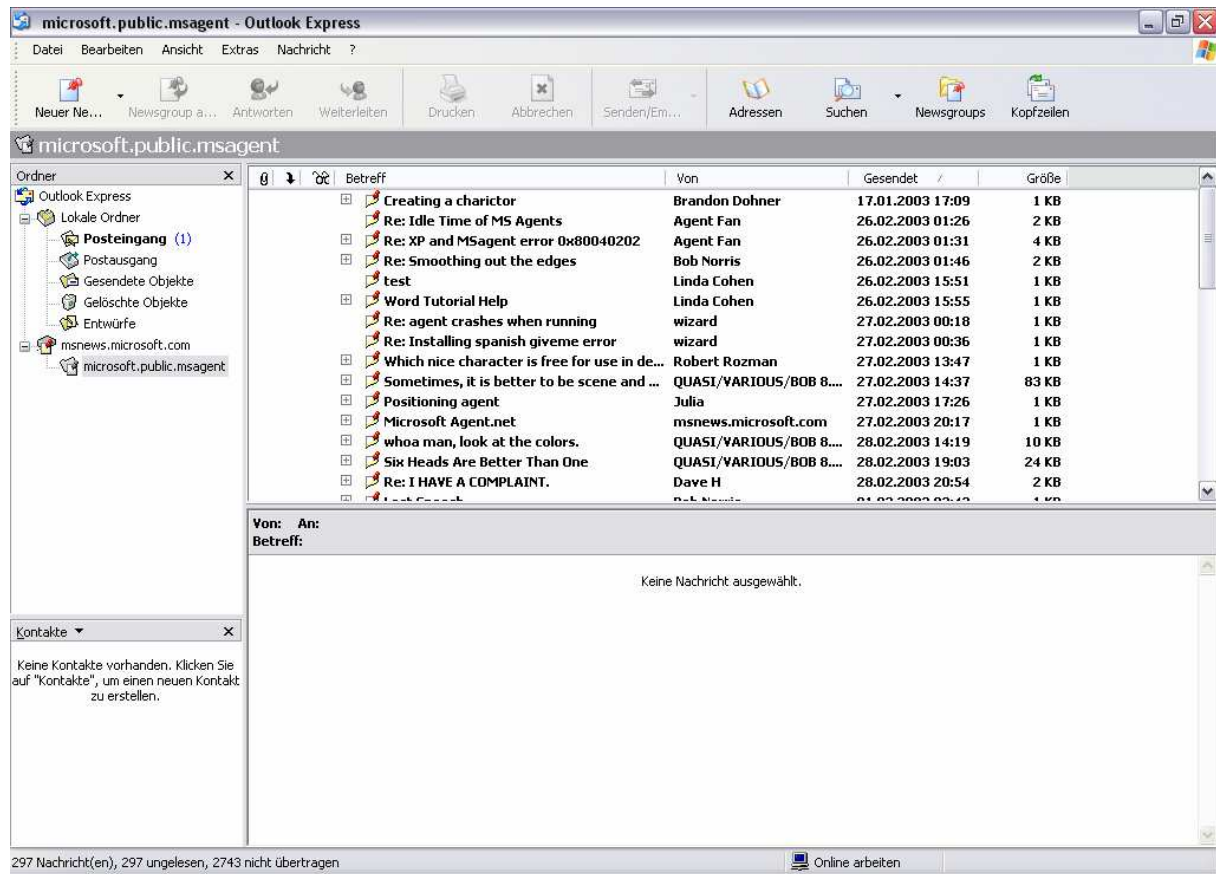
- A newsgroup for the Microsoft Speech technology is located at:  
[http://communities.microsoft.com/newsgroups/messageList.asp?ICP=MSCOM&sL CID=US&NewsGroup=microsoft.public.speech\\_tech&iPageNumber=1](http://communities.microsoft.com/newsgroups/messageList.asp?ICP=MSCOM&sL CID=US&NewsGroup=microsoft.public.speech_tech&iPageNumber=1)
- A newsgroup for the MS Agent technology can be found at this location:  
<news://msnews.microsoft.com/microsoft.public.msagent>
- Startpage for a lot of XP newsgroups:  
<http://www.microsoft.com/windowsxp/expertzone/newsgroups/default.asp>

The newsgroups are monitored by so-called MVPs (Most Valuable Professionals). The MVPs are IT specialists from Microsoft. They answer to requests of users to Microsoft technologies [Onl02].

Newsgroups can be accessed via a “normal” Internet Homepage or via the phrase <news://>. In the second case the URL of the newsgroup is inserted like the “normal” Internet Homepage to the MS Internet Explorer of Windows XP and then automatically Outlook Express 6 is started (figure 21) where the newsgroup <news://msnews.microsoft.com/microsoft.public.msagent> was typed into the command line of the MSIE<sup>86</sup>.

---

<sup>86</sup> Implemented with MS Windows XP Home



**Figure 21:** Outlook Express 6 with the newsgroup microsoft.public.msagent<sup>87</sup>.

#### Other Links:

- Here is the very rich resource for HTML “SelfHTML” located (German): <http://selfhtml.teamone.de>
- Link to the World Wide Web Consortium (W3C): <http://www.w3c.org>
- Cascading Style Sheets Homepage of W3C: <http://www.w3c.org/Style/CSS/>
- Document Object Model Homepage of W3C: <http://www.w3c.org/DOM/>
- HyperText Markup Language (HTML) Home Page of W3C: <http://www.w3c.org/MarkUp/>
- Javascript-based tutorial: <http://people.freenet.de/JavaScript/javap00.htm>
- ActiveX site for Object Rexx of Lee Peedin: <http://pragmaticlee.safedataisp.net/>

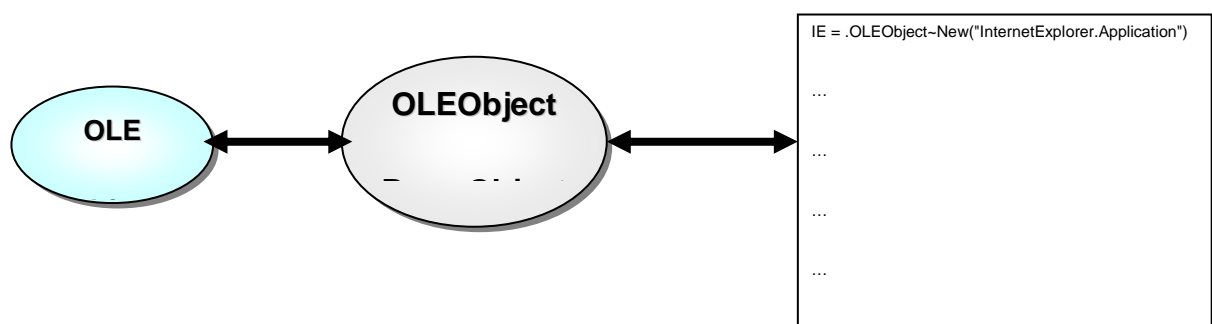
<sup>87</sup> Part of MS Windows XP

## 6.OLE and Object Rexx with OLEObject Class

This part contains the OLEObject class. This class is the core interface of Object Rexx for OLE. The methods `Init`, `Unknown`, `GetKnownEvents`, `GetKnownMethods`, `GetObject` and `GetOutParameters` of the OLEObject class and the type conversion are represented. The methods are always explained with a code example. Thereby the Microsoft Internet Explorer or parts of the Microsoft Office like MS Word or MS Excel are automated with Object Rexx. Sometimes Visual Basic Script code is used to clarify how to program the Object Rexx code.

Object Rexx supports OLE Automation. Object Rexx is an OLE Automation client. Therefore, the OLEObject class is used. The Rexx OLEObject class is not a built-in class [IBM01,p284].

In earlier Object Rexx versions the directive `::REQUIRES "OREXXOLE.CLS"` was needed. In the newest versions Object Rexx automatically loads it. In this way the directive `::REQUIRES "OREXXOLE.CLS"` with the class definition is not needed [Fla03a]. OREXXOLE.DLL contains the library with the code [En01,p9]. For the OLE object with its properties and methods the Rexx OLE object is like a proxy object. The Rexx OLE object is used like all other objects in Rexx. It transfers invocations of property get and property put operations and methods on to the real object [En01,p12].



**Figure 22:** Interaction of OLE object, OLEObject and Script<sup>88</sup>

Figure 22 illustrates the interaction of OLE object, OLEObject and the Object Rexx script.

<sup>88</sup> Modeled after [En01,p12]

## 6.1. Methods of the OLEObject Class

The table 4 contains a short reference over all OLEObject class methods and describes them.

The OLEObject class provides the following methods [IBM01,p285ff]:

Method	Description
INIT	With a CLSID or a ProgID an OLE object is instantiated.
UNKNOWN	Methods of the OLE object are called with this central mechanism.
GETCONSTANT	The value of a constant that is linked with an OLE object is retrieved.
GETKNOWNEVENTS	Returns a .stem with information on the events an OLE object can generate.
GETKNOWNMETHODS	Returns a .stem with information on the methods an OLE object offers.
GETOBJECT(Class method)	With a moniker <sup>89</sup> or nickname (a string) an OLE object can be accessed.
GETOUTPARAMETERS	This method returns an array, which contains the results of the single out parameters of the OLE object.

**Table 4:** Methods of the OLEObject class<sup>90</sup>

## 6.2. Type Conversion

OLEObject supports an automatic conversion to and from types. Thereby OLE uses, in opposition to Rexx, a strict typing of data. OLE types are named *variants*. That is because the OLE types are stored in one structure that gets flagged with the type it stands for [IBM01,p291f].

<sup>89</sup> c.p. 2.3.4.

<sup>90</sup> [IBM01,p285ff]

Table 5 illustrates the type conversion of VARIANT types and Rexx object [IBM01,p291f], [MLSR1a]:

VARIANT type		Rexx object	
VT_EMPTY	Indicates a value was not specified.	.NIL	
VT_NULL	Indicates a null reference	.NIL	
VT_VOID	Indicates a C style void	.NIL	
VT_I1	Indicates a char value	REXX string	A whole number
VT_I2	Indicates a short integer	REXX string	A whole number
VT_I4	Indicates a long integer	REXX string	A whole number
VT_I8	Indicates a 64-bit integer	REXX string	A whole number
VT_UI1	Indicates a byte	REXX string	A whole, positive number
VT_UI2	Indicates an unsigned short	REXX string	A whole, positive number
VT_UI4	Indicates an unsigned long	REXX string	A whole, positive number
VT_UI8	Indicates an 64-bit unsigned integer	REXX string	A whole, positive number
VT_R4	Indicates a float value	REXX string	A real number
VT_R8	Indicates a double value	REXX string	A real number
VT_CY	Indicates a currency value	REXX string	A fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right



VT_DATE	Indicates a DATE value	REXX string	
VT_BSTR	Indicates a BSTR string	REXX string	
VT_DISPATCH	Indicates an IDispatch pointer	REXX OLEObject	
VT_BOOL	Indicates a Boolean value	.TRUE or .FALSE	
VT_VARIANT	Indicates a VARIANT far pointer	Any REXX object that can be represented as a VARIANT	
VT_PTR	Indicates a pointer type	See VT_VARIANT	
VT_SAFEARRAY	Indicates a SAFEARRAY. Not valid in a VARIANT	REXX Array	

**Table 5: Type conversion**<sup>9192</sup>

### 6.3. Init<sup>93</sup>

INIT is the most important method of the `OLEObject` class. INIT instantiates an OLE object. The code examples demonstrate the INIT method when it is used with a ProgID, with a CLSID or with events.

Therefore, the CLSID or the ProgID is used. For the usage of events it is optionally possible to enable events (`WITHEVENTS`) or to disable events (`NOEVENTS`) which is the default setting. The CLSID respectively ProgID can be found out by looking up

<sup>91</sup> Taken from [IBM01,p291f]

<sup>92</sup> Taken from [MLSRla]

<sup>93</sup> Section uses [IBM01,p285]

the registry with the Registry Editor<sup>94</sup> or the documentation of the application or with tools like RGF\_OLEINFO.HTA<sup>95</sup>, Microsoft OLEViewer<sup>96</sup> or OLEINFO.REX<sup>97</sup>.

Syntax:

```
INIT(CLSID/ProgID, [NOEVENTS/WITHEVENTS])
```

### 6.3.1. Init with ProgID

Code 2 demonstrates the instantiation of an OLE object with the example of MS Word with the ProgID. The syntax is "Word.Application" because Rexx doesn't run "inside" Word and so first the Application object has to be generated [IBM02a,p2].

```
-----  
-- Init_Instantiation of Word with ProgID.REX --  
-----  
-- Instantiation of the MS Word with the ProgID  
Word = .OLEObject~New( "Word.Application" )  
-- The Visible property is set on true98. Word can be seen on the display  
Word~Visible = .TRUE
```

**Code 2:**        **Init\_Instantiation of Word with ProgID.REX**

### 6.3.2. Init with CLSID

The code 3 demonstrates the instantiation of an OLE object with the CLSID. This script runs with MS Word 2002.

---

<sup>94</sup> c.p. 2.6.1.

<sup>95</sup> c.p. 7.3.

<sup>96</sup> c.p. 2.6.2.

<sup>97</sup> c.p. 7.2.

<sup>98</sup> [MS03a]

```

-----
-- Init_Instantiation of Word with CLSID.REX --
-----
-- Instantiation of the MS Word with the ProgID
Word = .OLEObject~New("{000209FF-0000-0000-C000-000000000046}")
-- The Visible property is set on true99. Word can be seen on the display
Word~Visible = .TRUE

```

**Code 3:** Init\_Instantiation of Word with CLSID.REX

### 6.3.3. Init with WITHEVENTS<sup>100</sup>

Code 4 shows the usage of the parameter “WITHEVENTS”. An instance of the Microsoft Internet Explorer is created with the class `EventsWithIE` that is derived from `OLEObject`. In this class the method `OnVisible` is contained which is called if the event `OnVisible` is fired.

It is important that the directive `::CLASS WithEventsIE` gets the additional remark `SUBCLASS OLEObject`. The class `EventsWithIE` is a class below the class `OLEObject` and inherits from the superclass `OLEObject` all methods and variables [IBM01,p8].

The directive `::CLASS` generates a Rexx class. This Rexx class is now available through the Rexx environmental symbol “.EventsWithIE” (note the dot). In this way all methods like the method `OnVisible` which are defined by the `::METHOD` directive and that come subsequent to the `::CLASS` directive can be accessed [IBM01,p87].

The phrase `SUBCLASS` causes that the class `EventsWithIE` to become a subclass of the class `OLEObject` [IBM01,p88]

```

-----
-- Init_WITHEVENTS.rex --
-----
-- An instance of the InternetExplorer is generated. Here with the
-- parameter "WITHEVENTS"
IE = .EventsWithIE~new("InternetExplorer.Application","WITHEVENTS")
-- The InternetExplorer is shown101. This invokes the firing of the

```

<sup>99</sup> [MS03a]

<sup>100</sup> [IBM01b]

```

-- OnVisible event
IE~visible = .true
-- Class which derives from OLEObject
::CLASS EventsWithIE SUBCLASS OLEObject
-- Method which is called if the event OnVisible is fired in case
-- if the window is shown or hidden.
::METHOD OnVisible -- intercepts the Visible event
say "The Microsoft InternetExplorer is visible"

```

**Code 4:**      **Init\_WITHEVENTS.rex**<sup>102</sup>

## 6.4. UnKnown<sup>103</sup>

The UNKNOWN method is, as well as the INIT method<sup>104</sup>, a very important part of the OLEObject class of Object Rexx. It is a fundamental instrument, through which the methods of an OLE object are accessed.<sup>105</sup> The code examples explain the UNKNOWN method with arguments, without arguments and with identical method names.

The UNKNOWN message passes on all unknown messages to the OLE program [Fla02a,p13]. Methods of the OLE objects are called with this method. If the names of methods of the OLE object and OLEObject are equal, then the method must be called over the UNKNOWN method<sup>106</sup>. The problem occurs in the most cases with Copy or Class messages [Fla02a,p13]. The UNKNOWN method has two arguments. The first argument is the name of the method that is to call. The second argument contains the arguments of the method that is to call.

Syntax:

```
UNKNOWN( messagename , messageargs )
```

### 6.4.1. Unknown without Arguments

Code 5 demonstrates the UNKNOWN method without message arguments.

<sup>101</sup> [MS03b]

<sup>102</sup> Modeled after [IBM01b]

<sup>103</sup> This section uses [IBM01,p290]

<sup>104</sup> c. p. 6.3.

<sup>105</sup> c.p. 2.1.

<sup>106</sup> c.p. 12.1.3.

At first, the selection of a cell in MS Excel is implemented with the `Select` method in a direct way. In the next step a cell is selected with the `UNKNOWN` method. Thereby the `Select` method is given over to the `UNKNOWN` method.

```
-----
-- UnKnown_without_Arguments.rex --
-----

-- Instantiation of Excel
Excel = .OLEObject~new("Excel.Application")
-- A workbook with a worksheet is added. "1" is the index of
-- the worksheet107
Worksheet = Excel~Workbooks~Add~Worksheets[1]
-- The Visible property is set on true so that excel can be seen on the
-- display108
Excel~Visible = .true
-- The cell A7 is selected direct with the Select method109
WorkSheet~Range("A7")~Select
-- Handover of the value "This cell was selected" to the cell110
WorkSheet~Range("A7")~Value = "This cell was selected"
-- Message box with information
CALL RxMessageBox "The next cell is selected over the UNKNOWN method", -
"Information", "OK", "ASTERISK"
-- The cell C7 is selected indirect over the UNKNOWN method with
-- the Select method. Because there are no arguments, .NIL is set
WorkSheet~Range("C7")~UNKNOWN("Select",.nil)
```

**Code 5:** UnKnown\_without\_Arguments.rex

## 6.4.2. Unknown with Arguments

Code 6 shows the `UNKNOWN` method with message arguments.

For the demonstration, the `CheckSpelling` method of MS Excel is used. First, the method is used in the normal, direct use whereby arguments are hand over too. Then the method is implemented indirect over the `UNKNOWN` method, whereby an array is used to hand over the arguments. This script was programmed for the German version of MS Excel 2000<sup>111</sup>. For other versions the phrase “Benutzer.Dic” must

<sup>107</sup> [MS03c]

<sup>108</sup> [MS03d]

<sup>109</sup> [MS03h]

<sup>110</sup> [MS03f]

<sup>111</sup> Part of MS Office

be replaced. Probably instead of "Benutzer.Dic" "BenutzerDic" is used. Make a macro to get it.

```
-----
-- UnKnown_with_Arguments.rex --
-----

Excel = .OLEObject~New("Excel.Application") -- Instantiation of Excel
WordToCheck1 = "Test" -- Word to check
    -- Calls the method CheckSpelling and hands over the argument
    -- with the text to check and the name of the optional parameter
    -- CustomDictionary. The argument IgnoreUppercase is left free.
    -- This version of the CheckSpelling method is used with the
    -- Application object112
ResultOfCheckSpellingWithoutUnknown =Excel~CheckSpelling(WordToCheck1,Benutzer.Dic)
    -- Checks the result and tells if the spelling is correct
IF ResultOfCheckSpellingWithoutUnknown = 1 then -
    SAY "The spelling of" WordToCheck1" is correct."
ELSE SAY "The spelling of" WordToCheck1" is NOT correct."
WordToCheck2 = "Teest" --Word to check
    -- Array which is hand over to the variable ArrayForUnKnown
    -- with the word which is to check, the optional CustomDictionary
    -- "BenutzerDic" and the optional parameter IgnoreUppercase is left
    -- free.
ArrayForUnKnown = .array~of(WordToCheck2,Benutzer.Dic,)
    -- The method CheckSpelling is used indirect over the UNKNOWN method.
    -- Thereby the arguments of array ArrayForUnKnown are used. The
    -- result is hand over to the variable ResultOfCheckSpellingWithUnknown
ResultOfCheckSpellingWithUnknown = Excel~UNKNOWN("CheckSpelling", ArrayForUnKnown)
    -- Checks the result and tells if the spelling is correct
IF ResultOfCheckSpellingWithUnknown = 1 then SAY "The spelling of" -
    WordToCheck2 " is correct."
ELSE SAY "The spelling of" WordToCheck2 " is NOT correct."
```

**Code 6:** UnKnown\_with\_Arguments.rex

### 6.4.3. Unknown with identical Method Names

Code 7 demonstrates an UNKNOWN method that is used in a situation when the name of an OLE object method is equal to one of OLEObject.

This script creates a chart in an MS Excel sheet and copies this chart. The Copy method would not be transmitted to MS Excel 2000 but to the Object Rexx class object. The UNKNOWN method of OLEObject solves this problem [He02,p17ff].

<sup>112</sup> [MS03i]

First a macro (OtherScript 1) with the macro recorder tool of MS Excel is generated; this helps to get the Object Rexx code. It is obvious that not all of the macro code is necessary for the Object Rexx code. This macro contains a code that creates a chart with data from the cell A1. In the Object Rexx script code 7 there is, additionally to the macro, the chart selected and copied to the clipboard.

```
Sub Makro1()
'
' Makro2 Makro
' Makro am 03.03.2003 von FH aufgezeichnet
'
'
'
ActiveCell.FormulaR1C1 = "10"
Range("A1").Select
Charts.Add
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData Source:=Sheets("Tabelle1").Range("A1"), PlotBy:= _
    xlRows
ActiveChart.Location Where:=xlLocationAsObject, Name:="Tabelle1"
With ActiveChart
    .HasTitle = False
    .Axes(xlCategory, xlPrimary).HasTitle = False
    .Axes(xlValue, xlPrimary).HasTitle = False
End With
End Sub
```

**OtherScript 1: Excel macro for UnKnown\_Identical\_Methodnames.rex**

Code 7 uses the information of the macro OtherScript 1.

```
-----
-- UnKnown_Identical_Methodnames.rex --
-----
Excel = .OLEObject~new("Excel.Application") -- Instantiation of Excel
    -- A workbook with a worksheet is added. "1" is the index of the
    -- worksheet113
Worksheet = Excel~Workbooks~Add~Worksheets[1]
    -- The Visible property is set on true so that Excel can be seen
    -- on the display114
Excel~Visible = .TRUE
Worksheet~Range(A1)~Value = 10 -- The value of the cell A1 is set on "10"
Excel~Charts~Add -- The Add method places a new chart sheet
```

<sup>113</sup> [MS03c]

<sup>114</sup> [MS03d]

```
-- The Location method embeds the chart with the constant
-- "XlLocationAsObject" in the sheet "Tabelle1"115
Excel~ActiveChart~Location(Excel~GetConstant("XlLocationAsObject"),"Tabelle1")
-- Offers the source of data for the chart
Excel~ActiveChart~SetSourceData(Worksheet~Range("A1"))
-- The chart area in the active chart is selected
Excel~ActiveChart~ChartArea~select
-- Here the chart is copied to the clipboard. The Copy method
-- cannot be used in the normal way because this Copy method
-- wouldn't be sent to Excel but to the Object Rexx class. The
-- UNKNOWN method solves this problem.
Excel~ActiveChart~ChartArea~UNKNOWN("Copy",.nil)
```

**Code 7:**      **UnKnown\_Identical\_Methodnames.rex**<sup>116</sup>

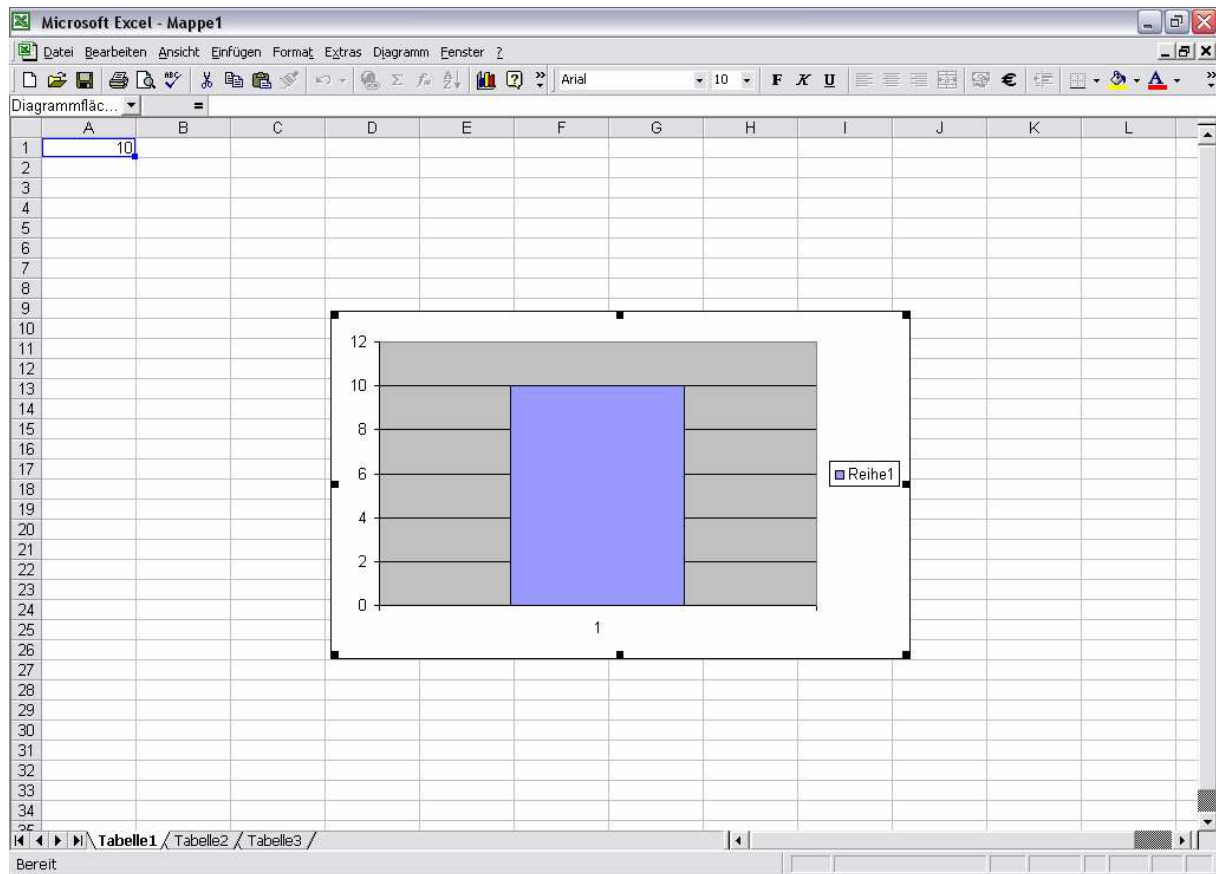
Figure 23 illustrates the result of code 7.

---

<sup>115</sup> c.p. 6.6.

<sup>116</sup> [He02,p17ff]





**Figure 23:** Snapshot of MS Excel<sup>117</sup>

## 6.5. GetObject<sup>118</sup>

The `GETOBJECT` method enables to get an active OLE object. This method is similar to the `Init` method<sup>119</sup>.

Therefore a moniker is used which informs OLE of the type of object that is necessary. The moniker<sup>120</sup> determines which object has to be generated or to be addressed in the case if the object is running. A moniker of the type file moniker is for example the name and location of a MS Word document like "C:\Test\TestWordDocument.DOC" [MLCOMaj]. Other monikers can be seen in section 10.

<sup>117</sup> Part of MS Office

<sup>118</sup> [IBM01,p289f]

<sup>119</sup> c.p. 6.3.

<sup>120</sup> c.p. 2.3.4.

The parameter `Class` is not obligatory. It is employed to install a subclass of `OLEObject` or to get an OLE object, which supports events (“WITHEVENTS”).

Syntax:

```
GETOBJECT(Moniker, Class)
```

Code 8 shows the implementation of the `GETOBJECT` method.

```
-----  
-- GetObject.rex --  
-----  
-- Opening of the Word file "TestWordDocument" and get an OLE object  
WordOLEObject = .OLEObject~GetObject("C:\Test\TestWordDocument.DOC")
```

**Code 8:      GetObject.rex**

## 6.6. GetConstant<sup>121</sup>

The `GetConstant` method returns the value of a constant. There are applications that have many constants. MS Word 2002 for example has 2682 constants<sup>122</sup>. The `GetConstant` method is represented in the example scripts with and without the name of a constant.

A constant is associated with an OLE object. The `.NIL` object is given back if the constant does not exist.

Syntax:

```
GETCONSTANT(Constantname)
```

It is possible for you to define constants with a programming language like Visual Basic for Applications. The Microsoft Office contains built-in constants. Their values are predefined. The application, to which an MS Office constant refers, can be determined by the constant prefix. For example the prefix “wd” for Word or the prefix “xl” for Excel [MS01].

---

<sup>121</sup> This section uses [IBM01,p286]

<sup>122</sup> Looked up with RGF\_OLEInfo.hta c.p. 7.3.

### 6.6.1. GetConstant with the Name of the Constant

This method is demonstrated with a script that writes several different texts in MS Word.

First, a macro is recorded in MS Word<sup>123</sup>. This macro is shown in OtherScript 2. It is obvious that there is much more data recorded than is needed by the Object Rexx script.

```
' Makro aufgezeichnet am 03.03.2003 von
'

Documents.Add DocumentType:=wdNewBlankDocument
Selection.TypeText Text:="Hello World"
Selection.Font.Color = wdColorRed
Selection.TypeText Text:="Hello World"
If Selection.Font.Underline = wdUnderlineNone Then
    Selection.Font.Underline = wdUnderlineSingle
Else
    Selection.Font.Underline = wdUnderlineNone
End If
Selection.TypeText Text:="Hello World"
With Selection.ParagraphFormat
    With .Shading
        .Texture = wdTextureNone
        .ForegroundColor = wdColorAutomatic
        .BackgroundColor = wdColorYellow
    End With
    .Borders(wdBorderLeft).LineStyle = wdLineStyleNone
    .Borders(wdBorderRight).LineStyle = wdLineStyleNone
    .Borders(wdBorderTop).LineStyle = wdLineStyleNone
    .Borders(wdBorderBottom).LineStyle = wdLineStyleNone
    With .Borders
        .DistanceFromTop = 1
        .DistanceFromLeft = 4
        .DistanceFromBottom = 1
        .DistanceFromRight = 4
        .Shadow = False
    End With
End With
```

---

<sup>123</sup> c.p. 5.2.

```

With Options
    .DefaultBorderLineStyle = wdLineStyleSingle
    .DefaultBorderLineWidth = wdLineWidth050pt
    .DefaultBorderColor = wdColorAutomatic
End With
End Sub

```

**OtherScript 2: MS Word macro for code 9**

In code 9 is the GetConstant method with the name of the constant demonstrated.

```

-----
-- GetConstant_Get Constant with name of Constant.rex --
-----

-- Instantiation of the MS Word with its ProgID
Word = .OLEObject~New("Word.Application")
Word~Visible = .TRUE -- Word is visible124
Word~Documents~Add -- A new word document is added
Word~Selection~TypeText("Hello World ") -- Writes the text "Hello World "
-- Changes the Color of the Font to Red with the Constant "wdColorRed"
Word~Selection~Font~Color = Word~GetConstant('wdColorRed')
-- Writes the text "Hello World " which has now the color red.
Word~Selection~TypeText("Hello World ")
-- Underlines the text. The value of the constant "wdUnderlineSingle"
-- is "1". The value is looked up with rgf_oleinfo.hta125
Word~Selection~Font~Underline = 1
-- Writes the text "Hello World " which is red and underlined
Word~Selection~TypeText("Hello World ")
-- The variable ConstantValueOfWdColorYellow the value of the
-- constant "wdColorYellow" is handed over.
ConstantValueOfWdColorYellow = Word~GetConstant('wdColorYellow')
-- The backgroundcolor of the paragraph gets the color yellow
-- which is indicated by variable ConstantValueOfWdColorYellow
Word~Selection~ParagraphFormat~Shading~BackgroundPatternColor -
= ConstantValueOfWdColorYellow

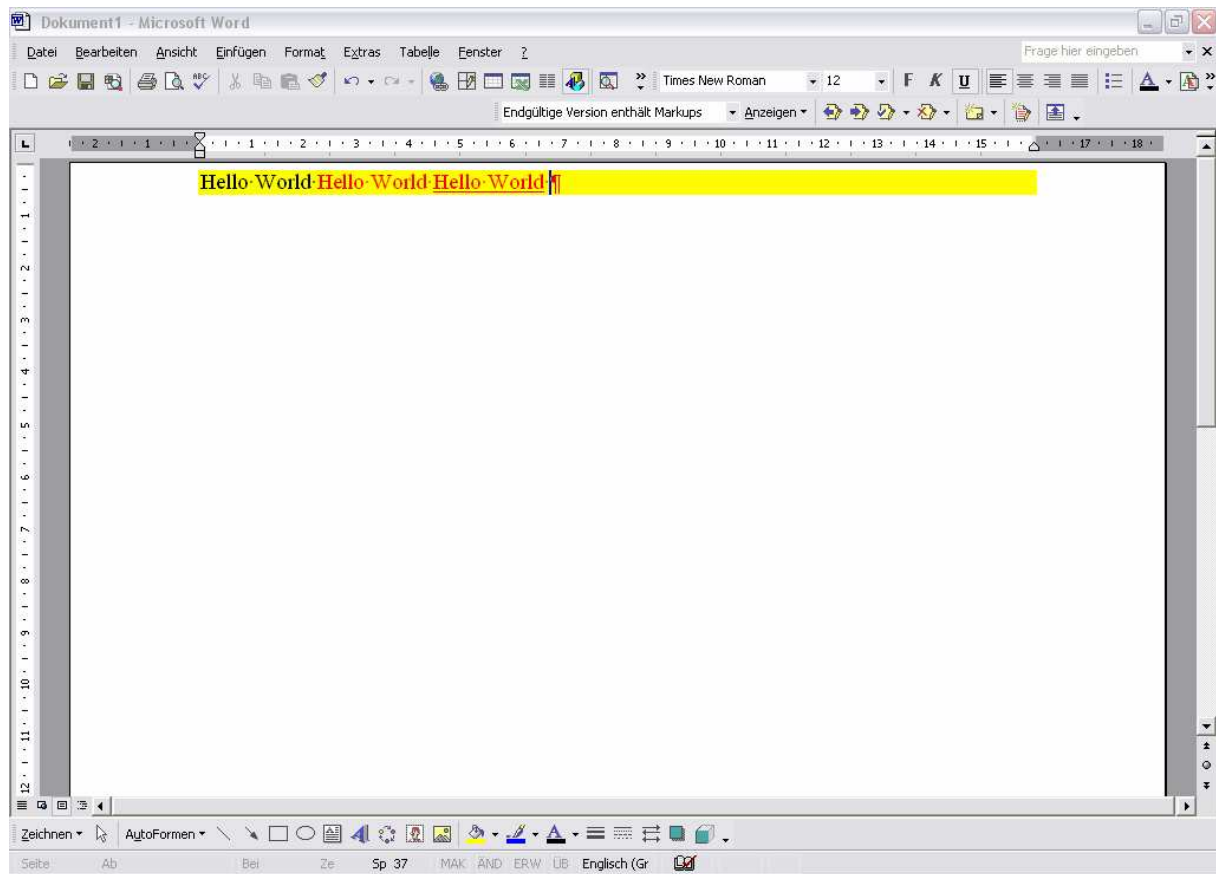
```

**Code 9: GetConstant\_GetConstant with name of Constant.REX**

Figure 24 illustrates the result of code 9.

<sup>124</sup> [MS03a]

<sup>125</sup> c.p. 7.3.



**Figure 24:** Snapshot of MS Word<sup>126</sup>

## 6.6.2. GetConstant without the Name of the Constant

Code 10 shows all constants of MS Excel 2000 with their values.

All names have a “!” symbol at their beginning because the name of the constant is left out and a stem collection is built.

```
-----
-- GetConstant_Get Constant without name of Constant.rex --
-----

-- Instantiation of the MS Excel with its ProgID
Excel = .OLEObject~new("Excel.Application")
-- A workbook with a worksheet is added. "1" is the index of the
-- worksheet127
Worksheet = Excel~Workbooks~Add~Worksheets[1]
-- The Visible property is set on true so that excel can be seen
-- on the display
Excel~Visible = .true
```

<sup>126</sup> Part of MS Office

<sup>127</sup> [MS03c]

```

constant. = Excel~GetConstant -- Creation of a stem collection
    -- The start value of counter = 1. The counter is needed that
    -- each entry is in a new cell.
counter = 1
    -- DO function over the collection of all constants
DO constantname OVER constant.
DO counter UNTIL counter > 0 -- DO function which never ends
    -- Description of the cell. "A" stands for the column and
    -- "counter" for the row. "||" prevents a blank character
a = "A" || counter
    -- Description of the cell. "B" stands for the column and
    -- "counter" for the row. "||" prevents a blank character
b = "B" || counter
    -- Cells with the constantnames
Worksheet~Range(a)~value = constantname
Worksheet~Range(b)~value = constant.constantname
counter = counter + 1 -- Increment of the counter
END -- End of the second DO function
END -- End of the first DO function
    -- This command adapts automatically the breadth of the A column.
Worksheet~Columns("A:A")~EntireColumn~AutoFit

```

**Code 10:**      **GetConstant\_GetConstant without name of Constant.REX<sup>128</sup>**

## 6.7. GetKnownEvents<sup>129</sup>

This method helps to get an overview over all events that are supplied from an application.

The method GETKNOWNEVENTS returns a stem with data about the events of an OLE object. This data with arguments of the belonging methods, types and names is contained in the type library of the object.

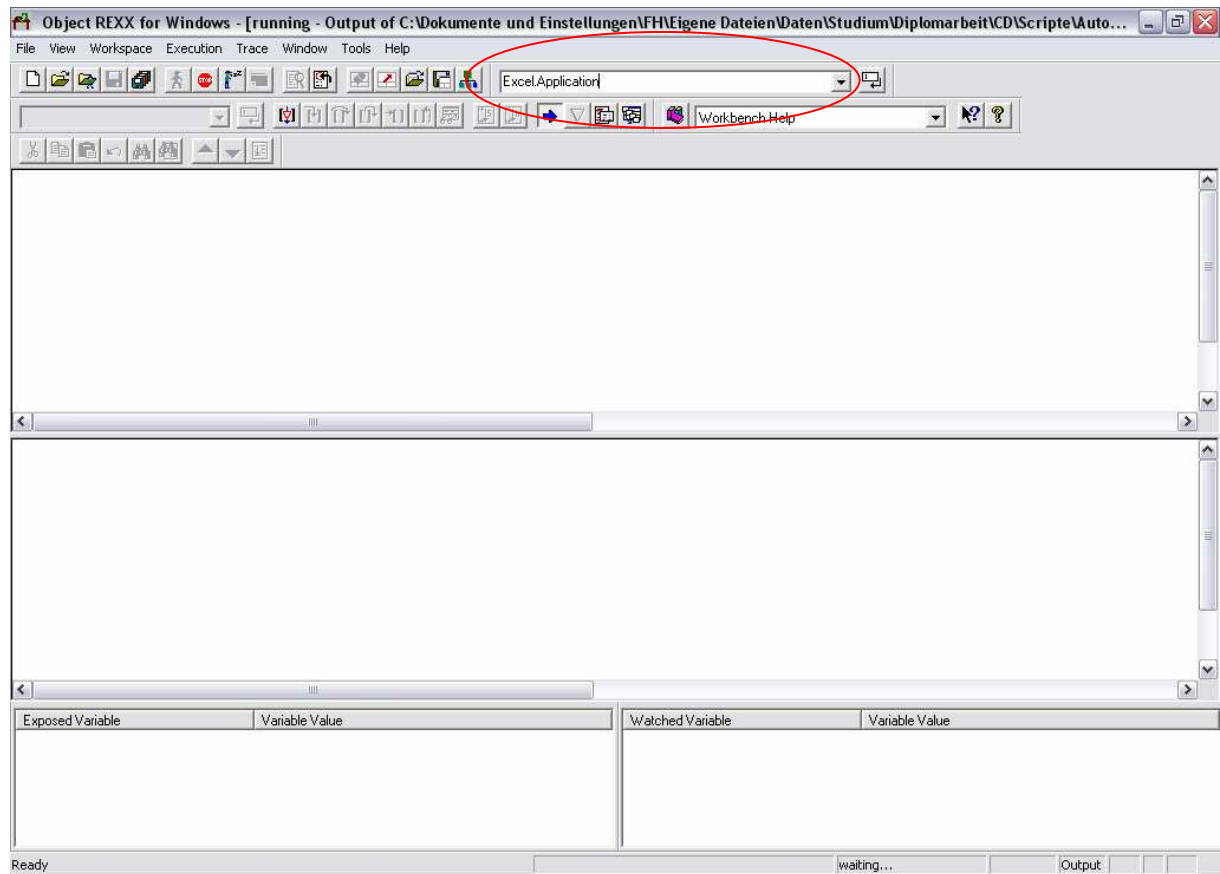
Syntax:

GETKNOWNEVENTS

Code 11 lists all events of an OLE object in an Excel sheet. Therefore, the ProgID of the OLE object must be inserted to the command line like demonstrated in figure 25.

<sup>128</sup> Modeled after [IBM01,p286]

<sup>129</sup> [IBM01,p286f]



**Figure 25:** Object REXX Workbench with the command line<sup>130</sup>

Code 11 writes data about all events. The name of an event, the description of the event, the number of parameters and name, type and flags of the parameter are written to an MS Excel sheet. If there is no information about events the .NIL object is given back and the function Termination is called.

```

-----
-- GetKnownEvents_AllEventsOfAnApplication.rex --
-----

    -- Message box with the invitation to insert the ProgID
CALL RxMessageBox "Insert the ProgID!", "Information", "OK", "ASTERISK"
PARSE PULL PROGID -- Hand over of the ProgID in the command window
    -- Instantiation of the Application with its ProgID with the
    -- parameter "NOEVENTS" and hand over to the variable "App"
App = .OLEObject~New(PROGID, "NOEVENTS")
event. = App~GetKnownEvents -- Stem creation
    -- Checks if information is available. If not then the .NIL object
    -- is given back and Termination is called. If the information is
    -- available the program goes on.

```

<sup>130</sup> IBM Object REXX Workbench

```

IF event. == .NIL then CALL Termination ELSE
Excel = .OLEObject~new("Excel.Application") -- Instantiation of Excel
    -- A workbook with a worksheet is added."1" is the index of the
    -- worksheet131
Worksheet = Excel~Workbooks~Add~Worksheets[1]
    -- The Visible property is set on true so that Excel can be seen
    -- on the display132
Excel~Visible = .true
    -- Handover of value "Eventnumber"133
Worksheet~Range("A1")~value = "Eventnumber"
    -- Handover of value "Name of the Event"
Worksheet~Range("B1")~value = "Name of the Event"
    -- Handover of value "Eventdescription"
Worksheet~Range("C1")~value = "Eventdescription"
    -- Handover of value "Number of Parameters of the Event"
Worksheet~Range("D1")~value = "Number of Parameters of the Event"
    -- Handover of value "Name of the Parameter"
Worksheet~Range("E1")~value = "Name of the Parameter"
    -- Handover of value "Type of the Parameter"
Worksheet~Range("F1")~value = "Type of the Parameter"
    -- Handover of value "Flags of the Parameter"
Worksheet~Range("G1")~value = "Flags of the Parameter"
    -- The cells A1 to G1 get the backgroundcolor bright yellow
WorkSheet~Range("A1:G1")~Interior~ColorIndex = 36
    -- The start value of counter = 2 so that the first line is free
    -- for the description. The CounterCell counter is needed that
    -- each entry is in a new cell.
CounterCell = 2
DO i = 1 TO event.0 -- DO function over all events
    -- The number of parameters is hand over to the variable
    -- NumberOfParameters
NumberOfParameters = event.i.!PARAMS.0
DO CounterCell UNTIL CounterCell > 0 -- DO function which never ends
    -- Description of the cells. "A, B, C, D" stands for the column
    -- and "counter" for the row. "||" prevents a blank character
a = "A" || counterCell
b = "B" || counterCell
c = "C" || counterCell
d = "D" || counterCell
Worksheet~Range(a)~value = event.i -- Handover of the eventnumber
    -- Handover of the name of the event
Worksheet~Range(b)~value = event.i.!NAME
    -- Handover of the description of the event
Worksheet~Range(c)~value = event.i.!DOC

```

<sup>131</sup> [MS03c]

<sup>132</sup> [MS03d]

<sup>133</sup> [MS03f]



```

-- Handover of the number of parameter
Worksheet~Range(d)~value = event.i.!PARAMS.0
-- Start value of the counter variable CounterNumberOfParameters
-- is set on "1"
CounterNumberOfParameters = 1
-- DO function that is executed until the Variable
-- CounterNumberOfParameters is less than NumberOfParameters + 1
DO WHILE CounterNumberOfParameters < NumberOfParameters + 1
-- Description of the cell. "E" stands for the column and "counter"
-- for the row. "||" prevents a blank character
e = "E"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the name of the parameter
Worksheet~Range(e)~value = event.i.!PARAMS.CounterNumberOfParameters.!NAME
-- Description of the cell. "F" stands for the column and "counter"
-- for the row. "||" prevents a blank character
f = "F"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the type of the parameter
Worksheet~Range(f)~value = event.i.!PARAMS.CounterNumberOfParameters.!TYPE
-- Description of the cell. "G" stands for the column and "counter" for
-- the row. "||" prevents a blank character
g = "G"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the flag of the parameter
Worksheet~Range(g)~value = event.i.!PARAMS.CounterNumberOfParameters.!FLAGS
-- Increment of counter CounterNumberOfParameters
CounterNumberOfParameters = CounterNumberOfParameters + 1
END -- End command for the third DO function
-- Increment of the counter CounterCell
CounterCell = CounterCell + NumberOfParameters
END -- End command for the second DO function
END -- End command for the first DO function
-- These commands adapts automatically the breadth of the
-- A, B, C, D, E, F and G columns.134135
Worksheet~Columns("A:A")~EntireColumn~AutoFit
Worksheet~Columns("B:B")~EntireColumn~AutoFit
Worksheet~Columns("C:C")~EntireColumn~AutoFit
Worksheet~Columns("D:D")~EntireColumn~AutoFit
Worksheet~Columns("E:E")~EntireColumn~AutoFit

```

<sup>134</sup> [MS03g]

<sup>135</sup> [MS03e]

```

Worksheet~Columns("F:F")~EntireColumn~AutoFit
Worksheet~Columns("G:G")~EntireColumn~AutoFit
EXIT -- Termination of the program
Termination: -- Target if the .NIL object is given back
SAY "There are NO events!"
EXIT -- Termination of the program

```

**Code 11:**      **GetKnownEvents\_AllEventsOfAnApplication.rex**

Figure 26 shows a MS Excel sheet with all events of MS Excel.

Eventnumber	Name of the Event	Eventdescription	Number of Parameters of the Event	Name of the Parameter	Type of the Parameter	Flags of the Parameter
1	SheetFollowHyperlink	(null)		2 Sh	VT_DISPATCH	[in]
2	WindowDeactivate	(null)		Target	VT_PTR	[in]
3	WindowActivate	(null)		2 Wb	VT_PTR	[in]
4	WindowResize	(null)		Wn	VT_PTR	[in]
5	WorkbookAddinUninstall	(null)		2 Wb	VT_PTR	[in]
6	WorkbookAddinInstall	(null)		Wn	VT_PTR	[in]
7	WorkbookNewSheet	(null)		2 Wb	VT_PTR	[in]
8	WorkbookBeforePrint	(null)		Sh	VT_DISPATCH	[in]
9	WorkbookBeforeSave	(null)		2 Wb	VT_PTR	[in]
10	WorkbookBeforeClose	(null)		Cancel	VT_PTR	[in]
11	WorkbookDeactivate	(null)		3 Wb	VT_PTR	[in]
12	WorkbookActivate	(null)		SaveAsUI	VT_BOOL	[in]
13	WorkbookOpen	(null)		Cancel	VT_PTR	[in]
14	SheetChange	(null)		2 Wb	VT_PTR	[in]
15	SheetCalculate	(null)		Cancel	VT_PTR	[in]
16	SheetDeactivate	(null)		1 Wb	VT_PTR	[in]
17	SheetActivate	(null)		1 Wb	VT_PTR	[in]
18	SheetBeforeRightClick	(null)		1 Wb	VT_PTR	[in]
19	SheetBeforeDoubleClick	(null)		2 Sh	VT_DISPATCH	[in]
20	SheetBeforeDoubleClick	(null)		Target	VT_PTR	[in]
21	SheetBeforeDoubleClick	(null)		1 Sh	VT_DISPATCH	[in]
22	SheetBeforeDoubleClick	(null)		1 Sh	VT_DISPATCH	[in]
23	SheetBeforeDoubleClick	(null)		1 Sh	VT_DISPATCH	[in]
24	SheetBeforeDoubleClick	(null)		3 Sh	VT_DISPATCH	[in]
25	SheetBeforeDoubleClick	(null)		Target	VT_PTR	[in]
26	SheetBeforeDoubleClick	(null)		Cancel	VT_PTR	[in]
27	SheetBeforeDoubleClick	(null)		3 Sh	VT_DISPATCH	[in]
28	SheetBeforeDoubleClick	(null)		Target	VT_PTR	[in]
29	SheetBeforeDoubleClick	(null)		Cancel	VT_PTR	[in]
30	SheetBeforeDoubleClick	(null)		2 Sh	VT_DISPATCH	[in]

**Figure 26:**      **Snapshot of MS Excel<sup>136</sup>**

## 6.8. GetKnownMethods<sup>137</sup>

This method helps to get an overview over all methods that are supplied from an application.

The method GETKNOWNMETHODS gives back a stem with data about the methods of an OLE object. This data with arguments, types and names is contained in the type

<sup>136</sup> Part of MS Office

<sup>137</sup> [IBM01,p287ff]

library of the object. Not all the data can be used directly. The .NIL object is given back if the data is not available. Some methods, which are only used internally, are hidden and not shown to the user.

Syntax:

GETKNOWNMETHODS

The following code 12 allows looking at all methods of an OLE object. Therefore the ProgID of the OLE object must be inserted to the command line<sup>138</sup>. The script writes data about all methods with their name, description of the method, name and description of the type library, return type of the method, MemberID, kind of invocation (normal method call, property or property put), number of parameters and name, type and flags of the parameter to an MS Excel sheet. If there is no information about methods the .NIL object is given back and the function Termination is called.

```
-----
-- GetKnownMethods_AllMethodsOfAnApplication.rex --
-----
    -- Message box with the invitation to insert the ProgID
CALL RxMessageBox "Insert the ProgID!", "Information", "OK", "ASTERISK"
PARSE PULL PROGID -- Hand over of the ProgID in the command window
    -- Instantiation of the Application with its ProgID and hand over to
App = .OLEObject~New(PROGID) -- the variable "App"
method. = App~GetKnownMethods -- Stem creation
    -- Checks if information is available. If not then the .NIL object is
    -- given back and Termination is called. If the information is
    -- available the program goes on.
IF method. == .nil then call Termination else
Excel = .OLEObject~New("Excel.Application") -- Instantiation of Excel
    -- A workbook with a worksheet is added. "1" is the index of the
Worksheet = Excel~Workbooks~Add~Worksheets[1] -- worksheet139
    -- The Visible property is set on true so that Excel can be seen on
Excel~Visible = .true -- the display140
    -- Handover of value "Methodnumber"141
Worksheet~Range("A1")~value = "Methodnumber"
```

<sup>138</sup> c.p. 6.7.

<sup>139</sup> [MS03c]

<sup>140</sup> [MS03d]

<sup>141</sup> [MS03f]

```

-- Handover of value "Name of the method"
Worksheet~Range("B1")~value = "Name of the method"
-- Handover of value "Methoddocumentation"
Worksheet~Range("C1")~value = "Methoddocumentation"
-- Handover of value "Name of the type library"
Worksheet~Range("D1")~value = "Name of the type library"
-- Handover of value "Documentation of the type library"
Worksheet~Range("E1")~value = "Documentation of the type library"
-- Handover of value "Returntype of the method"
Worksheet~Range("F1")~value = "Returntype of the method"
-- Handover of value "MemberID of the method"
Worksheet~Range("G1")~value = "MemberID of the method"
-- Handover of value "Kind of invocation of the method"
Worksheet~Range("H1")~value = "Kind of invocation of the method"
-- Handover of value "Number of Parameters of the method"
Worksheet~Range("I1")~value = "Number of Parameters of the method"
-- Handover of value "Name of the Parameter"
Worksheet~Range("J1")~value = "Name of the Parameter"
-- Handover of value "Type of the Parameter"
Worksheet~Range("K1")~value = "Type of the Parameter"
-- Handover of value "Flags of the Parameter"
Worksheet~Range("L1")~value = "Flags of the Parameter"
-- The cells A1 to G1 get the backgroundcolor with the index "35"
WorkSheet~Range("A1:L1")~Interior~ColorIndex = 35
-- The start value of counter = 2 so that the first line is free for
-- the description. The CounterCell counter is needed that each entry
CounterCell = 2 -- is in a new cell.
DO x = 1 TO method.0 -- DO function over all methods
-- The number of parameters is hand over to the variable
NumberOfParameters = method.x.!PARAMS.0 -- NumberOfParameters
DO CounterCell UNTIL CounterCell > 0 -- DO function which never ends
-- Description of the cells. "A, B, C, D, E, F, G, H, I" stands for
-- the column and "counter" for the row."|" prevents a blank character
a = "A" || counterCell
b = "B" || counterCell
c = "C" || counterCell
d = "D" || counterCell
e = "E" || counterCell
f = "F" || counterCell
g = "G" || counterCell
h = "H" || counterCell
i = "I" || counterCell
Worksheet~Range(a)~value = method.x -- Cell with the methodnumber
-- Cell with the name of the method
Worksheet~Range(b)~value = method.x.!NAME
-- Cell with the documentation of the method
Worksheet~Range(c)~value = method.x.!DOC
-- Cell with the name of the type library
Worksheet~Range(d)~value = method.!LIBNAME

```

```

-- Cell with the documentation of the type library
Worksheet~Range(e)~value = method.!LIBDOC
-- Cell with the returntype of the method
Worksheet~Range(f)~value = method.x.!RETTYPE
-- Cell with the MemberID of the method
Worksheet~Range(g)~value = method.x.!MEMID
-- Checks the kind of invocation because of the number presented with
-- method.i.!INVKIND und selects the appropriate documentation.
SELECT
WHEN method.x.!INVKIND = 1 THEN KindOfInvocation = "Normal method call"
WHEN method.x.!INVKIND = 2 THEN KindOfInvocation = "Property get"
WHEN method.x.!INVKIND = 4 THEN KindOfInvocation = "Property put"
END
-- Cell with the kind of invocation of the method
Worksheet~Range(h)~value = KindOfInvocation
-- Cell with the number of parameter
Worksheet~Range(i)~value = method.x.!PARAMS.0
-- Start value of the counter variable CounterNumberOfParameters is
CounterNumberOfParameters = 1 -- set on "1"
-- DO function that is executed until the Variable
-- CounterNumberOfParameters is less than NumberOfParameters + 1
DO while CounterNumberOfParameters < NumberOfParameters + 1
-- Description of the cell. "J" stands for the column and "counter"
-- for the row. "||" prevents a blank character
j = "J"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the name of the parameter
Worksheet~Range(j)~value = method.x.!PARAMS.CounterNumberOfParameters.!NAME
-- Description of the cell. "K" stands for the column and "counter"
-- for the row. "||" prevents a blank character
k = "K"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the type of the parameter
Worksheet~Range(k)~value = method.x.!PARAMS.CounterNumberOfParameters.!TYPE
-- Description of the cell. "L" stands for the column and "counter" for
-- the row. "||" prevents a blank character
l = "L"|| (CounterCell + CounterNumberOfParameters - 1)
-- Cell with the flag of the parameter
Worksheet~Range(l)~value = method.x.!PARAMS.CounterNumberOfParameters.!FLAGS
-- Increment of counter CounterNumberOfParameters
CounterNumberOfParameters = CounterNumberOfParameters + 1
END -- End command for the third DO function
-- Increment of the counter CounterCell
CounterCell = CounterCell + NumberOfParameters
END -- End command for the second DO function
END -- End command for the first DO function
-- List objects with the column IDs which are accessed in a fix order.
ColumnID = .list ~of("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L")
-- DO function which uses all objects of the list
DO ColumnAutoFit OVER ColumnID
-- Autofit is used for all columns which are provided in the list. The
-- breadth of the columns A, B, C, D, E, F and G are automatically

```

```

-- adapted.142143
Worksheet~Columns(ColumnAutoFit || ":" || ColumnAutoFit)~EntireColumn~AutoFit
END -- End of Do function
EXIT -- Termination of the program
Termination: -- Target if the .NIL object is given back
SAY "There are NO methods!"
EXIT -- Termination of the program

```

**Code 12:** GetKnownMethods\_AllMethodsOfAnApplication.rex

Figure 27 shows a MS Excel sheet with all methods of MS Word.

A	B	C	D	E	F	G
Methodnumber	Name of the method	Methoddocumentation	Name of the type library	Documentation of the type library	Returntype of the method	MemberID of the method
28	Visible	28.IDOC	_Application	ILIBDOC	VT_VOID	
31	ScreenUpdating	31.IDOC	_Application	ILIBDOC	VT_VOID	0000001a
33	PrintPreview	33.IDOC	_Application	ILIBDOC	VT_VOID	0000001b
36	DisplayStatusBar	36.IDOC	_Application	ILIBDOC	VT_VOID	0000001d
42	International	42.IDOC	_Application	ILIBDOC	VT_VARIANT	0000002e
47	UserName	47.IDOC	_Application	ILIBDOC	VT_VOID	
49	UserInitials	49.IDOC	_Application	ILIBDOC	VT_VOID	
51	UserAddress	51.IDOC	_Application	ILIBDOC	VT_VOID	
54	DisplayRecentFiles	54.IDOC	_Application	ILIBDOC	VT_VOID	
56	SynonymInfo	56.IDOC	_Application	ILIBDOC	VT_PTR	0000003b
59	DefaultSaveFormat	59.IDOC	_Application	ILIBDOC	VT_VOID	
62	ActivePrinter	62.IDOC	_Application	ILIBDOC	VT_VOID	
65	CustomizationContext	65.IDOC	_Application	ILIBDOC	VT_VOID	
67	KeysBoundTo	67.IDOC	_Application	ILIBDOC	VT_PTR	
68	FindKey	68.IDOC	_Application	ILIBDOC	VT_PTR	
70	Caption	70.IDOC	_Application	ILIBDOC	VT_VOID	
73	DisplayScrollBars	73.IDOC	_Application	ILIBDOC	VT_VOID	
75	StartupPath	75.IDOC	_Application	ILIBDOC	VT_VOID	
79	Left	79.IDOC	_Application	ILIBDOC	VT_VOID	
81	Top	81.IDOC	_Application	ILIBDOC	VT_VOID	
83	Width	83.IDOC	_Application	ILIBDOC	VT_VOID	
85	Height	85.IDOC	_Application	ILIBDOC	VT_VOID	0000005a
87	WindowState	87.IDOC	_Application	ILIBDOC	VT_VOID	0000005b
89	DisplayAutoCompleteTips	89.IDOC	_Application	ILIBDOC	VT_VOID	0000005c
92	DisplayAlerts	92.IDOC	_Application	ILIBDOC	VT_VOID	0000005e
95	StatusBar	95.IDOC	_Application	ILIBDOC	VT_VOID	
98	DisplayScreenTips	98.IDOC	_Application	ILIBDOC	VT_VOID	
100	EnableCancelKey	100.IDOC	_Application	ILIBDOC	VT_VOID	
105	DefaultTableSeparator	105.IDOC	_Application	ILIBDOC	VT_VOID	
107	ShowVisualBasicEditor	107.IDOC	_Application	ILIBDOC	VT_VOID	0000006c

**Figure 27:** Snapshot of MS Excel<sup>144</sup>

## 6.9. GetOutParameters<sup>145</sup>

The GETOUTPARAMETERS method offers an array with the results of the single out parameters of the OLE object.

<sup>142</sup> [MS03e]

<sup>143</sup> [MS03g]

<sup>144</sup> Part of MS Office

<sup>145</sup> [IBM01,p290]

If there are no out parameters, the `.NIL` object is given back. An out parameter is an argument to the OLE object which is filled in by the OLE object. If this is impossible in Rexx due to data encapsulation, the results are positioned in the array. The order in the out array is from left to right.

Syntax:

GETOUTPARAMETERS

## 7.Tools

There are several tools, which support the work with Automation. These tools offer for example information about the member functions of objects. The three tools METHINFO.rex, OLEInfo.rex and RGF\_OLEInfo.hta are represented.

### 7.1. METHINFO.rex<sup>146</sup>

The tool `METHINFO.rex` that is contained in the path `...\ObjREXX\SAMPLES\OLE\METHINFO` offers information about the methods of an OLE object. Therefore, the script `MAIN.rex` must be executed and the ProgID must be inserted.

### 7.2. OLEInfo.rex<sup>147</sup>

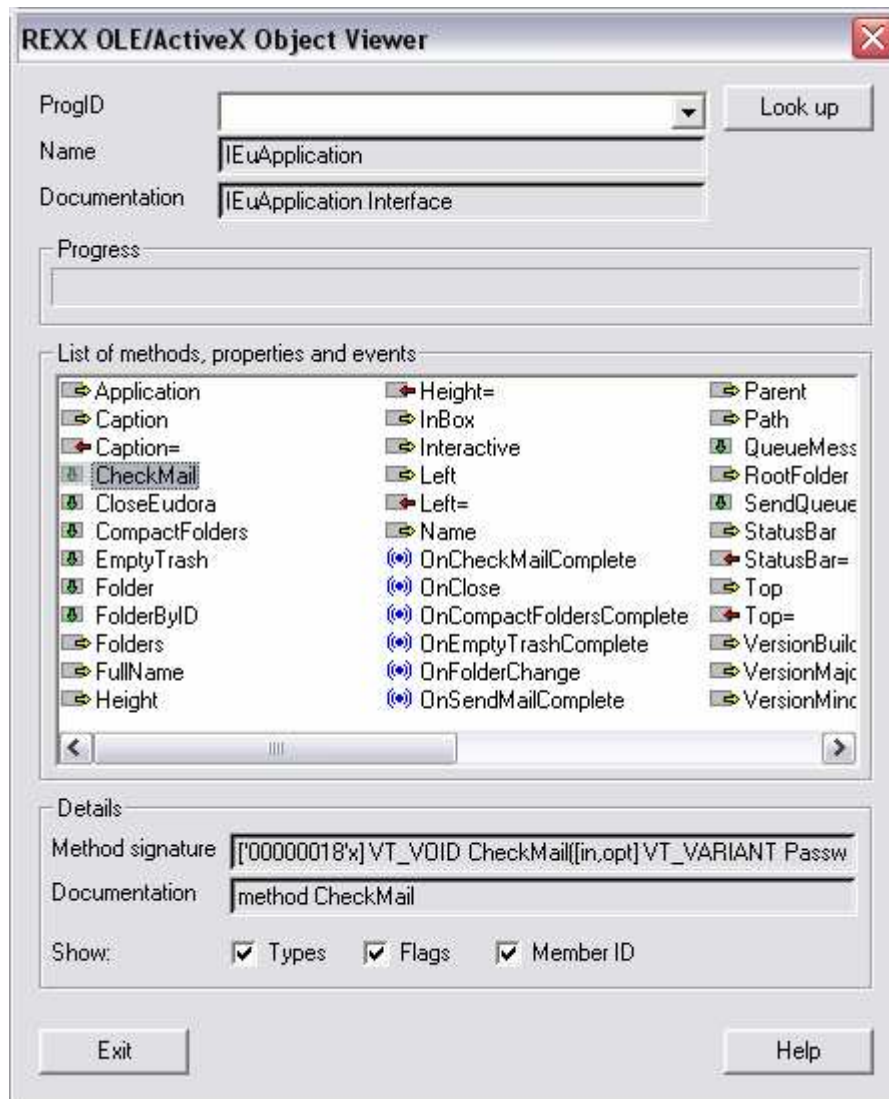
The tool `OLEInfo.rex` is contained in the path `...\ObjREXX\SAMPLES\OLE\OLEINFO`. It is a Rexx OLE/ActiveX object viewer. It offers information about the automated OLE objects with its ProgID and with its exposed methods, events and properties. Information about the method signature with flags, types and memberID and the method documentation is available and the method can be invoked. Figure 28 illustrates the graphical user interface of the IBM OLE/ActiveX object viewer.

---

<sup>146</sup> Taken from METHINFO.rex

<sup>147</sup> [IBM02b]





**Figure 28:** OLE/ActiveX Object Viewer with functions of Qualcomm Eudora<sup>148149</sup>

For more information look up in the `HELP.TXT` file in the same directory.

### 7.3. RGF\_OLEInfo.hta<sup>150</sup>

This tool created by Prof. Dr. Rony G. Flatscher offers a rich set of information about the installed OLE/ActiveX-COM objects.

There is data about the CLSID, ProgID, LocalServer32, InProcHandler32, the version independent ProgID, a short description of OLE object and the date of the registry

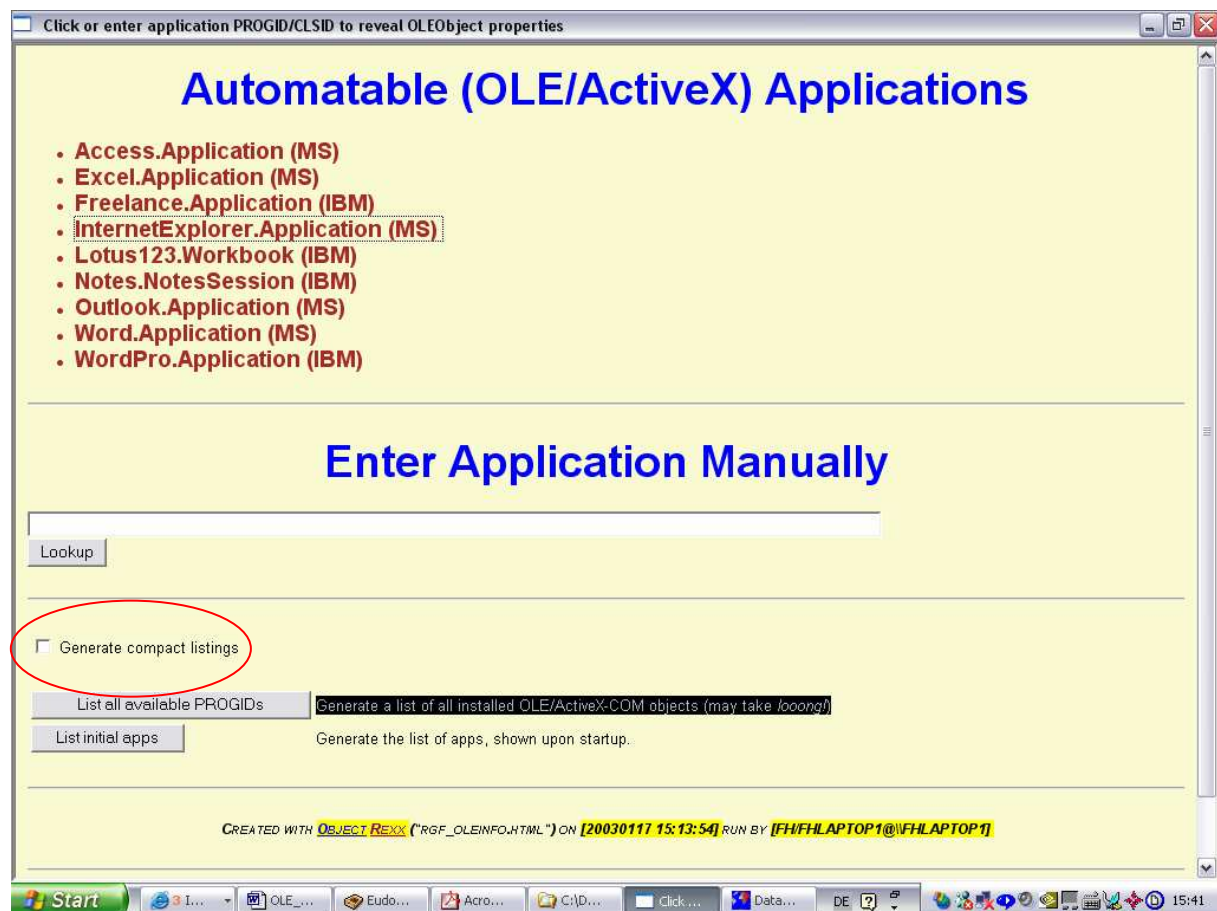
<sup>148</sup> User-interface of the IBM OLE/ActiveX Object Viewer

<sup>149</sup> To automate Qualcomm Eudora 5.2, enable in the menu Tools->Options Automation

<sup>150</sup> Taken from RGF\_OLEInfo.hta

entry. It offers information about the methods with documentation, arguments and return value. Information is available for read-only properties with documentation and return value and for write-only and read/write properties with documentation, arguments and return value. Methods with unknown invocation type properties and events are also described with documentation, arguments and return values. All constants with their name and value are shown.

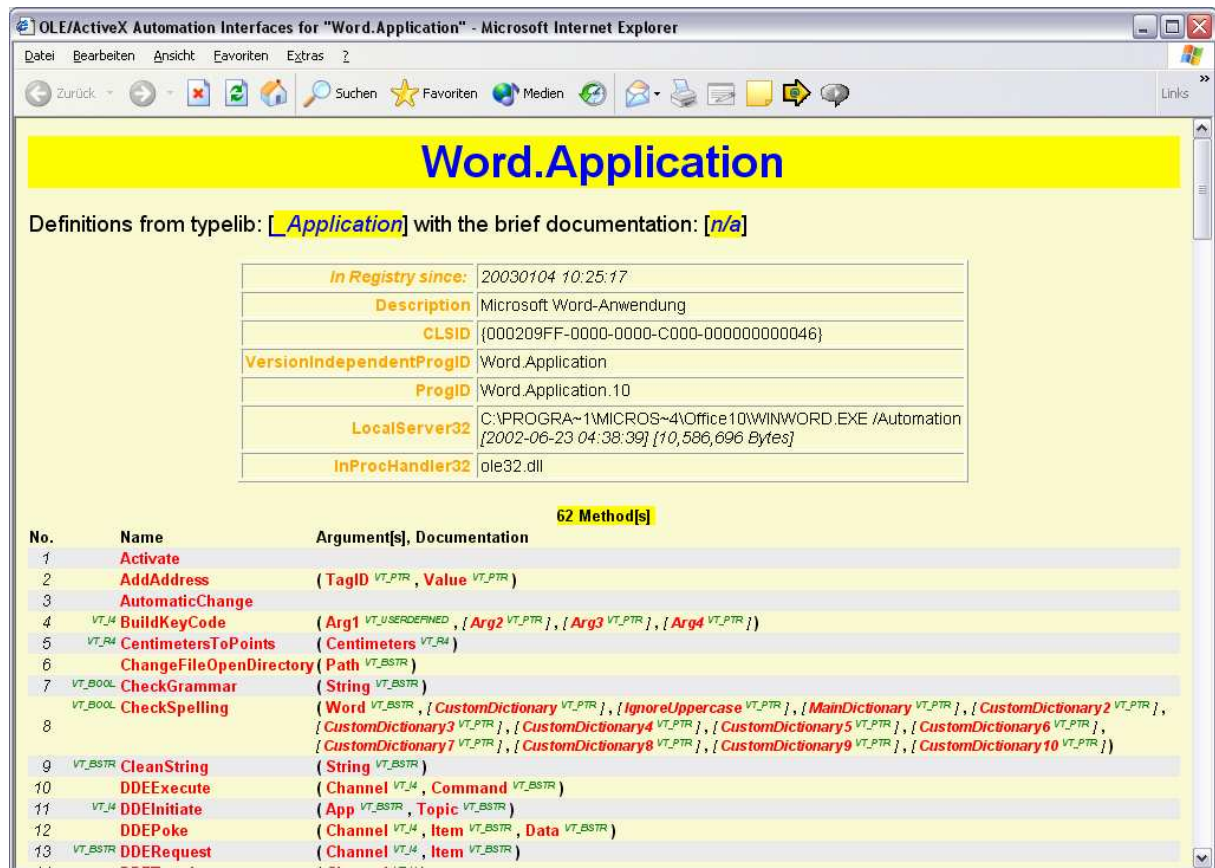
Figure 29 illustrates the start page of this tool. If the box in the red circle is clicked the user gets a compact listing.



**Figure 29:** Snapshot of the start page of “rgf\_oleinfo.hta”<sup>151</sup>

Such a compact listing of the MS Word member functions is shown in Figure 30.

<sup>151</sup> User interface of RGF\_OLEInfo.hta



**Figure 30: Snapshot of “RGF\_OLEInfo.hta” with compact listing.**

To use this OLE-/Active-X Query Tool Object Rexx must be updated to version 2.1.2<sup>152</sup>. Probably it is necessary to decrease the security level<sup>153</sup> of the MS Internet Explorer (version 6.0 or higher required).

Look up the readme.html file to get more information about this tool and how to get it<sup>154</sup>. After installing execute the file “RGF\_OLEInfo.hta” to run the tool<sup>155</sup> [Fla02a,p33].

<sup>152</sup> [http://www-](http://www-1.ibm.com/support/docview.wss?rs=0&q=Object+Rexx&uid=swg24003624&loc=en_US&cs=utf-8&cc=us&lang=en)

1.ibm.com/support/docview.wss?rs=0&q=Object+Rexx&uid=swg24003624&loc=en\_US&cs=utf-8&cc=us&lang=en

<sup>153</sup> MS IE (English version): Go to menu Tools->InternetOptions->SecurityLevel->CustomLevel

<sup>154</sup> <http://wi.wu-wien.ac.at/rgf/rexx/orx13/tmp/readme.html>

<sup>155</sup> c.p. 9.1.3.

## 8. Useful Object Rexx Classes

Object Rexx provides some useful Object Rexx classes, which facilitate the work with Automation. These classes enable the access of the registry, access of windows, managing of program groups, access of menus, using the clipboard and managing of event log data. In this section the classes WindowObject, MenuObject, WindowsProgramManager, WindowsManager, WindowsClipboard, WindowsRegistry and WindowsEventLog are discussed. These classes are also described in the Object Rexx Online Reference<sup>156</sup>.

### 8.1. WindowObject

Definition of properties of Window objects that are available for Object Rexx programs. This class enables the sending of Windows messages (e.g. WM\_COMMAND), mouse clicks, texts, keys etc.. Menus can be queried and Windows can be increased, moved, represented or put in the foreground [Fla02a,p2].

### 8.2. MenuObject

This class enables the locating of menus, submenus, menu position, menu ID etc [Fla02a,p2].

### 8.3. Object Rexx Classes Remoting the User Interface

The following classes support the remoting of user interfaces.

#### 8.3.1. WindowsProgramManager

This Object Rexx class requires the directive `::REQUIRES "WINSYSTEM.CLS"`. It shows program groups. It enables the definition and deletion of program groups and the contained files [Fla02a,p3]

#### 8.3.2. WindowsManager

This Object Rexx class requires the directive `::REQUIRES "WINSYSTEM.CLS"`. This class enables the pressing of buttons, choosing of menu items and the sending of

---

<sup>156</sup> Object Rexx Workbench menu: Help->Online Reference

text in a Window. It is also possible to find Windows by the text of the title or by coordinates and to choose the active Window [Fla02a,p3]

### **8.3.3. WindowsClipboard**

This Object Rexx class requires the directive `::REQUIRES "WINSYSTEM.CLS"`. This class enables the usage of the clipboard to for example transmitting text<sup>157</sup> [Fla02a,p4]

### **8.3.4. WindowsRegistry**

This Object Rexx class requires the directive `::REQUIRES "WINSYSTEM.CLS"`. With this class it is possible to query, define or delete Windows registry entries<sup>158</sup> [Fla02a,p4].

### **8.3.5. WindowsEventLog**

This Object Rexx class requires the directive `::REQUIRES "WINSYSTEM.CLS"`. This class enables reading, deleting and writing of event log data on Windows machines or across machine border [Fla02a,p4].

---

<sup>157</sup> c.p. 12.1.3.

<sup>158</sup> c.p. 2.6.3.

## 9. Embedding Object Rexx in HTML or XML

It is possible to insert Object Rexx code in so-called markup language like HTML or XML. Thereby the tags `<Script Language = "Object Rexx">` and `</Script>` include the Object Rexx script code [Fla02a,p46]. This chapter describes some basic concepts. These are Tag, Document Type Definition, HTML, XML, Cascading Style Sheets, Document Object Model and the Microsoft Internet Explorer.

For older Object Rexx versions the directive `"::REQUIRES OREXXOLE.CLS"` is not necessary in this case, because Object Rexx embedded in HTML is running over ActiveX Scripting [En02].

Commands can be put at each position in the document. The execution of commands ensues the document order. Public routines can be called from each location in the script. Commands with attributes for events are executed if the event is fired [Fla02a,p48].

### 9.1. Tag

A tag encloses a text. There is first an opening tag `<some_tag_name>` and after the text comes a closing tag `</some_tag_name>`. This enables the analysis of texts because in this way it is possible to ascertain which text parts are enclosed from which tags [Fla02a,p36].

### 9.2. Document Type Definition (DTD)

The Document Type Definition defines tags and their attributes. A content model regulates the hierarchical structure, how the tags are fit into each other and how often an element can be used. An instance of a DTD is a document that is marked up due to the DTD rules. Such a document is called "validated" [Fla02a,p37].

### 9.3. HTML (Hypertext Markup Language)

HTML is a markup language for the WWW.

A so-called HTML-browser parses a HTML document and formats the text due to the tags. For DTD version 4.01 is used and it is SGML-based (Standard Generalized Markup Language [Mü01]). Thereby it is possible to use the names of tags and

attributes irrespective of capitalization and to define exclusion rules. In some cases it is possible to leave end-tags if they are set clearly because of DTD rules [Fla02a,p38].

The content of a HTML is enclosed into the tags `<html>` and `</html>` which is the root element of a HTML file. The HTML file consists of the two parts `<head>` with the head data like the title of the file and the `<body>` with the content which is shown in the browser window. A comment is used as follows: `<!--This is a comment -->` [Mü01a]

If the file extension is not named `.html` but `.hta` (HTML Application) a secure, local execution is enabled<sup>159</sup> [Fla02a,p51].

## 9.4. XML (eXtensible Markup Language)

XML is a simplified version of SGML (Standard Generalized Markup Language).

XML enables the definition of DTD for markup languages. It is necessary to always set the end tag. The values of attributes can be enclosed with apostrophes and double quotation marks. Empty elements can be marked and the names must be marked exactly in the defined capitalization [Fla02a,p39]. It is possible to omit DTDs. The DTD can be derived if it is wellformed. The opening tags must have adequate end-tags. All tags have to fit into each other and the tags needn't overlap<sup>160</sup> [Fla02a,p40].

## 9.5. Cascading Style Sheets (CSS)

CSS enables the definition of formatting rules for elements [Fla02a,p40].

## 9.6. Document Object Model (DOM)

The Document Object Model (DOM) is a norm of the W3 Consortium. Thanks to DOM it is possible to change the elements of a web site dynamically [Mü01b]. In the Microsoft context DOM is called DHTML [Fla02c,p12]].

---

<sup>159</sup> c.p. 7.3. and 12.1.2.

<sup>160</sup> c.p. 13.8.3.

A HTML/XML file is parsed and a parse tree is generated with the elements as nodes as demonstrated in figure 31. There are Application Programming Interfaces (API) for creation, querying, changing or deleting of nodes in the tree and for intercepting events including the attributes of the tags. These APIs catch the events which occur if there are keyboard- or mouse-actions or if there are events which are created by the application like “document loaded” [Fla02a,p44].

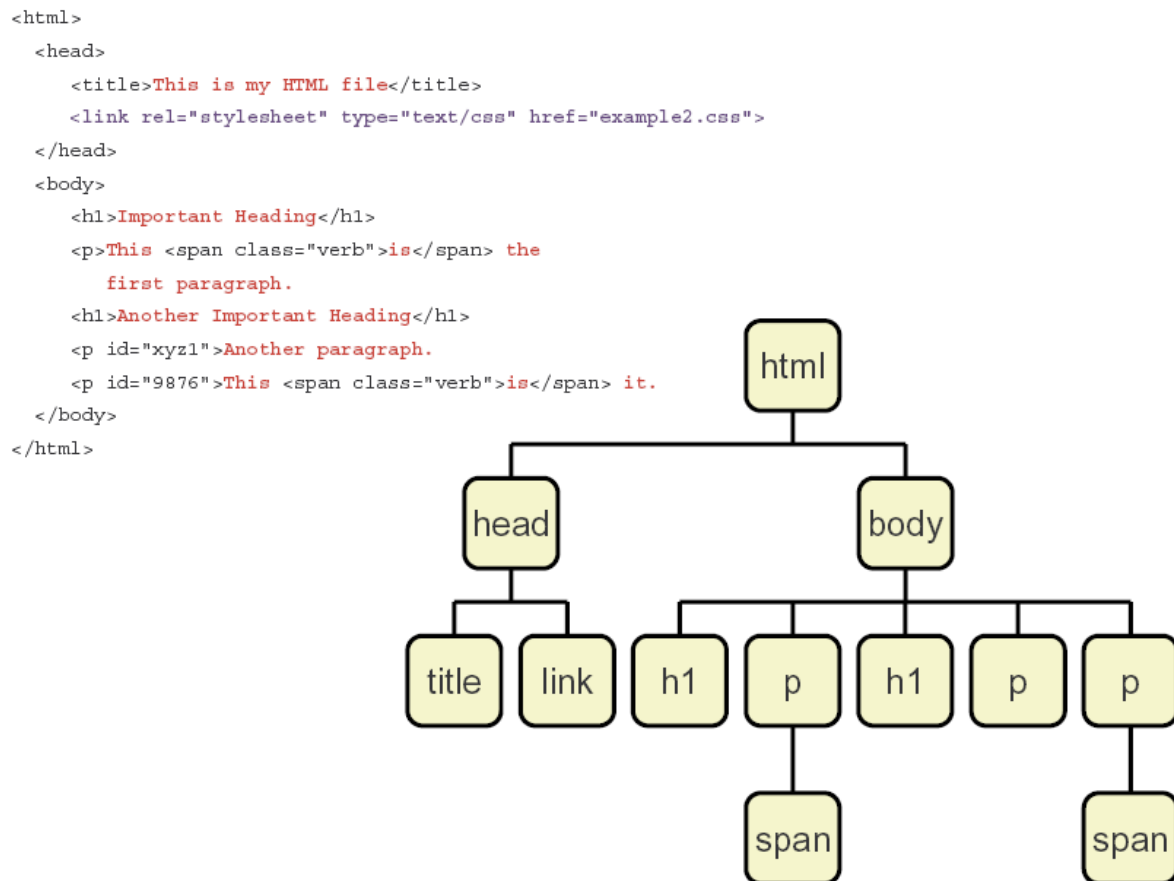


Figure 31: Example for DOM<sup>161</sup>

## 9.7. Microsoft Internet Explorer (MSIE)

The MS Internet Explorer is a Windows Scripting Host. It offers the OLE-object “window” which is a COM object for the implementation of DCOM (DHTML – dynamic HTML). All OLE-object-properties of “window” like “document” are used by the MSIE. In this way functions and methods like “all” or “tables” can be used.

<sup>161</sup> Taken from [Fla02a,p45]



Elements can be added, changed or erased. The MS Internet Explorer controls the execution of embedded scripts<sup>162</sup> [Fla02a,p47].

### 9.7.1. Embedding an Object Rexx Script in HTML

Code 13 demonstrates the embedding of Object Rexx in HTML.

Thereby a text input is shown. If the button is clicked an event is fired and the text is transferred to the head where a routine is called. In this routine the essential Object Rexx code is contained. There the text is hand over. After that, this script code runs Word and writes the text.

```
<html>
<head>
<title>Demonstration of OLE with Object Rexx</title>
  <!-- Beginning of the script code -->
<script language="Object Rexx">
  -----
  -- Embedding Object Rexx in HTML --
  -----

  -- Beginning of the routine doTheWork. It is referenced from the body
  -- if the button is pressed or clicked."Public" enables the transfer-
  -- ring of the data from the input area in the body to the routine
  ::routine doTheWork public
    -- Transfer of data to the variable text. Thereby refers the document
    -- object to the content which is shown in a browser window. "All" is
    -- an object which enables the access to single elements and content
    -- of HTML documents. This is part of DHTML. "Text" is the
    -- identifier."Value" is the content of the identifier.
    text = document~all~text~value
    Word = .OLEObject~New("Word.Application") -- Instantiation of Word
    -- The visibility of Word is set on true so that Word can be seen
    Word~Visible = .TRUE -- on the display
    Document = Word~Documents~Add -- A new Word document is added
    Word~Selection~TypeText(text) -- The inserted text is written to Word
    -- End of the script code
</script>
  <!-- End of the head -->
</head>
  <!-- Beginning of the body with the backgroundcolor red -->
<body bgcolor="red">
  <!-- The content after that text is centred -->
  <center>
```

<sup>162</sup> c.p. 13.7.4.1.

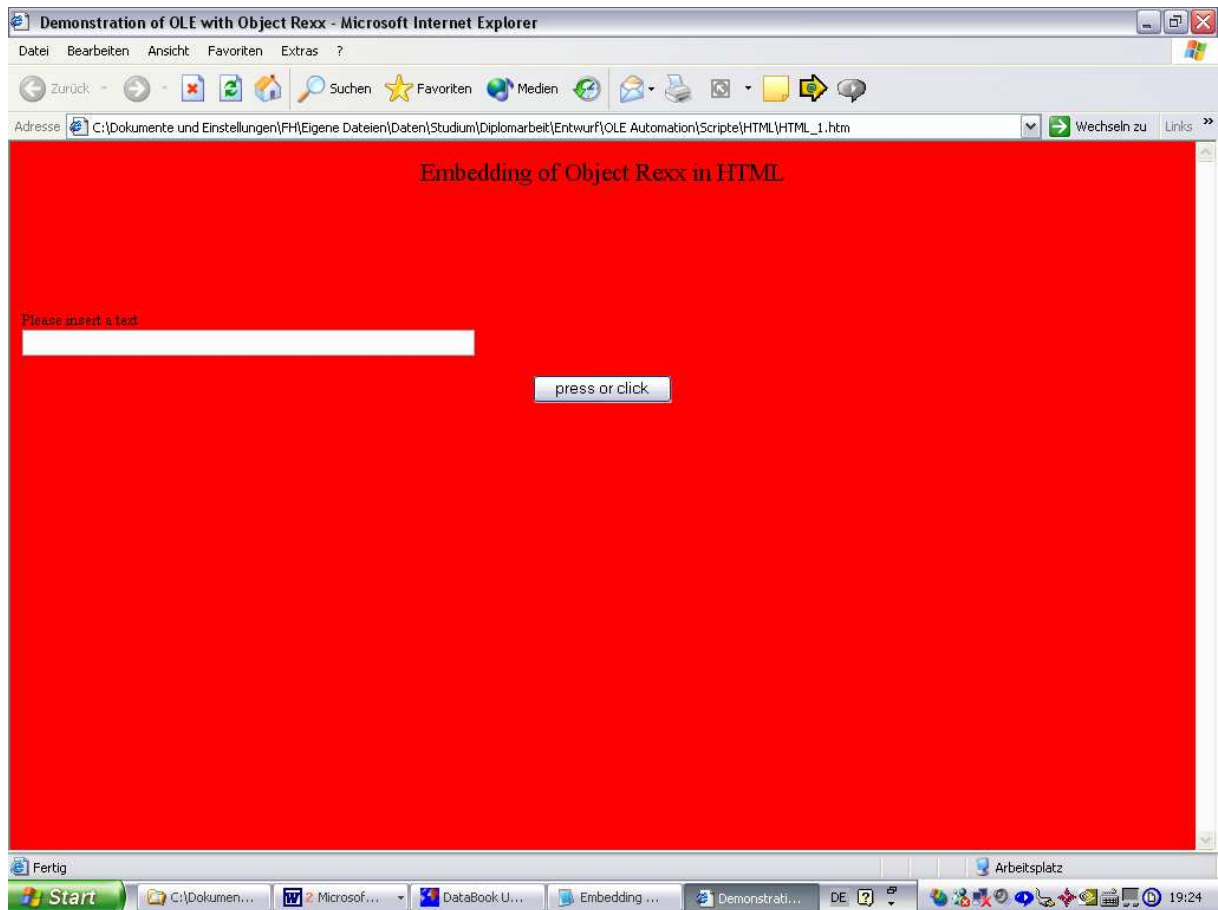
```

        <!-- The font has the size 5 and the text"Embedding of Object Rexx in
            HTML" is written -->
<font size=5>Embedding of Object Rexx in HTML</font>
        <!-- End of centring -->
</center>
        <!-- Seven line breaks -->
<br>
<br>
<br>
<br>
<br>
<br>
<br>
        <!-- Text with the demand to insert a text -->
<name>Please insert a text</name>
<br>
        <!-- Input area of the type text with the internal name "Text". -->
        <!-- The length of the area which is visible is "60".The internal -->
        <!-- length is"70".To reference to this object the id"text"is used-->
<input name="Text" size=60 maxlength=70 id="text">
<br>
<br>
<center>
        <!-- Input area of the type button.The text on the button is"press or
            click". The phrase"language="Object Rexx"embeds Object Rexx.If
            one of the two event handler "onmouseup"or "onkeypress" is
            executed the routine doTheWork is called -->
<input type=button value="press or click"
        language="Object Rexx"
        onmouseup="call doTheWork"
        onkeypress="call doTheWork">
</center>
</body>
</html>

```

**Code 13: Embedding Object Rexx in HTML.htm**

Figure 32 shows the MS Internet Explorer with the user interface of the file Embedding Object Rexx in HTML.htm.

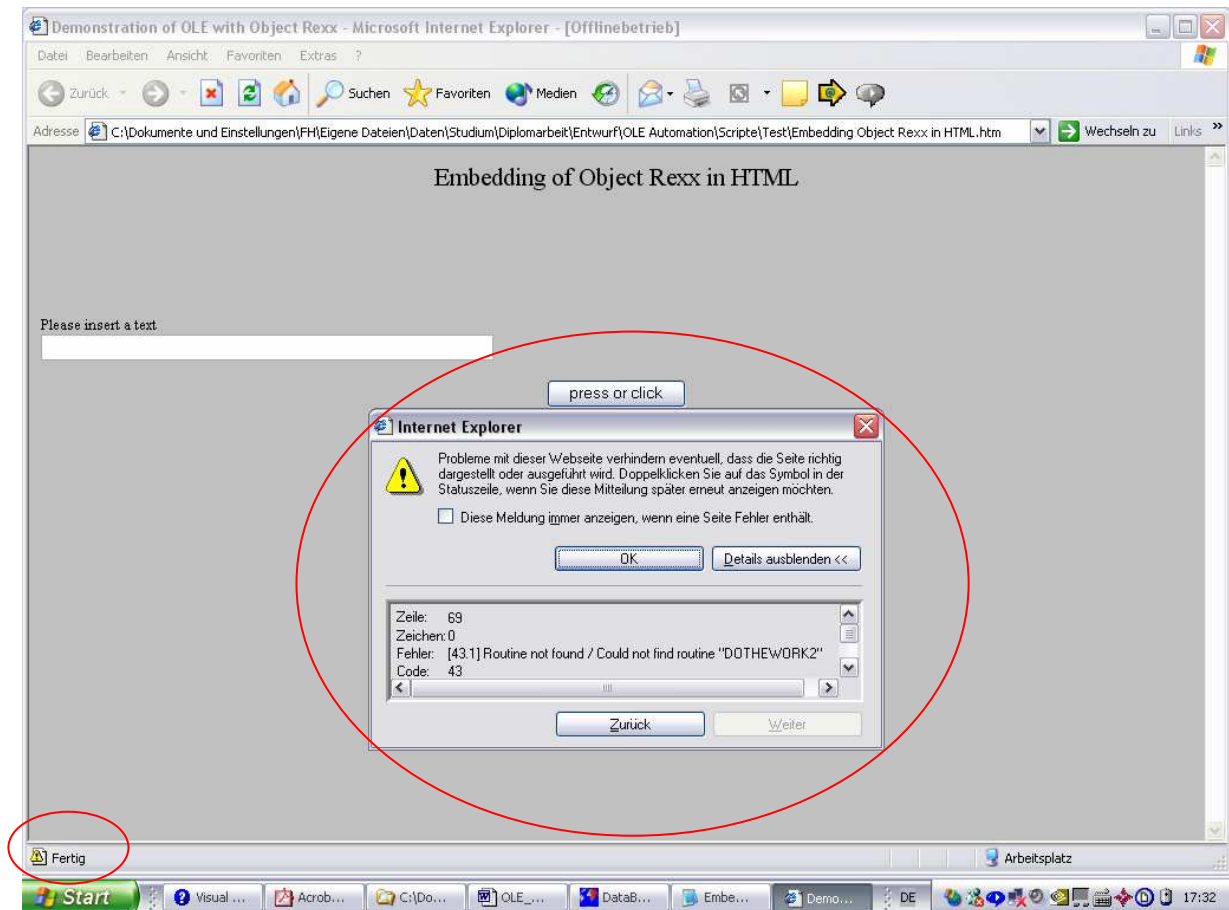


**Figure 32:** Snapshot of Embedding Object Rexx in HTML.htm.<sup>163</sup>

### 9.7.2. MSIE and Error

If there is an error in an Object Rexx script that is embedded in HTML, then in the left lower corner of the browser window is an alert sign (small red circle in figure 33). Double click this sign to get more information about the error (big red circle in figure 33).

<sup>163</sup> Part of MS Windows XP



**Figure 33:** Error handling with the MSIE.<sup>164</sup>

<sup>164</sup> Part of MS Windows XP

## 10. WMI<sup>165</sup>

The Windows Management Instrumentation (WMI) offers control and management information in an enterprise environment. It is a component of Microsoft Windows operating system [MLWMIa]. The scripts show for example all services running on the machine or shutting down the machine.

There are some examples for the implementation of WMI with Object Rexx in the folder ...\\ObjREXX\\SAMPLES\\OLE\\WMI.

### 10.1. Win32\_Service<sup>166</sup>

The first example shows all services in an MS Excel sheet which are on the machine.

Therefore, an object of the WMI class is obtained with the `GETOBJECT` method. First, there is the Visual Basic Script code `OtherScript 3` which handles this subject.

```
Set ServiceSet = _
GetObject("winmgmts:{impersonationLevel=impersonate}")._
    InstancesOf("Win32_Service")
for each Service in ServiceSet
    WScript.Echo Service.Description
Next
```

#### ***OtherScript 3: VBS script code for Win32\_Service<sup>167</sup>***

Code 14 contains the Object Rexx script which writes service name and service description to an MS Excel sheet additional to the VBS code.

As moniker<sup>168</sup> is `winmgmts` used. That moniker tells Windows Script Host to use the WMI objects [MLWMIj].

---

<sup>165</sup> c.p. 2.3.4. and 6.5.

<sup>166</sup> [MLWMIb]

<sup>167</sup> Taken from [MLWMIb]

<sup>168</sup> c.p. 2.3.4.

```

-----
-- WMI_ListAllServicesOnTheSystem.rex --
-----

Excel = .OLEObject~new("Excel.Application") -- Instantiation of Excel
    -- A workbook with a worksheet is added."1" is the index of the
    -- worksheet169
Worksheet = Excel~Workbooks~Add~Worksheets[1]
    -- The Visible property is set on true so that excel can be seen on the
    -- display170
Excel~Visible = .true
    -- Hands over the value "Servicename"
Worksheet~Range(A1)~value = "Servicename"
    -- Hands over the value "Servicedescription"
Worksheet~Range(B1)~value = "Servicedescription"
    -- Sets the counter variable a on "2" because it starts in the Excel
    -- sheet in the second line
a = 2
    -- Sets the counter variable b on "2" because it starts in the Excel
    -- sheet in the second line
b = 2
    -- Get an object from the WMI service (WinMgmt). The phrase
    -- "{impersonationLevel=impersonate}" informs the system to treat the
    -- current login credentials as those which are used for data or
    -- executing methods.
WMIObject=.OLEObject~GETOBJECT("winmgmts:{impersonationLevel=impersonate}")
    -- DO functions over all instances of the win32_Service class
DO service OVER WMIObject~InstancesOf("win32_Service")
    -- Used for the location of the cell. || is necessary that there is no
    -- empty space between the column identifier and the line number.
aa = "A" || a
bb = "B" || b
    -- Inserts to the adequate cell the name of the service
Worksheet~Range(aa)~value = service~name
    -- Inserts to the adequate cell the description of the service
Worksheet~Range(bb)~value = service~description
    -- Increment the counters a and b
a = a + 1
b = b + 1
END -- End of the DO functions
    -- This command adapts automatically the breadth of the A column
WorkSheet~Columns("A:A")~EntireColumn~AutoFit

```

<sup>169</sup> [MS03c]

<sup>170</sup> [MS03d]

```

-- The width of column B is set on 100
WorkSheet~Columns("B")~ColumnWidth = 100
-- All cells of the column B wrap their text
WorkSheet~Columns("B")~WrapText = .true
-- Sets the font type on bold in the cells A1 and B1
WorkSheet~Range("A1:B1")~Font~Bold = .true

```

**Code 14:** WMI\_ListAllServicesOnTheSystem.rex

## 10.2. Win32\_OperatingSystem

Code 15 is a short demonstration of the Win32\_OperatingSystem class.

Therefore, some properties of the operating system are shown. The Windows version, the Windows directory, the description of the machine, the encryption level and the serial number are listed [MLWMIc].

As moniker<sup>171</sup> is winmgmts used. That moniker tells Windows Script Host to use the WMI objects [MLWMIj].

```

-----
-- WMI_Win32_OperatingSystem.rex --
-----

-- Get an object from the WMI service (WinMgmt). The phrase
-- "{impersonationLevel=impersonate}" informs the system to treat the
-- current login credentials as those which are used for data or
-- executing methods.
WMIObject=.OLEObject~GETOBJECT("winmgmts:{impersonationLevel=impersonate}")
-- DO functions over all instances of the win32_OperatingSystem class
DO a OVER WMIObject~InstancesOf("Win32_OperatingSystem")
SAY "The Windows Version is: " a~Version
SAY "Windowsdirectory: " a~WindowsDirectory
SAY "Description: " a~Description

```

<sup>171</sup> c.p. 2.3.4.

```
SAY "Encryption Level: " a~EncryptionLevel
SAY "Serial number: " a~SerialNumber
END
```

**Code 15:** WMI\_Win32\_OperatingSystem.rex<sup>172</sup>

### 10.3. Win32\_DiskPartition

Code 16 shows some properties of the Win32\_DiskPartition class that offers information about a partition with index number, name and description [MLWMIId].

As moniker<sup>173</sup> is winmgmts used. That moniker tells Windows Script Host to use the WMI objects [MLWMIj].

```
-----
-- WMI_Win32_DiskPartition.rex --
-----

-- Get an object from the WMI service (WinMgmt). The phrase
-- "{impersonationLevel=impersonate}" informs the system to treat the
-- current login credentials as those which are used for data or
-- executing methods.
WMIObject=.OLEObject~GETOBJECT("winmgmts:{impersonationLevel=impersonate}")
-- DO functions over all instances of the Win32_DiskPartition class
DO dp OVER WMIObject~InstancesOf("Win32_DiskPartition")
-- Writes some properties of the Win32_DiskPartition class
SAY "Index number: " dp~Index " Name: " dp~name " Description: " -
dp~description
END
```

**Code 16:** WMI\_Win32\_DiskPartition.rex<sup>174</sup>

### 10.4. Win32\_LogicalDisk<sup>175176</sup>

The class Win32\_LogicalDisk is demonstrated with the following script. The quantity of free space on a local drive is determined. The properties which can be used in this example are accurately selected with the ExecQuery method. This script shows all drives that have less than 20 free space.

<sup>172</sup> [MLWMIc]

<sup>173</sup> c.p. 2.3.4.

<sup>174</sup> [MLWMIId]

<sup>175</sup> [IBM02c]

<sup>176</sup> [MLWMIe]



OtherScript 4 is a Visual Basic Script code script that handles also this subject.

```
Set DiskSet = GetObject("winmgmts:{impersonationLevel=impersonate}")_
    .ExecQuery("select FreeSpace,Size,Name from Win32_LogicalDisk where DriveType=3")
for each Disk in DiskSet
    If (Disk.FreeSpace/Disk.Size) < 0.20 Then
        WScript.Echo "Drive " + Disk.Name + " is low on space."
    End If
Next
```

**OtherScript 4: VBS script code for Win32\_LogicalDisk<sup>177</sup>**

Code 17 also lists the name and volume name of disks, which have enough space for contrast to OtherScript 4, which only tells the data of disks, which are low of space.

As moniker<sup>178</sup> is winmgmts used. That moniker tells Windows Script Host to use the WMI objects [MLWMI].

```
-----
-- WMI_Win32_LogicalDisk.rex --
-----

    -- Get an object from the WMI service (WinMgmt). The phrase
    -- "{impersonationLevel=impersonate}" informs the system to treat the
    -- current login credentials as those which are used for data or
    -- executing methods.
WMIObject=.OLEObject~GETOBJECT("winmgmts:{impersonationLevel=impersonate}")
    -- The properties FreeSpace, Size, Name and VolumeName from the class
    -- Win32_LogicalDisk are selected.The command"where"allows only drives
    -- of the type "3" which are local discs179.
    -- Only the selected properties can be used for queries.
DiskSet = WMIObject~ExecQuery("select FreeSpace,Size,Name,VolumeName " -
    "from Win32_LogicalDisk where DriveType=3")
    -- DO function over all instances of the Win32_LogicalDisk class
DO Disk OVER DiskSet
    -- The percentage of free space is checked if it is less than 20%.
```

<sup>177</sup> Taken from [MLWMIe]

<sup>178</sup> c.p. 2.3.4.

<sup>179</sup> [MLWMI]f

```

IF (Disk~FreeSpace/Disk~Size) < 0.20 THEN
    SAY "Drive " Disk~Name " with volume name " -
        Disk~VolumeName " is low on space."
    ELSE SAY "Drive " Disk~Name " with volume name " -
        Disk~VolumeName " has enough space."
END

```

**Code 17: WMI\_Win32\_LogicalDisk.rex**

## 10.5. Win32\_Process<sup>180</sup>

This script shows how to start a new process.

In this file the calculator is started. Therefore, a COM moniker<sup>181</sup> notation is used to reference the class. Then the class object is called itself because a new instance of the Win32\_Process class must be generated if a new process is generated. In this context a so-called static method is used which is referenced to the class definition itself. The methods in the foregoing scripts are so-called dynamic methods which are referenced to the individual instances.

The OtherScript 5 which contains Visual Basic Script code handles also this subject.

```

set process = _
GetObject("winmgmts:{impersonationLevel=impersonate}!Win32_Process")
result = process.Create ("notepad.exe",null,null,processid)
WScript.Echo "Method returned result = " & result
WScript.Echo "Id of new process is " & processed

```

**OtherScript 5: VBS script code for Win32\_Process<sup>182</sup>**

Code 18 doesn't launch the notepad like OtherScript 5 and doesn't tell the returned result the ID of the new process, but it launches the calculator.

```

-----
-- WMI_LaunchANewProcess.rex --
-----

-- Get an object from the WMI service (WinMgmt). The phrase
-- "{impersonationLevel=impersonate}" informs the system to
-- treat the current login credentials as those which are
-- used for data or executing methods. A COM moniker
-- notation (!Win32_Process) is used to reference the

```

<sup>180</sup> [MLWMIg]

<sup>181</sup> c.p. 2.3.4.

<sup>182</sup> Taken from [MLWMIg]

```

-- Win32_Process class.
Process = -
.OLEObject~GETOBJECT("winmgmts:{impersonationLevel=impersonate}!Win32_Process")
-- Launches the calculator by using the Win32_Process class
-- object itself. "Create" is a so-called static method.
Process~Create("calc")

```

**Code 18:** WMI\_LaunchANewProcess.rex

## 10.6. Win32Shutdown<sup>183</sup>

The following script demonstrates the Win32Shutdown method of the Win32\_OperatingSystem class.

This script will shut down the local machine. **Caution:** Close respectively save all running processes before running this script.

First the OtherScript 6, a Visual Basic Script code script, handles also this subject.

```

Set colOperatingSystems = _
GetObject("winmgmts:{(Shutdown)}").ExecQuery("Select * from Win32_OperatingSystem")
For Each objOperatingSystem in colOperatingSystems
    ObjOperatingSystem.Win32Shutdown(1)
Next

```

**OtherScript 6:** VBS script for the demonstration of Win32Shutdown<sup>184</sup>

Code 19 is the Object Rexx version of this item.

As moniker<sup>185</sup> is winmgmts used. That moniker tells Windows Script Host to use the WMI objects [MLWMIh].

```

-----
-- WMI_Win32Shutdown.rex --
-----

-- Get an object from the WMI service (WinMgmt). The phrase
-- "{(Shutdown)}" offers the privilege to perform a remote186.
WMIObject = .OLEObject~GETOBJECT("winmgmts:{(Shutdown)}")
-- All instances are selected from the Win32_OperatingSystem class
colOperatingSystems = WMIObject~ExecQuery("select * from Win32_OperatingSystem")
-- DO function over all instances of the Win32_OperatingSystem class

```

<sup>183</sup> [SGa]

<sup>184</sup> Taken from [SGa]

<sup>185</sup> c.p. 2.3.4.

<sup>186</sup> [MLWMIh]

```
DO colOperatingSystem OVER colOperatingSystems
    -- The Win32Shutdown method is executed with the flag 1 which means
    -- "Shutdown". The flag "2" e.g. means "Reboot"187.
colOperatingSystem~Win32Shutdown(1)
END -- End of DO function
```

**Code 19:**      **WMI\_Win32Shutdown.rex**

---

<sup>187</sup> [MLWMIi]

## 11. Automation of Microsoft Agent Technology

The Microsoft Agent technology enables a new user interface to communicate with the computer. Thereby it is possible to embed animated characters to Web pages and applications. These agents can be accessed by speech recognition to receive spoken commands. This section introduces the MS Agent technology, an overview over many functions and its usage of events is shown. Figure 34 illustrates the characters Peedy and Merlin that are part of the MS Agent technology.



**Figure 34:** Snapshot of MSAgents<sup>188</sup>

The agents can also speak. This is possible with recorded audio or via a text-to-speech (TTS) engine [MLAGTa]. Such a TTS engine can be downloaded for several languages from the Microsoft Agent download page<sup>189</sup>. The Microsoft Agent technology is already installed on Microsoft Windows XP, Windows and Windows Me systems. For other systems, the Microsoft Agent technology has to be downloaded from the Microsoft Agent download page. On this page are also some other characters downloadable. There is the SAPI 4.0 runtime support available that is necessary for Windows XP machines. For using the speech recognition the speech recognition engine must be downloaded. There is only an US English speech recognition engine available [MS01a].

To access the agent technology with Object Rexx the new `Orexxole.dll` from the 13.1.2003 is needed [Fla03b].

### 11.1. Introduction to MS Agent Technology

Code 20 shows the instantiation of an agent, shows a short animation and lets the agent speak. Therefore, the text-to-speech engine US English is needed.

---

<sup>188</sup> Characters Merlin and Peedy which are part of the MS Agent technology

<sup>189</sup> <http://microsoft.com/products/msagent/downloads.htm>

```

-----
-- Agent_Intro.rex --
-----

-- Instantiation of the agent object
AgentObject = .OLEObject~New("Agent.Control.1")
-- Connects the current control to the Microsoft Agent server190.
AgentObject~Connected = .True
-- The character is loaded into the Characters collection. The
-- first parameter is the characterID. It is required and refers
-- to the character data. The second parameter is the provider
-- with the location of the character's definition file191.
AgentObject~Characters~Load("Merlin", "Merlin.acs")
Merlin = AgentObject~Characters("Merlin") -- Decreases the writing expense
-- The Top property sets the agent "300"192 pixel from the top.
Merlin~Top = 300
-- The Left property sets the agent "450" pixel from the left193.
Merlin~Left = 450
-- The LanguageID property determines the language of the speech
-- recognition engine, the commands of character's pop-up menu and
-- the word balloon text. In this case the language is US English194.
Merlin~LanguageID = x2d(409)
-- The Show method makes the agent visible. It starts also the Showing
Merlin~Show -- animation195.
CALL SysSleep 1 -- The system sleeps for 1 second
-- The Play method implements the animation "GetAttention"196.
Merlin~Play("GetAttention")
-- The Speak method speaks the text inside the brackets197. The reason
-- for"objStatus ="is explained with the procedure CkStatus at the end.
objStatus = Merlin~Speak("Hello, my name is Merlin. I am an agent!")
-- Calls the procedure "CkStatus"
CALL CkStatus
-- The Play method implements the animation "Greet".
objStatus = Merlin~Play("Greet")
CALL CkStatus -- Calls the procedure "CkStatus"
CALL SysSleep 2 --The system sleeps for 2 second
-- The memory is freed from the agent "Merlin"198.

```

<sup>190</sup> [MLAGTb]

<sup>191</sup> [MLAGTc]

<sup>192</sup> [MLAGTd]

<sup>193</sup> [MLAGTe]

<sup>194</sup> [MLAGTf]

<sup>195</sup> [MLAGTg]

<sup>196</sup> [MLAGTh]

<sup>197</sup> [MLAGTi]

<sup>198</sup> [MLAGTj]

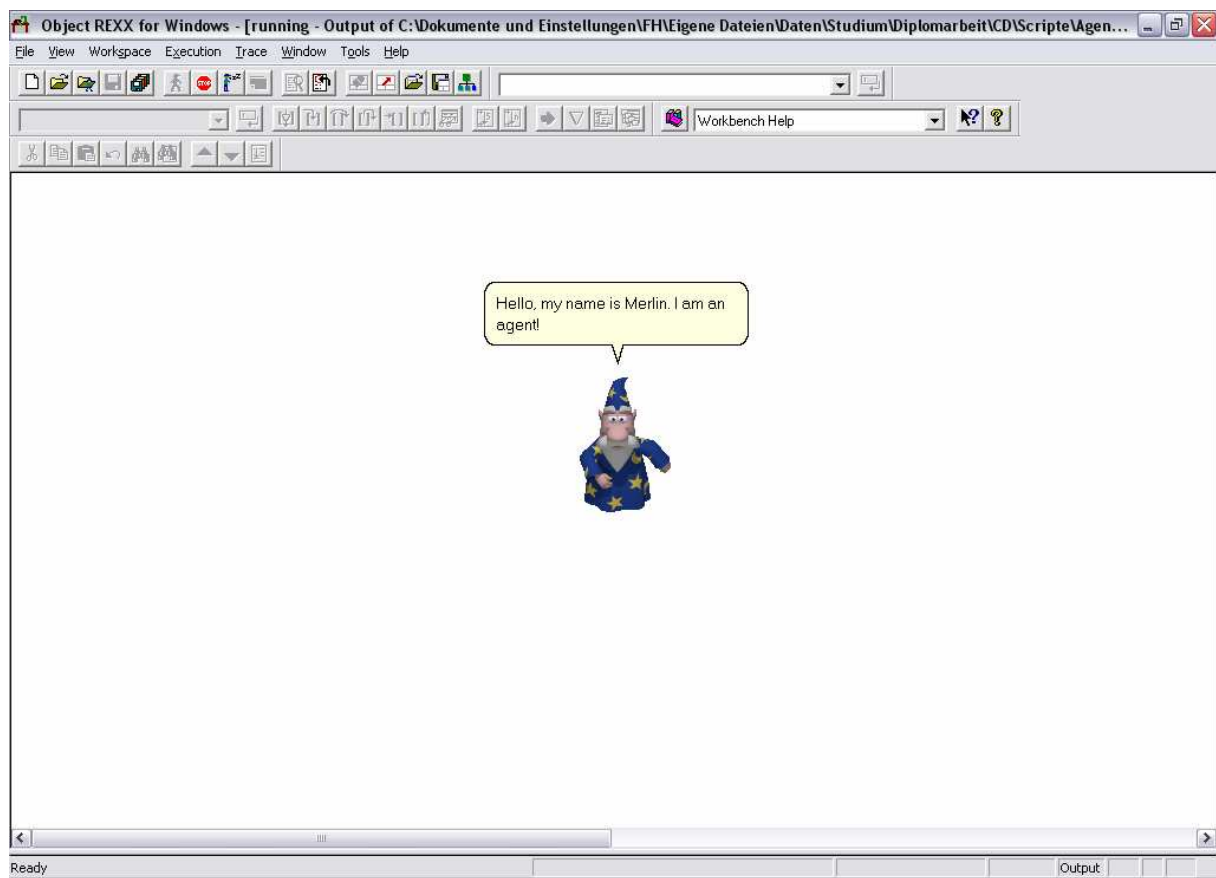
```

AgentObject~Characters~Unload("Merlin")
EXIT -- The program terminates.
      -- The procedure "CkStatus" is needed to prevent that the program
      -- continues to execute while the agent is speaking or playing.
      -- SysSleep makes a short delay199.
CkStatus:
  DO UNTIL objStatus~Status = 0
  CALL SysSleep 1
  END
RETURN

```

**Code 20:**      **Agent\_Intro.rex**

Figure 35 illustrates the execution of code 20.



**Figure 35:**      **Snapshot of Merlin<sup>200</sup>**

## 11.2. Overview of MS Agent Technology

Code 21 gives an overview of the agent's possibilities like animations, methods, speech engines, speech recognition or some windows.

<sup>199</sup> [Pe03]

<sup>200</sup> Part of MS Agent Technology

For this file the US English, German and French TTS engines are required. The agents Merlin, Peedy, Robby and Genie are used in this file. It is necessary to download them from the MS agent download page<sup>201</sup>. Probably Merlin is already installed on your machine. There is a .wav file (audio file) used for a short demonstration. This is the Tada.wav file and it is normally installed with Windows. It is located in the path \windows\media. Ensure that this location is correct. At the end of this script speech recognition is demonstrated. Therefore a microphone is needed and the speech recognition engine must be downloaded. There are ten seconds to speak to the computer. Nevertheless, this file should also run without error if there is no microphone or if speech recognition is not installed or deactivated.

This script uses the methods Speak, Play, Load, Show, Hide, MoveTo, UnLoad and Think. Properties, which are used, are the Connected, Top, Height, Left, GUID and Speed property. The script tells how to get the HotKey for speech recognition and demonstrates the windows with the properties of the characters, the popup menu and how to make there an entry, the MS Agent property sheet and the voice command window. Note that if you are working with other windows on your machine during the runtime of this script, it is possible that some windows described in the last part of the script will not occur.

```
-----
-- Agent_Overview.rex --
-----

-- Instantiation of the Agent object
AgentObject = .OLEObject~New("Agent.Control.1")
-- Connects the current control to the Microsoft Agent server202.
AgentObject~Connected = .True
-- The character is loaded into the Characters collection. The
-- first parameter is the characterID. It is required and refers
-- to the character data. The second parameter is the provider
-- with the location of the character's definition file203.
AgentObject~Characters~Load("Merlin", "Merlin.acs")
Merlin = AgentObject~Characters("Merlin") -- Decreases the writing expense
-- The Top property sets the top edge of the position of the agent
Merlin~Top = 250 -- to "250"204.
```

<sup>201</sup> <http://microsoft.com/products/msagent/downloads.htm>

<sup>202</sup> [MLAGTb]

<sup>203</sup> [MLAGTc]



```

-- The Left property sets the left edge of the position of the agent
Merlin~Left = 700 -- to "700"205.
-- The LanguageID property determines the language of the speech
-- recognition engine, the commands of character's pop-up menu and
-- the word balloon text. In this case the language is US English206.
Merlin~LanguageID = x2d(409)
-- The Show method makes the agent visible. It starts also the Showing
Merlin~Show -- animation207.
CALL SysSleep 1 -- The system sleeps for one second
-- The Speak method speaks the text inside the brackets208.
-- Sends the agent, the action and the argument to the routine Act.
CALL Act Merlin, "Speak", "Hello, I am Merlin and I am an agent!"
-- The Play method implements the animation "Explain"209.
CALL Act Merlin, "Play", "Explain"
-- Content refers to210
CALL Act Merlin, "Speak", "I want to tell you something about ",
"agents. Agents can be controlled with Object Rexx thanks to ActiveX."
CALL Act Merlin, "Play", "Suggest"
CALL Act Merlin, "Speak", "Firstly the Agent object must be created. After",
"that the Connected property is set on true."
CALL Act Merlin, "Play", "Blink"
CALL Act Merlin, "Speak", "Then the agent with its character is loaded. ",
"Let us try that. I will conjure up another agent. He will be ",
"positioned at the coordinates 250 and 250.",
" I set his LanguageID on US English and then I will show him."
CALL Act Merlin, "Play", "DoMagic1"
-- The agent Genie is loaded, positioned and got its language and is
AgentObject~Characters~Load("Genie", "Genie.acs") -- shown.
Genie = AgentObject~Characters("Genie")
Genie~Top = 250
Genie~Left = 250
Genie~LanguageID = "&H0409" -- US English language ID
Genie~Show
CALL Act Genie, "Play", "GetAttention"
CALL Act Genie, "Speak", "Hallo I am Genie! I can make you look like a ",
" dwarf with the Height property!"
CALL Act Genie, "Play", "GestureLeft"
Merlin~Height = 50 -- The agent Merlin is set on height "50" 211

```

<sup>204</sup> [MLAGTd]

<sup>205</sup> [MLAGTe]

<sup>206</sup> [MLAGTf]

<sup>207</sup> [MLAGTg]

<sup>208</sup> [MLAGTi]

<sup>209</sup> [MLAGTh]

<sup>210</sup> [To99]

<sup>211</sup> [MLAGTI]

---

```

CALL Act Merlin, "Speak", "Do not make me angry!"
CALL Act Genie, "Play", "GestureUp"
Merlin~Height = 128
CALL Act Merlin, "Play", "LookRight"
CALL Act Merlin, "Speak", "That is enough! It was a mistake to fetch you."
    -- The agent Merlin is moved to the position 350, 100212.
CALL Act Merlin, "MoveTo", 350,100
CALL Act Merlin, "Speak", "I will invoke the Hide method!"
CALL Act Merlin, "Play", "DoMagic2"
CALL SysSleep 1 -- The system sleeps for "1" second
Genie~Hide -- The Hide method hides the agent "Genie"213.
    -- The agent Peedy is loaded, positioned, got its language and is
AgentObject~Characters~Load("Peedy", "Peedy.acs") -- shown.
Peedy = AgentObject~Characters("Peedy")
Peedy~Top = 100
Peedy~Left = 750
Peedy~LanguageID = "&H0407" -- German language ID
Peedy~Show
CALL Act Peedy, "Play", "GetAttention"
CALL Act Merlin, "Play", "LookLeft"
    -- The text is written in thought balloon214.
CALL Act Merlin, "Think", "Such an ugly crow!"
CALL Act Peedy, "Speak", "Hallo ich bin Peedy." -- Peedy speaks German
CALL Act Merlin, "Speak", "I can not understand you. ",
    "What did you say? Come on change your LanguageID!"
CALL Act Peedy, "Play", "Sad"
CALL Act Peedy, "Speak", "Ich versteh den nicht. Ich probiere ",
    "es einfach mit einer neuen TTS Maschine."
Peedy~LanguageID = "&H040C" -- Changing to the French language ID
    -- Peedy is speaking French
CALL Act Peedy, "Speak", "Salut. Je suis Peedy. Est-ce que tu me comprend?"
CALL Act Merlin, "Play", "LookLeft"
CALL Act Merlin, "Play", "LookRight"
CALL Act Merlin, "Play", "LookLeft"
CALL Act Merlin, "Play", "LookRight"
CALL Act Merlin, "Play", "LookLeft"
CALL Act Peedy, "Play", "Pleased"
Peedy~LanguageID = x2d(409) -- Changing to US English language ID
    -- Now Peedy is speaking English
CALL Act Peedy, "Speak", "Okay that is the last language I had ",
    "downloaded from the MS Agent download page. I hope it works."
CALL Act Merlin, "Play", "LookLeft"
CALL Act Merlin, "Speak", "Fine! What do you know about us agents?"
CALL Act Peedy, "Play", "Explain"

```

---

<sup>212</sup> [MLAGTm]

<sup>213</sup> [MLAGTn]

<sup>214</sup> [MLAGTo]

```

-- Content refers to215
CALL Act Peedy, "Speak", "Okay. We can be used for conversational ",
    "interfaces for Web pages and applications. We are interactive ",
    "and we can make animations. We can speak via a text-to-speech ",
    "engine and recorded audio."
-- Content refers to216
CALL Act Peedy, "Speak", "We are able to accept voice commands ",
    "which are spoken. We are a further user interface."
CALL Act Merlin, "Speak", "That is right. Instead of speaking we ",
    "can use a .wav file like... "
-- It is also possible to output .wav files with the Speak method217
CALL Act Merlin, "Speak", "", "\windows\media\tada.wav"
CALL Act Peedy, "Play", "Idle1_2"
CALL Act Peedy, "Speak", "It is also possible to position your PopupMenu ",
    "with the Hide function at the coordinates 500 and 200."
-- Shows the popup menu at the position 500,200218
Merlin~ShowPopupMenu(500,500)
CALL Act Peedy, "Play", "LookRight"
CALL Act Merlin, "Speak", "Oh, I would have it almost forgotten. Before ",
    "I have hidden Genie! Thank you for remembering. Do you know how to ",
    "show him after hiding him?"
CALL Act Peedy, "Play", "Idle2_2"
CALL Act Peedy, "Speak", "No."
CALL Act Merlin, "Play", "GestureLeft"
CALL Act Merlin, "Speak", "That is not good.Have you seen Robby?He knows ",
    "it probably."
CALL Act Peedy, "Play", "Search"
CALL Act Peedy, "Speak", "No."
-- The agent Robby is loaded, positioned, got its language and is
AgentObject~Characters~Load("Robby", "Robby.acs") -- shown.
Robby = AgentObject~Characters("Robby")
Robby~Top = 100
Robby~Left = 50
Robby~LanguageID = x2d(409) -- US English ID
Robby~Show
CALL Act Robby, "Play", "GetAttention"
CALL Act Robby, "Speak", "Ho, ho. I heard my name! I am Robby."
-- Shows with Speed property the speech output219
CALL Act Robby, "Speak", "My speech speed is "AgentObject~Characters("Robby")~Speed"."
CALL Act Robby, "Speak", "That is not bad for a robot."
-- Shows the unique identifier of Robby220.

```

<sup>215</sup> [MLAGTa]

<sup>216</sup> [MLAGTa]

<sup>217</sup> [MLAGTi]

<sup>218</sup> [MLAGTp]

<sup>219</sup> [MLAGTq]

---

```

CALL Act Robby, "Speak", "My GUID is " AgentObject~Characters("Robby")~GUID"."
CALL Act Robby, "Speak", "What is your problem?"
CALL Act Robby, "Play", "GestureLeft"
CALL Act Peedy, "Speak", "Do you know how to show an agent?"
CALL Act Peedy, "Play", "Idle1_1"
CALL Act Robby, "Play", "Explain"
CALL Act Robby, "Speak", "Use the Show method!"
CALL Act Robby, "Play", "Suggest"
Genie~Show
    -- Content refers to221
CALL Act Genie, "Speak", "I want you to note that for XP, it is ",
    "required to download SAPI 4.0 runtime binaries from the MS Agent",
    " download page to make speech engines work!"
CALL Act Robby, "Play", "Idle3_1"
CALL Act Robby, "Speak", "That is enough work for today. But before I",
    " will change my font."
Robby~Balloon~FontName = Andy -- Changes the font type to "Andy"222
CALL Act Robby, "Speak", "I will go home. Bye."
CALL Act Robby, "Play", "Idle2_2"
CALL Act Genie, "Speak", "Do not forget. Free the space from you. ",
    "Therefore you have to use the UnLoad method"
AgentObject~Characters~Unload("Robby") -- Robby is freed from the memory223
CALL Act Genie, "Play", "Idle3_1"
CALL Act Genie, "Speak", "I am going voluntarily. Bye."
AgentObject~Characters~Unload("Genie")
CALL Act Merlin, "Speak", "Okay. Let us talk about the Command ",
    "collections. It is possible to add a Command object with the ",
    "Add method. I will add a command which is named TestCommand."
    -- The Add method of the Commands object adds a new command with
    -- its name as ID, the caption with "T" as shortcut determined by
    -- the"&"string and voice string to be recognized by a speech engine.224
Merlin~Commands~Add("TestCommand", "&Test", "Test")
    -- The PopupMenu is shown. The SYSSLEEP call and the second
    -- ShowPopupMenu command are needed because there is still
    -- a PopupMenu (Hide) on the display and otherwise it doesn't work.
Merlin~ShowPopupMenu(500,300)
CALL SysSleep 2
Merlin~ShowPopupMenu(500,300)
    -- Shows the caption with the Caption property225
CALL Act Merlin, "Speak", "Here the caption is "Merlin~Commands("TestCommand")~Caption

```

---

<sup>220</sup> [MLAGTr]

<sup>221</sup> [MS01a]

<sup>222</sup> [MLAGTs]

<sup>223</sup> [MLAGTt]

<sup>224</sup> [MLAGTu]

<sup>225</sup> [MLAGTv]

```

CALL SysSleep 1
    -- Shows the voice string with the Voice property226
CALL Act Merlin, "Speak", "The voice value is " Merlin~Commands("TestCommand")~Voice
CALL Act Merlin, "Speak", "This voice value enables to access this command",
    " with speech recognition"
    -- Shows the MSAgent Property Sheet window with the Visible property227.
AgentObject~PropertySheet~Visible = .true
CALL Act Peedy, "Play", "GestureRight"
CALL Act Peedy, "Speak", "The MS Agent Property Sheet window shows ",
    "current settings of output, speech input and copyright. You can",
    " also change these settings, like for example if speech ",
    "recognition is enabled or not, with this instrument."
CALL SysSleep 3 -- The system sleeps for "3" seconds to watch the window.
    -- The MS Agent Property Sheet window is hidden.
AgentObject~PropertySheet~Visible = .false
    -- Shows the Commands Window with setting the Visible property
AgentObject~CommandsWindow~Visible = .true -- on true228.
CALL Act Peedy, "Play", "LookDown"
CALL Act Peedy, "Speak", "Another window is the Voice Commands Window.",
    " It shows the commands which are voice-enabled for speech recognition."
CALL SysSleep 3
    -- Closes the Commands Window by setting the Visible property on false.
AgentObject~CommandsWindow~Visible = .false
CALL Act Merlin, "Play", "LookLeft"
CALL Act Merlin, "Speak", "Oh, Peedy, that is a good heading. Speech ",
    "Recognition. To use Speech Recognition you have to download from the",
    " MS Agent download page the U.S. English speech recognition engine."
CALL Act Peedy, "Play", "Acknowledge"
    -- Checks if on the machine in speech input is enabled. If this is the
    -- case then the first DO function is implemented229
IF AgentObject~SpeechInput~Enabled = 1 THEN
    -- Beginning of the DO function which is implemented if speech input is
    DO -- available on the machine.
        CALL Act Merlin, "Speak", "On your machine is speech recognition enabled."
        -- Offers the shortcut with the HotKey property with that speech input
        -- can be accessed.
        CALL Act Merlin, "Speak", "The HotKey is " AgentObject~SpeechInput~HotKey "."
        CALL Act Merlin, "Speak", "Press this HotKey to execute speech recognition"
        CALL Act Peedy, "Play", "StartListening"
        CALL Act Merlin, "Play", "StartListening"
        CALL Act Merlin, "Speak", "Okay let us make a test. Wait until I ended ",
            "speaking. Take your microphone and speak therein the word *hide*. ",
            "This will make Peedy disappear. Press the HotKey " ,

```

<sup>226</sup> [MLAGTw]

<sup>227</sup> [MLAGTx]

<sup>228</sup> [MLAGTy]

<sup>229</sup> [MLAGTz]

```

AgentObject~SpeechInput~HotKey " and then a small message box comes.",
" During this box displayed you can speak. This box should be ",
"referenced to Peedy. If not, first say *Peedy* and then *hide*.",
" Come on."
-- The system sleeps for "10" seconds so that the user has time to
CALL SysSleep 10 -- implement the speech recognition
END -- End of the DO function
ELSE DO -- If speech recognition is not enabled this DO function is used.
CALL Act Merlin, "Speak", "On your machine is speech recognition NOT enabled."
CALL Act Peedy, "Play", "Surprised"
CALL Act Peedy, "Play", "Idle3_1"
CALL Act Peedy, "Speak", "Bye"
CALL Act Peedy, "MoveTo", 50,500
Peedy~Hide
END
CALL Act Merlin, "Speak", "At last I want to show you a window which",
" shows the properties of each character."
-- Shows a window with that the default properties of the agents are
AgentObject~ShowDefaultCharacterProperties -- described230.
CALL Act Merlin, "Play", "Surprised"
CALL sysSleep 3
CALL Act Merlin, "Play", "Idle3_1"
CALL Act Merlin, "Speak", "That is all we wanted to tell you. Bye."
AgentObject~Characters~Unload("Merlin")
AgentObject~Characters~Unload("Peedy")
EXIT -- Terminates the program
-- send agent object (arg #1) the message given as arg #2; if there are
-- additional arguments, pass them as an array object (function arg())
::ROUTINE Act -- has to create an array-object starting with arg #3)
objStatus= .message~new(arg(1), arg(2), "A", arg(3,"Array"))~send
DO UNTIL objStatus~status=0
CALL SysSleep 1
END
RETURN

```

**Code 21:** Agent\_Overview.rex

Figure 36 illustrates a scene of Agent\_Overview.rex with Peedy, Merlin and Genie.

<sup>230</sup> [MLAGTaa]

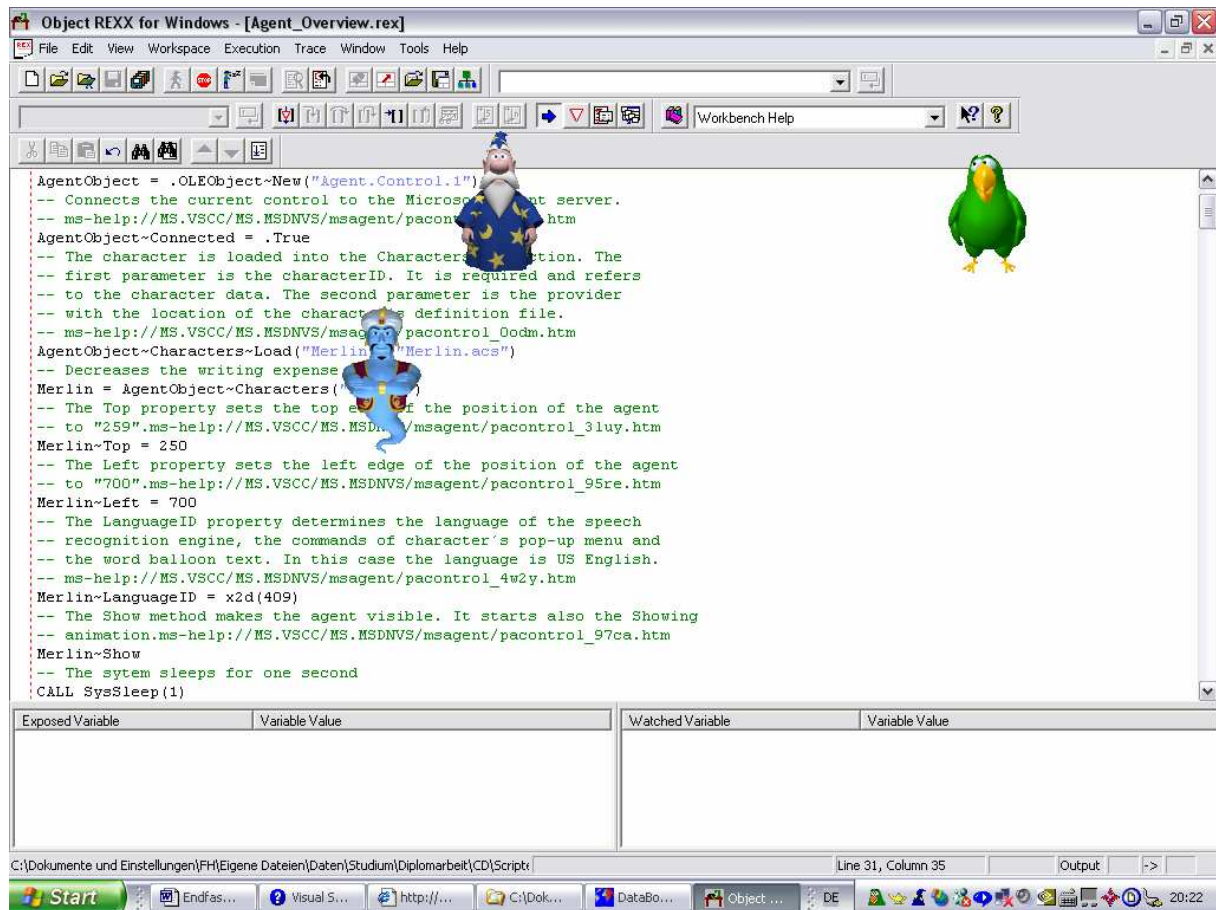


Figure 36: Snapshot of Agent\_Overview.rex<sup>231</sup>

## 11.3. MS Agent and Events

Code 22 demonstrates how to use events with the MS Agent technology.

In this script an event is fired if the agent is shown with the `Show` method. Then the `Show` method of the class `DemoOfEvents`, which is a subclass of `OLEObject`, is launched<sup>232</sup>. There the arguments are hand over and their values are printed to the display. To run this script the new `OREXXOLE.DLL` from the 17.2.2003 is needed. Without this DLL events of MS Agent can't be handled. This is because in some situations the events are hand over to Object Rexx in another kind as described in the MS Agent documentation [Doe03a]. The DLL is changed in two steps.

<sup>231</sup> IBM Object Rexx Workbench with MS Agents characters

<sup>232</sup> c.p. 6.3.3.

First rename the old OREXXOLE.DLL with the following command: `ren c:\Programme\ObjREXX\orexxole.dll orexxole.dll.bkp` and then copy the new DLL to the folder `c:\programme\ObjREXX [Fla03c]`.

```
-----
-- Agent_Events.rex --
-----

-- Instantiation of the agent object
AgentObject = .DemoOfEvents~New("Agent.Control.2", "WITHEVENTS")
-- Connects the current control to the Microsoft Agent server233.
AgentObject~Connected = .True
-- The character is loaded into the Characters collection. The
-- first parameter is the characterID. It is required and refers
-- to the character data. The second parameter is the provider
-- with the location of the character's definition file234.
AgentObject~Characters~Load("Merlin", "Merlin.acs")
Merlin = AgentObject~Characters("Merlin") -- Decreases the writing expense
-- The Top property sets the agent "250" pixel from the top235.
Merlin~Top = 250
-- The Left property sets the agent "250" pixel from the left236.
Merlin~Left = 250
-- The LanguageID property determines the language of the speech
-- recognition engine, the commands of character's pop-up menu and
-- the word balloon text. In this case the language is US English237.
Merlin~LanguageID = x2d(409)
-- The Show method makes the agent visible. It starts also the Showing
-- animation238. Here the Show event is fired
Merlin~Show
-- The Speak method speaks the text inside the brackets239.
-- The reason for "objStatus =" is explained with the procedure
objStatus = Merlin~Speak("Hello I am Merlin.") -- CkStatus at the end.
CALL CkStatus -- Calls the procedure "CkStatus"
EXIT -- The program terminates.
-- The procedure "CkStatus" is needed to prevent that the program
-- continues to execute while the agent is speaking or playing.
-- Sysssleep makes a short delay240.
```

---

<sup>233</sup> [MLAGTb]

<sup>234</sup> [MLAGTc]

<sup>235</sup> [MLAGTd]

<sup>236</sup> [MLAGTe]

<sup>237</sup> [MLAGTf]

<sup>238</sup> [MLAGTg]

<sup>239</sup> [MLAGTi]

<sup>240</sup> [Pe03]



```
CkStatus:
    DO UNTIL objStatus~Status = 0
        CALL SysSleep 1
    END
RETURN
    -- Class which derives from OLEObject
::CLASS DemoOfEvents SUBCLASS OLEObject
    -- Method which is called if the event Show is fired when the
::METHOD Show -- agent occurs on the display241.
    -- Hands over the arguments CharacterID and Cause
USE ARG CharacterID, Cause
    -- Prints a text and the content of the argument Cause to the display.
    -- Cause = 4 means that the client application showed the agent.
SAY "The cause is: " Cause
    -- Prints a text and the content of the argument CharacterID to the
SAY "The CharacterID is: "CharacterID -- display
SAY "There was an Show event." -- Prints a text to the display
```

**Code 22:**      **Agent\_Events.rex**

---

<sup>241</sup> [MLAGTk]

## 12. Automation of Microsoft Speech

The Microsoft Speech SDK 5.1 is a new user interface. It lets the computer speak and enables speech recognition. This section discusses the text to speech synthesis with an introduction to TTS (text-to-speech), an Object Rexx script with the speech technology is embedded in a HTML file and then there is a script that reads a MS Word document. The possibility to use speech recognition with Object Rexx is also explained.

The Microsoft Speech SDK (software development kit) contains the Microsoft speech synthesis (or text-to-speech) engine, the Microsoft speech recognition engine and the speech application programming interface (SAPI) which supports Automation. It can be downloaded from the Microsoft Homepage<sup>242</sup>.

It includes the documentation, which can also be single downloaded from the same page. The menu "Start->Programs->Microsoft Speech SDK 5.1" offers the documentation, tutorials, samples and tools [SPa]. Figure 37 shows this documentation.

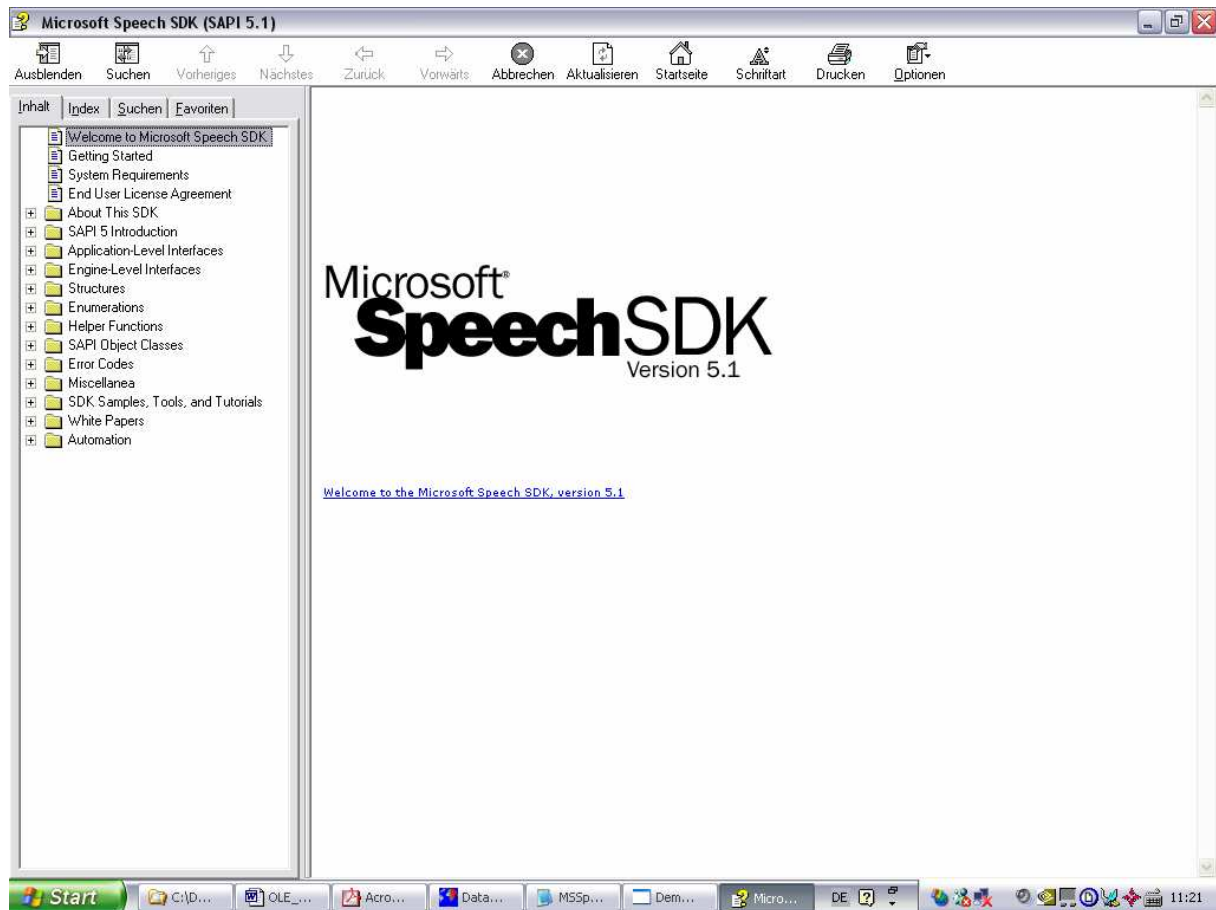
There is a newsgroup for Microsoft Speech where questions are answered<sup>243</sup>.

---

<sup>242</sup> <http://www.microsoft.com/speech/download/sdk51/>

<sup>243</sup>

[http://communities.microsoft.com/newsgroups/messageList.asp?ICP=MSCOM&sLCID=US&NewsGroup=microsoft.public.speech\\_tech&iPageNumber=1](http://communities.microsoft.com/newsgroups/messageList.asp?ICP=MSCOM&sLCID=US&NewsGroup=microsoft.public.speech_tech&iPageNumber=1)



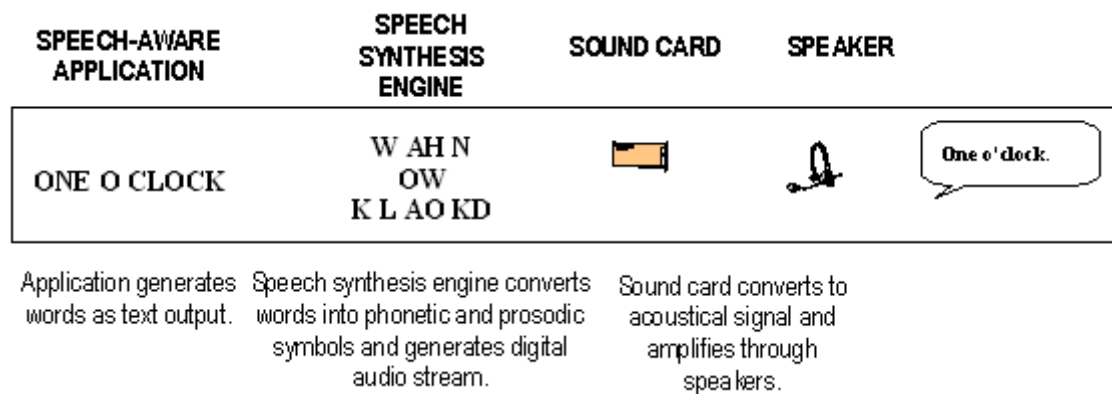
**Figure 37:** Snapshot of the MS Speech SDK 5.1 Help<sup>244</sup>

## 12.1. Text-To-Speech Synthesis (TTS)

Text-To-Speech synthesis is a further output possibility which signification will increase in the future. This section introduces MS Speech TTS with the technical background. It is also demonstrated how to embed MS Speech TTS in HTML and how to read a MS Word document with Object Rexx.

Speech Synthesis converts text into spoken language. As shown in the figure 38 the words are converted to phonemes. Thereby an audio stream is created which is transformed by the sound card and emitted by a speaker [MS02].

<sup>244</sup> Part of MS Speech SDK 5.1



**Figure 38: Speech recognition process flow<sup>245</sup>**

The TTS technology is only available from Microsoft in US English and Chinese. Third party vendors for further TTS machines can be found on the Microsoft Homepage<sup>246</sup> [MS02].

The registry entry `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens` shows which voices are installed on the machine [SPb].

L&H TTS engine for the MS Agent Technology can't be used with MS Speech SDK 5.1 because they are based on SAPI4 [NGS03].

### 12.1.1. Introduction to MS Speech TTS

Code 23 demonstrates Microsoft Speech synthesis. Thereby the four voices Sample TTS Voice (says "blah" for all unknown words [MS03]), Microsoft Mary, Microsoft Mike and Microsoft Sam are used. The changing of the volume and of the speech speed is shown. At the end, all available voices on the machine are enumerated.

<sup>245</sup> Taken from [MS02]

<sup>246</sup> <http://www.microsoft.com/speech/evaluation/thirdparty/engines.asp>

```

-----
-- MSSpeech_TTS_1.rex --
-----
SpO=.OLEObject~New("SAPI.SpVoice") -- Instantiation of a TTS object247
    -- The voice is set with the Voice property248. Therefore the GetVoices
    -- method249.GetVoices chooses the voice by the variable voicex.
SpO~Voice = SpO~GetVoices("name=SampleTTSVoice", "")[0]
    -- The Speak method speaks the inserted text250.
SpO~Speak("Sample TTS Voice, hello")
SpO~Voice = SpO~GetVoices("name=Microsoft Mike","", "[0]
SpO~Speak("Hello. I am the voice Mike")
    -- The voice is set with the Voice property. Therefore the GetVoices
    -- method chooses the voice by the variable voicex and the language
    -- (here English)251252.
SpO~Voice = SpO~GetVoices("name=Microsoft Sam","Language=409")[0]
SpO~Speak("Hello.I am the voice Sam!")
SpO~Voice = SpO~GetVoices("name=Microsoft Mary","Language=409")[0]
SpO~Speak("Hello. I am the voice Mary!")
SpO~Volume = 80 -- The volume is set with the Volume property253
SpO~Speak("The volume is set on " SpO~Volume)
SpO~Volume = 100
SpO~Speak("Now the volume is set on 100. That is the maximum volume.")
SpO~Rate = -10 -- The speech speed is set with the Rate property254
SpO~Speak("This is the slowest speaking rate. The rate is " SpO~Rate)
SpO~Rate = 10
SpO~Speak("This is the fastest speaking rate. The rate is " SpO~Rate)
SpO~Rate = -2
SpO~Speak("This is the speaking rate " SpO~Rate)
SpO~Speak("Now the available voices are enumerated.")

```

<sup>247</sup> Looked up with RGF\_OLEInfo.hta c.p. 7.3.

<sup>248</sup> [SPc]

<sup>249</sup> [SPc]

<sup>250</sup> [SPc]

<sup>251</sup> [SPd]

<sup>252</sup> [SPb]

<sup>253</sup> [SPc]

<sup>254</sup> [SPc]

```

DO a OVER SpO~GetVoices -- DO function which enumerates all voices
    -- Offers the name of a voice and hands it over to the variable
    -- "strvoice"255
strvoice = a~Getdescription
SpO~Speak("The voice " strvoice " is available!" )
SAY strvoice
END -- End of DO function

```

**Code 23:** MSSpeech\_TTS\_1.rex

## 12.1.2. MS Speech TTS embedded in HTML

The next file embeds an Object Rexx script in a HTML document. This HTML document is stored with an .HTA (HTML Application) extension. This ensures a local and save execution<sup>256</sup> [Fla02a,p51].

Code 24 speaks a text which can be inserted by the user. The user can vary the volume, the speech speed (rate) and the voice. Press the *Speak* button to start the speech.

```

<html>
    <!-- Beginning of the head -->
<head>
    <!-- Text of the title -->
<title>Demonstration of Speech with Object Rexx</title>
    <!-- Beginning of the Object Rexx source code -->
<script language="Object Rexx">
    -----
    -- MSSpeech_TTS_2.hta --
    -----

    -- Beginning of the routine doTheWork. It is referenced from the
    -- body if the button is pressed or clicked."Public" enables the
    -- transferring of the data from the input area in the body to the
::routine doTheWork public -- routine
    -- Transfer of data to the variables "speechtext", "volume" and "rate".
    -- Thereby refers the document object to the content which is shown
    -- in a browser window. "All" is an object which enables the access
    -- to single elements and content of HTML documents. This is part of
    -- DHTML. "speechtext", "volume" and "rate" are the identifiers.
    -- "Value" is the content of the identifier.
speechtext = document~all~speechtext~value
volume = document~all~volume~value
rate = document~all~rate~value

```

<sup>255</sup> [SPd]

<sup>256</sup> c.p. 9.1.3.

```

-- The radiobuttons are evaluated with a SELECT function. The button
SELECT -- which is selected hands over its voice to the variable vx.
  WHEN document~all~MicrosoftMike~checked = 1 THEN vx = "Microsoft Mike"
  WHEN document~all~MicrosoftMary~checked = 1 THEN vx = "Microsoft Mary"
  WHEN document~all~MicrosoftSam~checked = 1 THEN vx = "Microsoft Sam"
END
SpO=.OLEObject~new("SAPI.SpVoice") -- Instantiation of a TTS object257
SpO~Volume = volume -- The volume is set with the Volume property258
SpO~Rate = rate -- The speech speed is set with the Rate property259
  -- The voice is set with the Voice property. Therefore the GetVoices
  -- method chooses the voice by the variable vx and the language260 261
SpO~Voice = SpO~GetVoices("name=vx,"Language=409")[0] -- (here English)
  -- The Speak method speaks the inserted text which is hand over by
SpO~Speak(speechtext) -- the speechtext variable262.
  -- End of the Object Rexx code
</script>
  <!-- End of head -->
</head>
  <!-- Beginning of the body with the backgroundcolor "gold" -->
<body bgcolor="gold">
  <!-- All after that tag is centred -->
  <center>
    <!-- A text is written with the font size "7" -->
    <font size=7>Demonstration of Microsoft Speech with Object Rexx</font>
    <!-- End of center tag -->
  </center>
    <!-- Two line breaks -->
  <br>
  <br>
    <!-- Beginning of marquee text -->
  <marquee>
    <!-- Text with the font color "red", font size "5" and font type
    "Andy" is written -->
    <font color="red" size=5 face="Andy">Object Rexx can speak!!
  </font>
  </marquee>
  <br>
  <br>
  <br>
  <br>

```

<sup>257</sup> Looked up with RGF\_OLEInfo.hta c.p. 7.3.

<sup>258</sup> [SPc]

<sup>259</sup> [SPc]

<sup>260</sup> [SPd]

<sup>261</sup> [SPb]

<sup>262</sup> [SPc]

```

<center>
    <!-- A text is written -->
    <x>Text which is to be spoken:</x>
    <br>
    <!-- Input area of the type text with the internal name "SpeechText".
         The length of the area which is visible is "100".The internal
         length is "150".To reference to this object the id "speechtext"
         is used -->
    <input type=text name="SpeechText" size=100 maxlength=150
        id="speechtext">
    <br>
    <br>
    <br>
    <br>
    <br>
    <x>Volume ( a value between 0 and 100 ) :</x>
    <br>
    <br>
    <!-- Input area of the type text with the internal name "Volume". The
         length of the area which is visible is "3".The internal length
         is "3". To reference to this object the id "volume" is used -->
    <input name="Volume" size=3 maxlength=3 id="volume">
    <br>
    <br>
    <br>
    <br>
    <br>
    <x>Rate ( a value between -10 and 10 ) : </x>
    <br>
    <br>
    <!-- Input area of the type text with the internal name "Rate". The
         length of the area which is visible is "3".The internal length
         is "3". To reference to this object the id "rate" is used -->
    <input name="Rate" size=3 maxlength=3 id="rate">
    <br>
    <br>
    <br>
    <br>
    <br>
    <x>Voices:</x>
    <br>
    <br>
    <!-- Input area of the type radio button with the internal name
         "voicek". Value determines the internal value "MicrosoftMike"
         of the radio button. The identifier is "MicrosoftMike". There
         is also the text "Microsoft Mike" is written. -->
    <input type="radio" name="voiceK" value="MicrosoftMike"
        id="MicrosoftMike"> Microsoft Mike<br>
    <br>

```



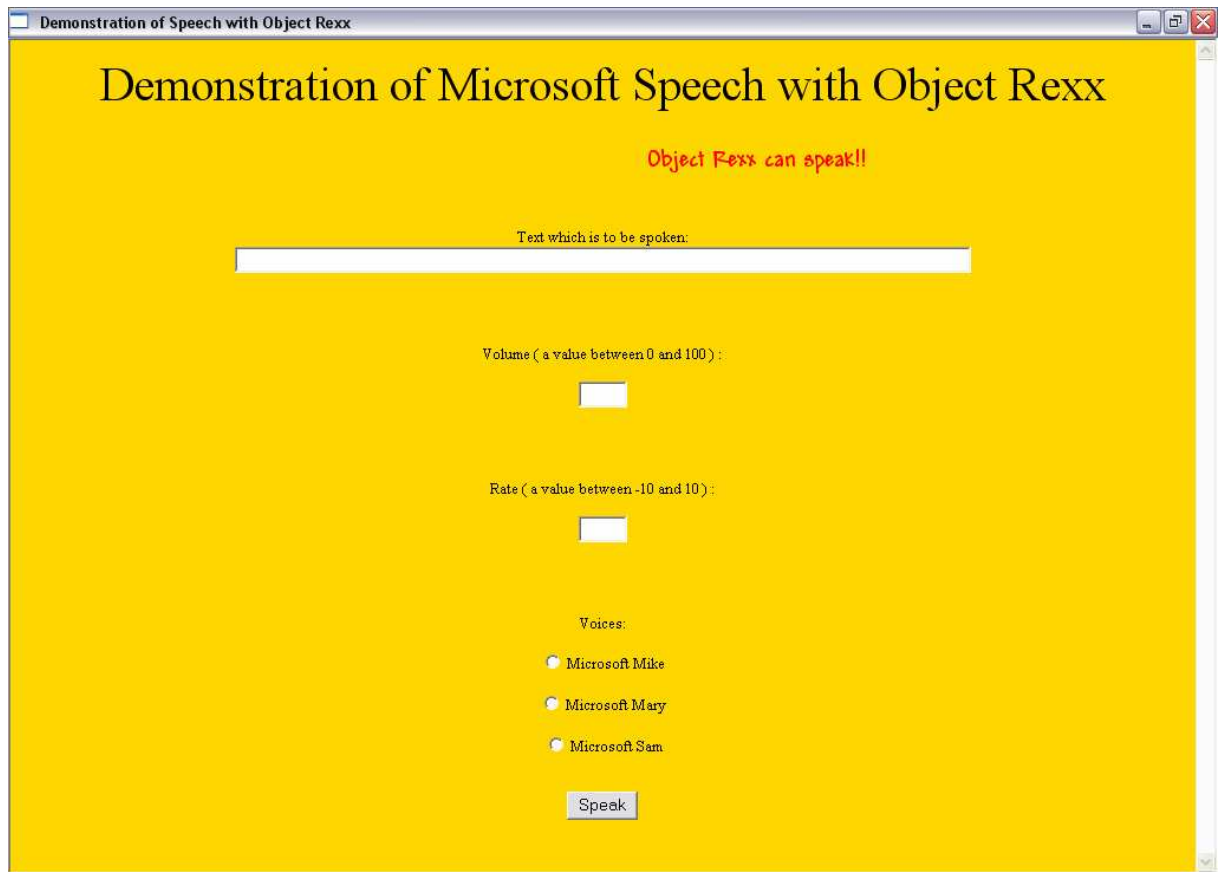
```

        <!-- Input area of the type radio button with the same internal name
        "voicek". This means that the radio buttons are members of the
        same group. Value determines the internal value "MicrosoftMary"
        of the radio button. The identifier is "MicrosoftMary".There is
        also the text "Microsoft Mary" is written. -->
<input type="radio" name="VoiceK" value="MicrosoftMary"
      id="MicrosoftMary"> Microsoft Mary<br>
<br>
        <!-- Input area of the type radio button with the same internal name
        "voicek". This means that the radio buttons are members of the
        same group. Value determines the internal value "MicrosoftSam"
        of the radio button. The identifier is "MicrosoftSam". There is
        also the text "Microsoft Sam" is written. -->
<input type="radio" name="VoiceK" value="MicrosoftSam"
      id="MicrosoftSam"> Microsoft Sam
<br>
<br>
<br>
        <!-- Input area of the type button.The text on the button is"Speak".
        The phrase "language="Object Rexx" embeds Object Rexx.If one of
        the two event handler"onmouseup"or "onkeypress" is executed the
        routine doTheWork is called -->
<input type=button value="Speak"
      language="Object Rexx"
      onmouseup="call doTheWork"
      onkeypress="call doTheWork">
</center>
<!-- End of the body -->
</body>
<!-- End of the HTML file -->
</html>

```

**Code 24:** MSSpeech\_TTS\_2.hta

Figure 39 illustrates the user-interface of MSSpeech\_TTS\_2.hta.



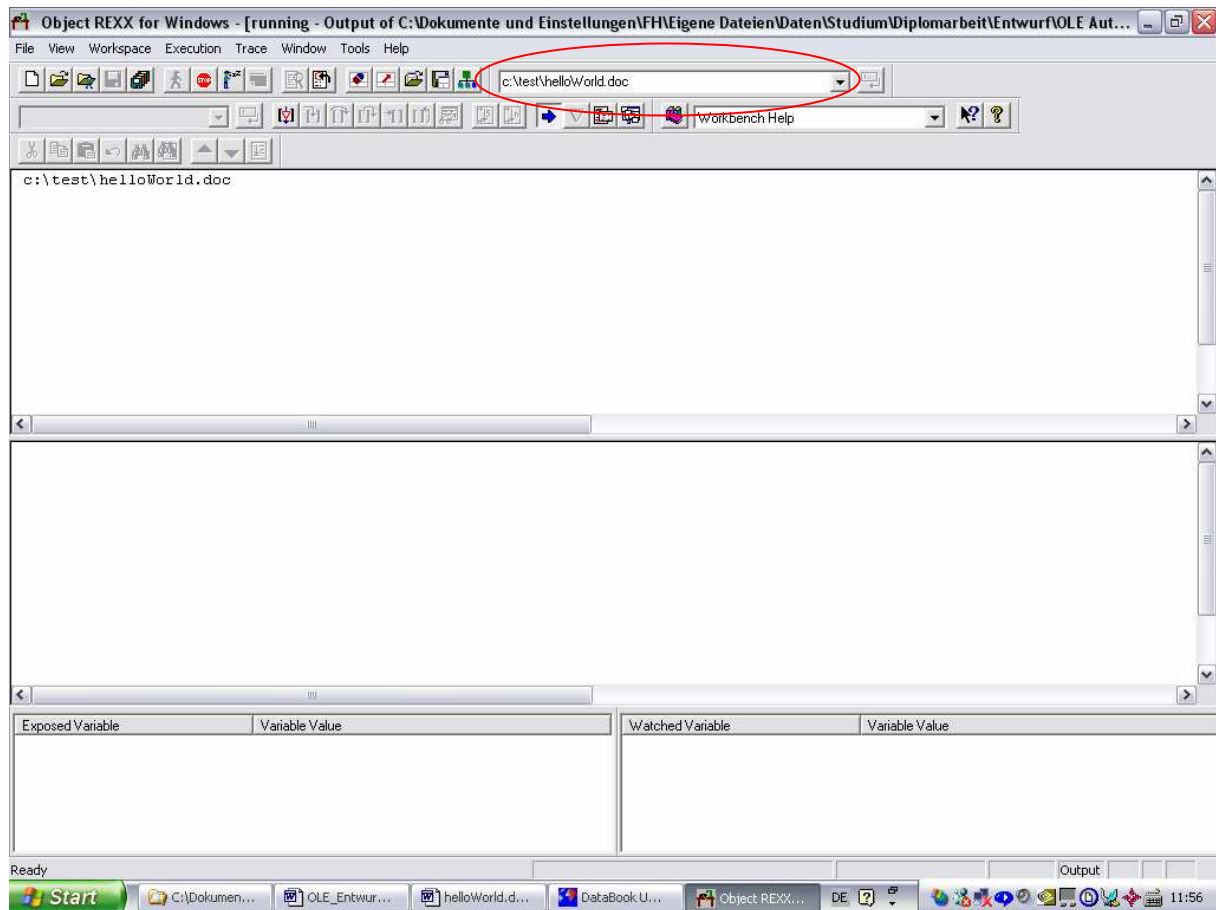
**Figure 39:** Snapshot of MSSpeech\_TTS\_2.hta<sup>263</sup>

### 12.1.3. Reading a MS Word Document

Code 25 reads a Word document. The user is asked to insert the location and the name of a Word document that should be read to the Object Rexx Workbench command line as shown in figure 40.

---

<sup>263</sup> Part of MS Windows XP



**Figure 40:** Snapshot of the IBM Object Rexx Workbench with command line.

After that, the Word document is opened and its content is copied to the clipboard.

This part is recorded by the macro recorder of Microsoft Word to get information about the required source code (OtherScript 7). The MS Word document "HelloWorld.doc" is loaded, the whole document is selected and copied to the clipboard. The MS Word document should contain an English text.

```

Sub Makrol()
'
' Makrol Makro
' Makro aufgezeichnet am 05.02.2003 von
'
    ChangeFileOpenDirectory "C:\Test\"
    Documents.Open FileName:="HelloWorld.doc", ConfirmConversions:=False, _
        ReadOnly:=False, AddToRecentFiles:=False, PasswordDocument:="", _
        PasswordTemplate:="", Revert:=False, WritePasswordDocument:="", _
        WritePasswordTemplate:="", Format:=wdOpenFormatAuto
    Selection.WholeStory
    Selection.Copy
End Sub

```

**OtherScript 7: Macro for MSSpeech\_TTS\_3\_Word.rex.**

Additionally in code 25 is the content of the clipboard pasted by the Object Rexx class WindowsClipboard to the Microsoft Speech object SpVoice. Therefore the directive `::REQUIRES "WINSYSTEM.CLS"` is needed<sup>264</sup>. Then the text is spoken. The script also works if there is a graphic included in the selection.

```

-----
-- MSSpeech_TTS_3_Word.rex --
-----

    -- Message box with the invitation to insert the path and file name.
CALL RxMessageBox "Insert the file name with the path", "Information", ,
    "OK", "ASTERISK"

    -- Hand over of the path and file name in the command window to the
PARSE PULL docu -- variable "docu"
Word = .OLEObject~New("Word.Application") -- Instantiation of Word
Word~Visible = .TRUE -- The visibility of Word is set on true
Document = Word~Documents~Open(docu) -- The document is opened
Word~Selection~WholeStory -- All of the Word document is selected
    -- Here the selected text is copied to the clipboard. The Copy method
    -- cannot be used in the normal way because this Copy method wouldn't be
    -- sent to Word but to the Object Rexx class. The UNKNOWN method solves
Word~Selection~Unknown("Copy", .nil) -- this problem265.
cb = .WindowsClipboard~New -- A clipboard object is created266
    -- The Paste method hands over the content of the clipboard267
wordtext = cb~Paste

```

<sup>264</sup> c.p. 8.3.3.

<sup>265</sup> c.p. 6.4.

<sup>266</sup> [IBM01c]

<sup>267</sup> [IBM01,p283]

```

SpO=.OLEObject~New("SAPI.SpVoice") -- Instantiation of a TTS object268
    -- The voice is set with the Voice property. Therefore the GetVoices
    -- method chooses the voice by the variable voicex and the language
    -- (here English)269270.
SpO~Voice = SpO~GetVoices("name=Microsoft Sam","Language=409")[0]
    -- The Speak method speaks the text of the Word document which is hand
    -- over by the variable271.
SpO~Speak(wordtext)
    -- Loads definition of the WindowsClipboard class
::REQUIRES "WINSYSTEM.CLS"

```

**Code 25: MSSpeech\_TTS\_3\_Word.rex**

## 12.2. Speech Recognition

Speech recognition is a further input possibility which signification will increase in the future. This section provides information about the technical background, the possibilities and problems with Object Rexx.

Speech recognition (speech-to-text) digitizes sound waves. These sound waves are transformed to phonemes or basic language units. From these phonemes are words constructed which are contextually analyzed to check the correct spelling of a word (right and write). Figure 41 shows this recognition process.

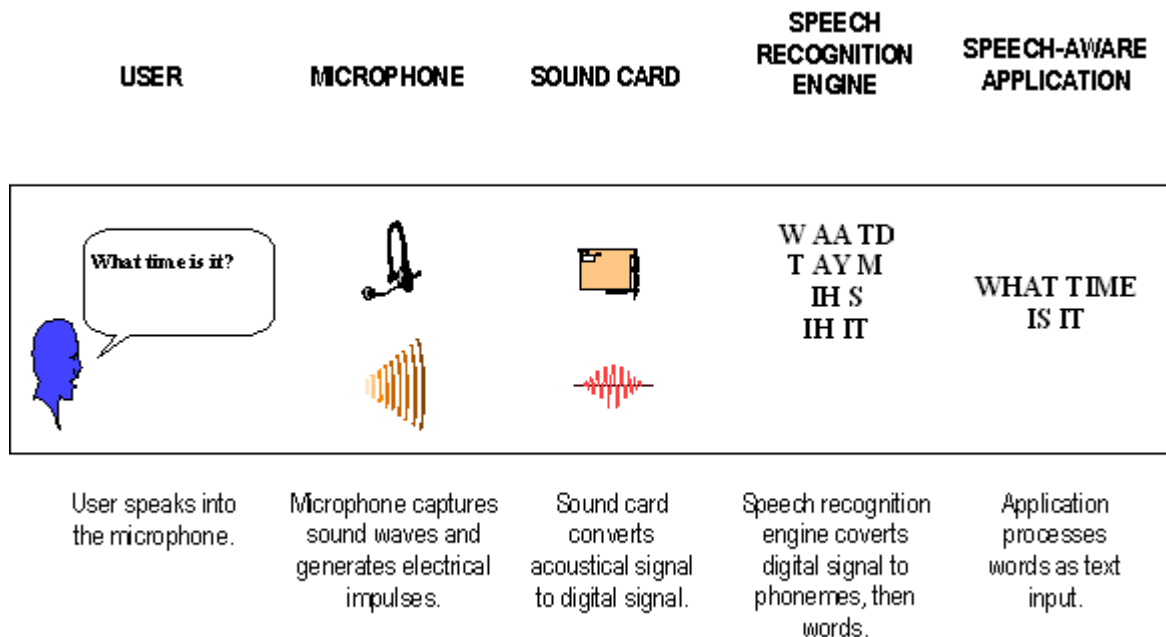
---

<sup>268</sup> Looked up with RGF\_OLEInfo.hta c.p. 7.3.

<sup>269</sup> [SPd]

<sup>270</sup> [SPb]

<sup>271</sup> [SPc]



**Figure 41:** Speech recognition process<sup>272</sup>

So-called speech recognition engines are software drivers. These software drivers transform the acoustical signal to a digital signal. Continuous speech means that the speaker can talk to the microphone without pause. There are two modes for continuous speech recognition engines. The one is *Command and Control* (speaking commands and asking questions) and the other is *Dictation* (enables the dictating of letters, memos or e-mail messages) [MS02].

### 12.2.1. Dictation Recognition

Code 26 is a *dictation recognition* example.

It should enable free dictation. If there is not spoken to the microphone after the invitation to speak then the program terminates. If there is spoken to the microphone after the invitation to speak, then the program runs so long it is spoken to the microphone. Only if speaking is stopped the program terminates. There is no error message. Nevertheless, there is no event fired. The problem is that Object Rexx does not support the events of MS Speech. The reason therefore is that the events of MS Speech are not hand over to Object Rexx via the generic interfaces `IDispatch`

<sup>272</sup> Taken from [MS02]

and IUnknown. The events are hand over via special ID's. These ID's are not queried with Object Rexx [Doe03c].

```
-----
-- MSSpeech_SR_1_Dictation.rex --
-----

    -- Instantiates the SR object. Here is a an object used which enables
    -- shared context to allow resources to be used by other recognition
    -- contexts or applications273. Events are enabled.
SO = .EventsWithSO~New("SAPI.SpSharedRecoContext", "WITHEVENTS")
    -- The CreateGrammar method creates an object based on
SGrammar = SO~CreateGrammar -- ISpeechRecoGrammar274
    -- The DictationSetState method sets the dictation topic state on
SGrammar~DictationSetState(1) -- active275
    -- Creates a message box with invitation to speak
CALL RxMessageBox "Speak", "Information", ,
    "OK", "ASTERISK"
    -- Class which derives from OLEObject276
::CLASS EventsWithSO SUBCLASS OLEObject
    -- Method which is called if the Recognition event277 is fired. This
    -- occurs when the speech recognition(SR)engine produces a recognition.
::METHOD Recognition
    USE arg Streamnumber, Streamposition, RecognitionType, Result
    say Result~PhraseInfo~GetText
    say "I have recognized s.th."
    -- The FalseRecognition event278 occurs when the speech recognition (SR)
    -- engine produces a false recognition.
::METHOD FalseRecognition
    USE arg Streamnumber, Streamposition, Result
```

---

<sup>273</sup> [SPe]

<sup>274</sup> [SPf]

<sup>275</sup> [SPg]

<sup>276</sup> c.p. 6.3.3.

<sup>277</sup> [SPh]

<sup>278</sup> [SPi]

```

SAY "No recognition"
    -- If the SR engine encounters the start of an audio input stream then
::METHOD StartStream -- the StartStream event is fired279.
    use arg Streamnumber, Streamposition
    SAY Streamnumber

```

**Code 26:** MSSpeech\_SR\_1\_Dictation.rex<sup>280</sup>

## 12.2.2. Command and Control Recognition

Code 27 and code 28 demonstrate the *Command and Control* (C&C) recognition.

### 12.2.2.1. C&C Recognition with Configuration File

Code 27 describes the Command and Control Recognition with a configuration file.

Therefore a second file is used which contains the grammar with the recognizable text. The file name is `solx.xml` and the text is “one”. If the program is started it runs without error. If it is spoken the word “one” or not there is no event fired. The problem is that Object Rexx does not support the events of MS Speech. The reason therefore is that the events of MS Speech are not hand over to Object Rexx via the generic interfaces `IDispatch` and `IUnknown`. The events are hand over via special ID’s. These ID’s are not queried with Object Rexx [Doe03c].

```

-----
-- MSSpeech_SR_2_CommandAndControl.rex --
-----

    -- Instantiates the SR object. Here is a an object used which enables
    -- shared context to allow resources to be used by other recognition
    -- contexts or applications281. Events are enabled.
SO = .EventsWithSO~New( "SAPI.SpSharedRecoContext", "WITHEVENTS" )
    -- The CreateGrammar method creates an object based on
SGrammar = SO~CreateGrammar -- ISpeechRecoGrammar282
    -- The CmdLoadFromFile method283 loads a command and control grammar
    -- from the file "solx.xml"."1" means that the grammar is loaded
    -- dynamically, meaning that rules can be modified and committed at run

```

<sup>279</sup> [SPj]

<sup>280</sup> Modelled after [SPk]

<sup>281</sup> [SPe]

<sup>282</sup> [SPf]

<sup>283</sup> [SPI]



```

SGrammar~CmdLoadFromFile("c:\Test\solx.xml",1) -- time284.
    -- The CmdSetRuleIdState method285 activates or deactivates a rule by
SGrammar~CmdSetRuleIdState(0, 1) -- its rule ID
    -- Message box with the invitation to speak.
CALL RxMessageBox "Speak", "Information",,,
    "OK", "ASTERISK"
    -- Class which derives from OLEObject286
::CLASS EventsWithSO SUBCLASS OLEObject
    -- Method which is called if an event is fired. The Recognition event
    -- occurs when the speech recognition (SR)engine produces a
::METHOD Recognition -- recognition287.
    use arg Streamnumber, Streamposition, RecognitionType, Result
    Result~PhraseInfo~GetText
    say "I have recognized s.th."
    -- The FalseRecognition event occurs when the speech recognition (SR)
::METHOD FalseRecognition -- engine produces a false recognition288.
    use arg Streamnumber, Streamposition, Result
    SAY "No recognition"
    -- If the SR engine encounters the start of an audio input stream then
::METHOD StartStream -- the StartStream event is fired289
    use arg Streamnumber, Streamposition
    SAY Streamnumber

```

**Code 27: MSSpeech\_SR\_2\_CommandAndControl.rex<sup>290</sup>**

OtherScript 8 contains the file solx.xml with the grammar with the recognizable text. This file is taken from [SPo]. Instead of “new +game” is “one” used.

<sup>284</sup> [SPm]

<sup>285</sup> [SPn]

<sup>286</sup> c.p. 6.3.3.

<sup>287</sup> [SPh]

<sup>288</sup> [SPi]

<sup>289</sup> [SPj]

<sup>290</sup> Modeled after [SPo]

```

<GRAMMAR LANGID="409">
  <DEFINE>
    <ID NAME="RID_NewGame" VAL="101"/>
  </DEFINE>

  <RULE NAME="newgame" ID="RID_NewGame" TOPLEVEL="ACTIVE">
    <P> new +game </P>
  </RULE>
</GRAMMAR>

```

*OtherScript 8: solx.xml*<sup>291</sup>

### 12.2.2.2. C&C and the Creation of a new Grammar Rule

Code 28 was modeled after an example from Inigo Surguy that was programmed with the language Python<sup>292</sup>. It should create a new grammar rule and fire the Recognition event if the word “Hello” is said. This example doesn’t work correctly. There comes always the following error message:

```
28 *-* SGrammar~Rules~Commit
```

```
Error 92 running C:\...\SR_1.rex line 28: OLE error
```

```
Error 92.906: OLE exception: Code: 80045062 Source:
unavailable Description: unavailable
```

```

-----
-- MSSpeech_SR_3_CommandAndControl_2.rex --
-----

-- Instantiates the SR object. Here is a an object used which enables
-- shared context to allow resources to be used by other recognition
-- contexts or applications293. Events are enabled.
SpeechObject = .EventsWithSO~new("SAPI.SpSharedRecognizer","WITHEVENTS")
-- The CreateRecoContext method294 creates a recognition context object
-- from the recognizer.
SContext = SpeechObject~CreateRecoContext
-- The CreateGrammar method295 creates an object based on
SGrammar = SContext~CreateGrammar -- ISpeechRecoGrammar
--The DictationSetState method sets the dictation topic state296.

```

<sup>291</sup> Taken from [SPo]

<sup>292</sup> <http://www.surguy.net/articles/speechrecognition.xml>

<sup>293</sup> [SPe]

<sup>294</sup> [SPp]

<sup>295</sup> [SPf]

---

```

SGrammar~DictationSetState(0) -- Here inactive297
    -- The Add method298 creates a new ISpeechGrammarRule object in an
    -- ISpeechGrammarRules collection.
    -- "wordsRule" is the rule name, "33" means "SRATopLevel = 1" +
    -- "SRADynamic = 32" and "0" is the rule ID299.
WordsRule = SGrammar~Rules~Add("wordsRule",33,0)
    -- The Clear method300 clears a rule, leaving only its initial state.
WordsRule~Clear
    -- The AddWordTransition method301 adds a word transition from this rule
    -- state to another rule state in the same rule. The AddState method302
    -- adds a state to a speech rule303.
WordsRule~Initialstate~AddWordTransition(WordsRule~AddState, "Hello")
    -- The Commit method compiles the rules in the rule collection304.
SGrammar~Rules~Commit
    -- The CmdSetRuleState method305 activates or deactivates a rule by
    -- its name. Here it is activated306.
SGrammar~CmdSetRuleState("wordsRule",1)
    -- The Commit method compiles the rules in the rule collection.
SGrammar~Rules~Commit
    -- Creates a message box with invitation to speak
CALL RxMessageBox "Speak", "Information", ,
    "OK", "ASTERISK"
    -- Class which derives from OLEObject307
::CLASS EventsWithSO SUBCLASS OLEObject

```

---

<sup>296</sup> [SPg]

<sup>297</sup> [SPq]

<sup>298</sup> [SPr]

<sup>299</sup> [SPs]

<sup>300</sup> [SPt]

<sup>301</sup> [SPu]

<sup>302</sup> [SPv]

<sup>303</sup> [Doe03d]

<sup>304</sup> [SPw]

<sup>305</sup> [SPx]

<sup>306</sup> [Spy]

<sup>307</sup> c.p. 6.3.3.

```
-- Method which is called if an event is fired.  
-- The Recognition event occurs when the speech recognition  
::METHOD Recognition -- (SR) engine produces a recognition308.  
USE arg Streamnumber, Streamposition, RecognitionType, Result  
Result~PhraseInfo~GetText  
SAY "I have recognized s.th."
```

**Code 28:**      **MSSpeech\_SR\_3\_CommandAndControl\_2.rex**

---

<sup>308</sup> [SPh]

## 13. Windows Script Host

A disadvantage of further Windows versions was that they were not able to be used for automation tasks. There were only MS-DOS Batch files available [Ge03]. This was changed with the advent of Windows Script Host (WSH) [Mo98]. WSH enables the interaction among ActiveX components, the access of the registry, the launching of applications or the communication with the operating system. It is possible to generate installation scripts for applications or to automate tasks for the user [Ge03]. Windows Script Host is a Windows administration tool and has the two components host and engine. For the interaction of these two parts are used the interfaces `IPersist*`, `IActiveScript`, `IActiveScriptSite`, `IDispatch` and `IconnectionPoint`.

This section discusses scripting, Object Rexx and Windows Script Host, Windows Script Host and Windows Script Engine, types of script files, kinds of running a script, the instantiating of objects, the WSH object model, the `FileSystemObject` object, the `Dictionary` object, security in Windows Script Host, starting applications with WSH and Windows Script Components.

Windows Script Host (WSH) is language-independent. It is possible to run scripts from the command prompt and from the Windows desktop. WSH is the host for a script. That means that it makes services and objects available for the script [MLWSHa]. There are three WSH applications known at the point of time this paper was written: Internet Information Services (IIS) whereat IIS is a WSH host and then WSH engines can be used for scripting of ASP [Fla03e], MS Internet Explorer and the WSH Shell which is used as Windows interface [Fla03].

With scripting, variables can be set and stored, and it can be worked with data in HTML code [MLWSHc].

WSH makes objects from applications available in the runtime environment of the scripting language [Fla02c,p8].

WSH objects and services enable [MLWSHa]:

- Connection to printers,

- changing of registry keys,
- changing and retrieving of environment variables,
- fundamental functions as `CreateObject` and `GetObject`,
- printing of messages to the screen,
- mapping network drives and
- remote control of Windows applications.

WSH is included in for example Microsoft Windows 98, 2000 or XP. For Windows 95 the Windows Script Host 5.6 is downloadable from the Microsoft Homepage<sup>309</sup> [MLWSHa]. The Windows Script Host is included and updated with the MS Internet Explorer [Fla03].

A recommended source for information is the System Administration Scripting Guide. This guide can be downloaded from the Microsoft Homepage<sup>310</sup>.

## 13.1. Scripting

This chapter contains information about script basics and server-side scripting to improve the understanding.

### 13.1.1. Script Basics<sup>311</sup>

A script is programmed in a scripting language like VBScript, JScript, Python, DOS Batch language Perl or Object Rexx<sup>312</sup>.

Scripting facilitates work with data in HTML code to communicate with the user, check the browser or the input and to work with controls and applets [MLWSHc].

Computer systems on a network can be remotely administered.

---

<sup>309</sup> <http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169>

<sup>310</sup> <http://www.microsoft.com/downloads/release.asp?ReleaseID=38942>

<sup>311</sup> [MLWSHa]

<sup>312</sup> [Fla02d]

A Windows script can be written with an unpretentious text editor. It is only necessary to save the file with the appropriate extension [MLWSHd].

A script is normally stored in a file and it is used for remote-control and to automate applications. Thereby it is a sequence of repetitious commands pointed to shells or applications [Fla02d].

A script is suited for the following purposes:

- Starting other programs
- Key sequences are transmitted to an application
- Changing the Windows environment
- Logon procedures that are automatic
- Nonrepetitive tasks
- Sequence of tasks
- Response to an event

### **13.1.2. Server-Side Scripting<sup>313</sup>**

Server-side scripting allows the running of scripts on the server for example for data that can be stored in a database. Active Server Pages (ASP) is used for server-side scripting with generating an ASP file. There could be a mix of components like Java applets or ActiveX, scripting and HTML. ASP has the five standard objects `Application` (lifetime control and share application-level information), `Server` (Internet Information Server control), `Session` (settings and information about the user's present web-server session), `Request` (information from the user) and `Response` (information to the user). ASP events are supported by Object Rexx [IBM01,p505].

---

<sup>313</sup> [MLWSHe]

## 13.2. Object Rexx and Windows Script Host<sup>314</sup>

This section provides information about the support of Windows Script Host by Object Rexx. The COM interfaces which Object Rexx supports and basic items are discussed.

### 13.2.1. Basics

Object Rexx is a script engine for Windows Script Host.

Object Rexx does not support WSH Script Debugging, which enables to view the source code, to view and modify property and variable values, to check and view the script flow and to control the pace of the script execution [MS01b], and DCOM<sup>315</sup>.

### 13.2.2. COM Interfaces

Not all of the WSH engines interfaces are supported by Object Rexx. Some interfaces that are supported are generated dynamically. Not all methods of a supported interface are implemented although the code for all methods of that interface is present. Not implemented methods return E-NOTIMPL.

Interfaces with full support:

- IActiveScriptError
- IActiveScriptParse
- IActiveScriptParseProcedure
- IObjectSafety
- IUnknown

Interfaces that are also supported:

- IActiveScript

---

<sup>314</sup> This section uses [IBM01c,p152ff]

<sup>315</sup> c.p. 2.7.



- Close
- AddNamedItem
- AddTypeLib
- GetScriptDispatch
- GetScriptState
- SetScriptSite
- SetScriptState
- IDispatch
  - GetIDsOfNames
  - Invoke
- IDispatchEx
  - GetDispID (no support for dynamic generation of methods or properties)
  - GetMemberName
  - GetNextDispID
  - InvokeEx (no support for dynamic generation of methods or properties)

## 13.3. Host and Engine

This section describes script host and engine and their interaction. Script host and engine are the two kinds of script components.

### 13.3.1. Script Host and Script Engine Basics

With Windows Script Host VBScript, Jscript, Python, Perl and Rexx can be used as an engine. .vbs and .js files are already registered in Windows; others have to be registered before running a file [MLWSHf].

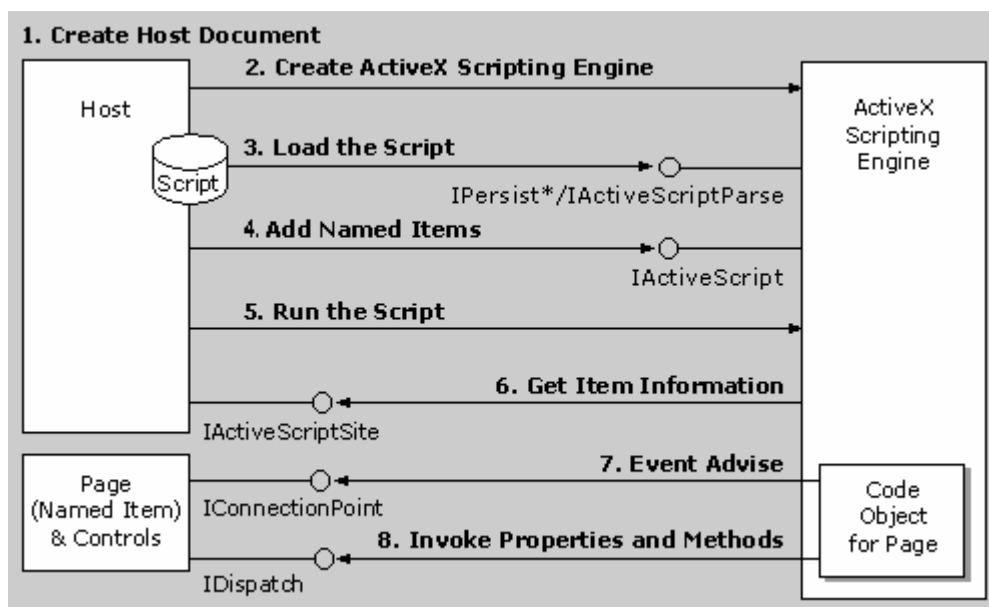
There are two kinds of script components. There is a script engine and a script host [IBM01,p493].

The script host generates the script engine that is an OLE object and implements the scripts. A Script host are for example the Shell, Internet authoring tools or the Microsoft Internet Explorer, CScript and WScript. The script host can add initialized objects to the runtime environment of the WSE and it is any application that uses one of the `IActiveScript` family interfaces [Fla02d,p9].

The Script engine runs with any run-time or language environment like Microsoft Visual Basic Scripting Edition, Lisp, Perl or Object Rexx that supports the COM interfaces `IActiveScript`, `IActiveScriptParse` and `IPersist` and OLE/ActiveX Automation. The WSE makes it possible that objects from applications are obtainable in the runtime environment [Fla02d,p3,p8].

### 13.3.2. Interaction between Scripting Host and Engine<sup>316</sup>

Script interfaces offer the possibility for an application to use OLE Automation and scripting skills. The interaction of host and engine is illustrated in figure 42.



**Figure 42:** Interaction between Scripting Host and Engine<sup>317</sup>

<sup>316</sup> [MLWSHg]

<sup>317</sup> Taken from [MLWSHg]

1. Creation of a document or project.
2. Creation of the Windows Script Engine with the `CoCreateInstance` method.
3. The script is loaded with an `IPersist*` interface.
4. Named Items (remarkable OLE COM object to the script) are added with the `IActiveScript::AddNamedItem` method.
5. The script is running.
6. Information about the item with the `IActiveScriptSite::GetItemInfo` method.
7. Connection of the scripting engine to the events with the `IConnectionPoint` interface.
8. Methods and properties are called with standard OLE binding mechanisms or `IDispatch::Invoke`.

## 13.4. Types of Script File<sup>318</sup>

In the sequent sections there are used several kinds of Windows Script Host files and several kinds of files which contain Windows Script Host objects. These types of Windows Script files are described in this section.

The first is the `.wsf` file, which is a container or project file, and the second is the `.wsh` file, which is a property file for a script file. Other types are `.vbs` (VBScript), `.js` (JScript), `.bat` (MS DOS batch file), ASP page, `.html` (HTML file) or `.rxs` and `.rex` (both Object Rexx).

### 13.4.1. WSF File Using Windows Script Files (.wsf)

This file format is used often in further code examples.

A WSF file (`.wsf`) is structured in XML. The `.wsf` format allows multiple-engine support to join several languages in a `.wsf` file and can be edited with any XML editor or any other editor. It is a project or container file. Functions from several languages can be inserted in the project. Type libraries are used to add constants to the code

---

<sup>318</sup> [MLWSHh]

and the code of several jobs can be saved in one file. A WSF file can be programmed with any script engine<sup>319</sup> [Fla02d,p10].

A script can be split up to several parts. As beginning, a `.wsf` file is generated and the other parts referenced by the `.wsf` file are for example Visual Basic Script or JScript files [MLWSHi].

### **13.4.2. WSH File**

A WSH file is automatically created by setting the properties for a script file. It is a text file. With a WSH file the implementation of one or more scripts is handled [MLWSHk].

### **13.4.3. REX File**

A REX file is a file with Object Rexx code. It is created with the Object Rexx Workbench and can contain script code.

### **13.4.4. RXS File**

There are several Object Rexx scripts, which only work with this extension.

A RXS File means ObjectRexxScriptFile. It contains an Object Rexx program that is started via Windows Script Host [Fla02e]. It is suitable for the usage of `wscript.exe`<sup>320</sup> or `cscript.exe`<sup>321</sup> [Fla03]. The `.rxs` files can be executed with a double-click in the Windows Explorer or or with the name of the file. Thereby WScript is launched and the script file is hand over as argument [Fla03e].

---

<sup>319</sup> c.p. 13.8.3.

<sup>320</sup> c.p. 13.5.2.

<sup>321</sup> c.p. 13.5.1.

The following two commands are inserted to the command window<sup>322</sup> [Fla03e]:

```
C:\Assoc.rxs
```

Output:

```
.rxs=ObjectRexxScriptFile
```

And:

```
C:\ftype ObjectRexxScriptFile
```

Output:

```
ObjectRexxScriptFile=%SystemRoot%\system32\WScript.exe "%1" %*
```

## 13.5. Running a Script

There are different possibilities how to run a script. This chapter describes these possibilities.

### 13.5.1. CScript<sup>323</sup>

With `Cscript.exe` scripts can be started from the command prompt. The command has the following syntax.

```
cscript scriptname.extension [script options and parameters]
```

The output of the script is sent to the command window.

### 13.5.2. WScript

Another possibility to run a script is from Windows. Therefore, `Wscript.exe` is used. It offers a Windows-based dialog box for the properties. Here the output is a windowed output [MLWSHm].

---

<sup>322</sup> Start->Run->Cmd

<sup>323</sup> [MLWSH!]

### 13.5.3. Embedding a Script in a HTML File<sup>324325</sup>

A further possibility of running scripts is to embed them in a HTML file with the Microsoft Internet Explorer. It is possible to interact in this way with any scripting engine [Fla02d]. This section was taken from [He02,p10]

There is a head and a body in the HTML file.

The head contains the title and the original source code of the Object Rexx script. Here are the operations that are running in the background. The body contains what can be seen on the monitor. There are the headlines, references and the input area for the data. HTML-files consist of ASCII-text. HTML-commands are written in “tags”. That means that they are marked with pointed brackets. There is always an opening tag and a closing tag. All between them is the range of validity of the tag.

Dynamic HTML (DHTML) makes it possible to change elements of a www-site dynamically. This site behaves like an application. To act in this way event-handlers are used. Knowledge in script languages like Object Rexx is needed to work with DHTML<sup>326</sup>.

### 13.5.4. Other Possibilities to Run a Script

Scripts can also be started with a double-click in the explorer.

## 13.6. Instantiating of Objects

Here is explained how to instantiate WSH objects with Object Rexx.

There are two possibilities to declare objects with Object Rexx. The one is the WSH method `WScript~CreateObject()` and the other is the Object Rexx method `.OLEObject~New()`, in which `New` generates a new instance of the object<sup>327</sup>. The first possibility offers the advantage that it supports the events of the object. On the other side, it is a COM object implementing a function that can be carried out

---

<sup>324</sup> c.p. 9.1.3.

<sup>325</sup> c.p. 13.7.4.1.

<sup>326</sup> Taken from [He02,p10]

<sup>327</sup> c.p. 6.3.

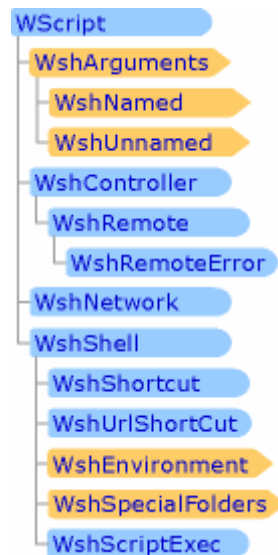
internally. If the `WScript` object is used outside WSH, like for example in a REX file, it does not work. Therefore, a REX file<sup>328</sup> is used. [IBM01,p512].

## 13.7. WSH Object Model<sup>329</sup>

This is the broadest section of the WSH chapter. It discusses the whole WSH object model. It explains for example how to execute a script on another machine in a network or how to send keystrokes to the active window.

Parts are the `WSHArguments` object, the `WSHController` object, the `WSHNetwork` object and the `WSHShell` object.

The WSH consists of 14 objects that are illustrated in Figure 43.



**Figure 43: WSH Object Model<sup>330</sup>**

- `WScript` is the root object of the object model and it is always available from every script file. It offers information about the host file name and host version, the name of the script file, command-line arguments and to the default output device. It enables the creation, connection and disconnection of objects, and sync events and stops the script's execution programmatically.

<sup>328</sup> c.p. 13.7.1.

<sup>329</sup> [MLWSHn]

<sup>330</sup> Taken from [MLWSHn]

- `WshArguments` allows accessing the entire set of command-line arguments. With `WshNamed` the set of named command-line arguments can be accessed. `WshUnnamed` allows accessing the set of unnamed command-line arguments.
- `WshController` allows with the method `CreateScript()`, to generate a remote script process. `WshRemote` allows the manipulation of other scripts and programs. `WshRemote Error` shows the error information in the case of a script error.
- `WshNetwork` maps or closes network shares, allows the access to the shared resources of a network, offers data of a user in the network and makes the connection and disconnection to network printers and shares.
- `WshShell` generates shortcuts, changes the environmental variables, starts a program locally, accesses the system folder and changes the contents of the registry. `WshShortcut` offers a programmatically creation of a shortcut. `WshURLShortcut` creates a shortcut to an Internet URL. `WshEnvironment` allows utilization of environmental variables like `PATH` or `PROMPT`. `WshSpecialfolders` allows accessing “Windows Special Folders” like the Start Menu folder or the Desktop folder. `WshScriptExec` offers error and status information about a script that runs with the `Exec` method of the `WshShell` object.

The COM interfaces used by the Windows Script Model are based on two categories:

- Helper Functions to perform actions with methods and properties.
- Script Execution and Troubleshooting to perform messages to the screen, essential COM functions or changing of the Windows Script Host.

### **13.7.1. WshArguments Object<sup>331</sup>**

The `WshArguments` object accesses the command-line arguments that are used.

---

<sup>331</sup> This section uses [MLWSHo]



This script has the extension .rxs. To run that script the MS-DOS Shell is used and the script runs with the command `cscript wsh_arg.rxs /W s h` or `wscript wsh_arg.rxs /W s h`. Note that the `WScript` object runs only if it is dynamically generated by `WScript` or `CScript` to pass the pointer to Object Rexx and it is not registered in the Windows registry [IBM01,p508]. The `Arguments` property returns the `WshArguments` object. The `Arguments` property needs the `WScript` object. The `Named` and the `Unnamed` properties of the `WshArguments` object return the `WshNamed` respectively the `WshUnnamed` object. `WScript~Echo` creates if the script is executed with `CScript` an output like the `SAY` command. If the script is started with `WScript` a pop-up box is generated [IBM01,p494].

Figure 44 illustrates the MS-DOS shell<sup>332</sup> where the file `WSH_Arg.rxs` with the arguments is executed.

```

C:\Test>cscript wsh_arg.rxs /W s h
Microsoft (R) Windows Script Host, Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. Alle Rechte vorbehalten.

There are 3 Arguments
There are 3 Arguments
There are 1 named arguments.
There are 2 unnamed arguments.
Windows Script Host
1
0
C:\Test>_

```

**Figure 44:** Snapshot of the MS-DOS Shell<sup>333</sup>

Code 29 contains the script `WSH_Arg.rxs`.

<sup>332</sup> Start->Run->Command

<sup>333</sup> Part of MS Windows XP

```

-----
-- WSH_Arg.rxs --
-----
    -- Input: cscript wsh_arg.rxs /W s h
    --      or wscript wsh_arg.rxs /W s h
    -- The number of Arguments in the command-line is counted with
    -- the count property
SAY "There are " wscript~Arguments~count " Arguments"
    -- Stops the execution of the script for 5000 milliseconds334
WScript~Sleep(5000)
    -- The number of Arguments in the command-line is counted with the
SAY "There are " wscript~Arguments~length " Arguments" -- length property335
    -- The number of named Arguments is counted with the length
    -- property
WScript~Echo("There are " WScript~Arguments~Named~length " named arguments.")
    -- The number of unnamed Arguments is counted with the Count
    -- property336
WScript~Echo("There are " WScript~Arguments~Unnamed~count " unnamed arguments.")
WScript~Echo(WScript~Name) -- Offers the name of the WScript object
    -- Exists method recognizes if a named argument exists337.1=true
WScript~Echo(WScript~Arguments~Named~Exists("W"))
    -- Unnamed Arguments are not recognized.
WScript~Echo(WScript~Arguments~Named~Exists("s"))

```

**Code 29: WSH\_Arg.rxs**

### 13.7.2. WshController<sup>338</sup>

This object has the method `CreateScript` that references the `WSHRemote` object. The `WSHController` object enables the instantiation of script on a remote machine [Es02]. It is explained how to use this object on the local machine, on multiple machines and the usage of events.

There are two scripts, the control script and the remote script. The control script instantiates a `WSHController` object and connects to the remote machine. The remote script is copied to the memory of the remote machine. The remote script is there executed [WSS02].

<sup>334</sup> [MLWSHp]

<sup>335</sup> [MLWSHq]

<sup>336</sup> [MLWSHr]

<sup>337</sup> [MLWSHs]

<sup>338</sup> [MLWSHt]

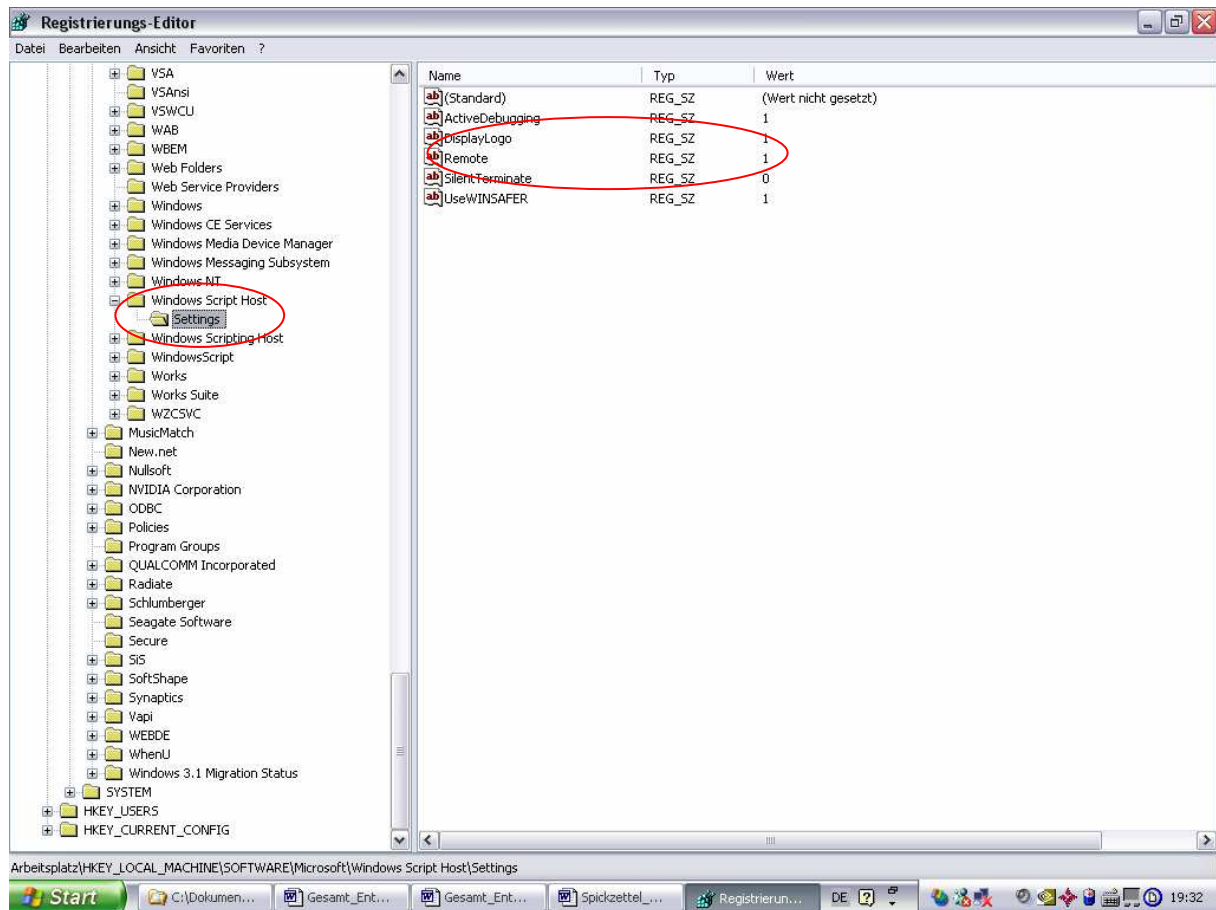
The method `CreateScript` has the following syntax:  
`CreateScript(CommandLine,[MachineName])`. The parameter `CommandLine` is required and contains the path of the control script seen from the controller machine. The parameter `MachineName` is optional and contains the name of the remote machine. If this parameter is left free, the remote script is executed on the controller machine [MLWSHu].

To use the remote functionality WSH 5.6 has to be installed<sup>339</sup> (for Windows XP or Internet Explorer 6 or greater WSH 5.6 is already installed), the remote and the local machine must have Windows NT 4 SP3 or higher and the registry has to be changed. Thereby for the key `HKEY_LOCAL_MACHINE \SOFTWARE \Microsoft \Windows Script Host \Settings` an entry must be added. Right-click that subkey in the Registry Editor<sup>340</sup>, click on New and choose String Value in the Edit menu. Insert for value name "Remote" and as data value „1“ [MS03k]. This key must be activated on that machine on which the remote script is executed. If the remote script is executed on a machine other as the source script this key needn't be set on "1" in the source system but it may be set on "1". It is not necessary to restart the system after modifying this key. Figure 45 shows in the small circle the key and in the big circle the new entry.

---

<sup>339</sup> <http://msdn.microsoft.com/scripting>

<sup>340</sup> c.p. 2.6.1.



**Figure 45:** Snapshot of the registry with regedit.exe<sup>341</sup>

For Windows XP machines it is probably necessary to use the command `C:\> wscript -regserver` in the menu Start->Run. This is why there is a configuration bug which prevents WSH 5.6 from setting up correctly [WSS02].

### 13.7.2.1. WSHController on the local Machine

This example shows the `WSHController` object with a remote script that is launched on the same machine. For this example, a MS Windows XP Home machine was used.

The first script code 30 is the control script.

<sup>341</sup> The Registry Editor is part of MS Windows XP c.p. 2.6.1.

```

-----
-- WshController_OnTheSameMachine.rex --
-----

-- Instantiation of the Controller Object
Controller = .OLEObject~New("WSHController")
-- The method CreateScript generates a WSHRemote object342
RemoteScript = Controller~CreateScript("C:\TestWSHCtr\WSHRemoteCalc.rxs")
-- The Status property tells the Status of the remote script
-- In this case the status is 0 which means that the remote script
SAY RemoteScript~Status -- object is generated but not executed343.
-- The Execute method begins the execution of the remote script
RemoteScript~Execute -- object344.
-- In this case the status is 1 which means that the remote script
SAY RemoteScript~Status -- object is running.
-- DO function which is executed until the value of the status is 2
DO UNTIL RemoteScript~Status = 2
    CALL SysSleep 1
END
-- In this case the status is 2 which means that the remote script
-- object is not still running.
SAY RemoteScript~Status

```

**Code 30:** WSHController\_OnTheSameMachine.rex<sup>345</sup>

Code 31 is the remote script WSHRemoteCalc.rxs that is launched by the controller script. It has the file extension .RXS because it must be a script file. Therein the calculator is launched with the Exec method<sup>346</sup>.

```

-----
-- WSHRemoteCalc.rxs --
-----

-- Instantiation of the Shell object
WshShell = .OLEObject~New("WScript.Shell")
WshShell~Exec("calc") -- Launches the Calculator

```

**Code 31:** WSHRemoteCalc.rxs

### 13.7.2.2. WSHController on multiple Machines

In the next example the WSHController object with the WSHRemote object is used on multiple machines. In this case two MS Windows XP Pro machines were used. It

<sup>342</sup> [MLWSHu]

<sup>343</sup> [MLWSHv]

<sup>344</sup> [MLWSHw]

<sup>345</sup> Modelled after [WSS02]

<sup>346</sup> c.p. 13.7.4.5.

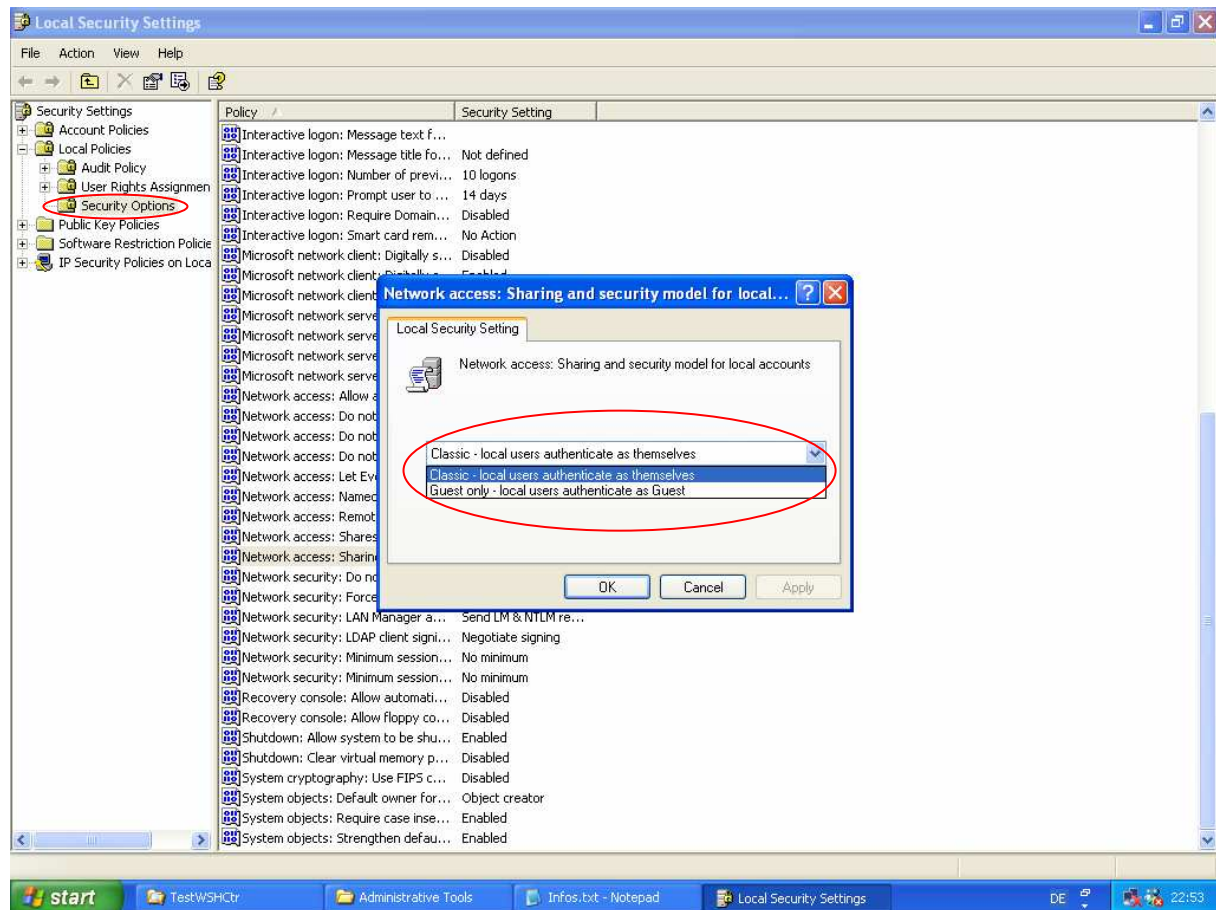
doesn't work with MS Windows XP Home machines or if one of the two machines is a MS Windows XP Home machine [DI03a].

Both the control script and the remote script are on the control machine.

There is a registry changing necessary on the target computer. By default a `ForceGuest` issue is set for Windows XP machines. `ForceGuest` means that all that is coming from the network is authenticated as `Guest` user. To change this, open the Local Security Policy console located in the folder Administrative Tools<sup>347</sup>. In the Security Policy console choose Security Settings\Local Policies\Security Options. Then look for Network Access: Sharing And Security Model For Local Accounts. Change the setting from `Guest` to `Classic` [DI03]. Figure 46 shows the setting in the Security Policy console.

---

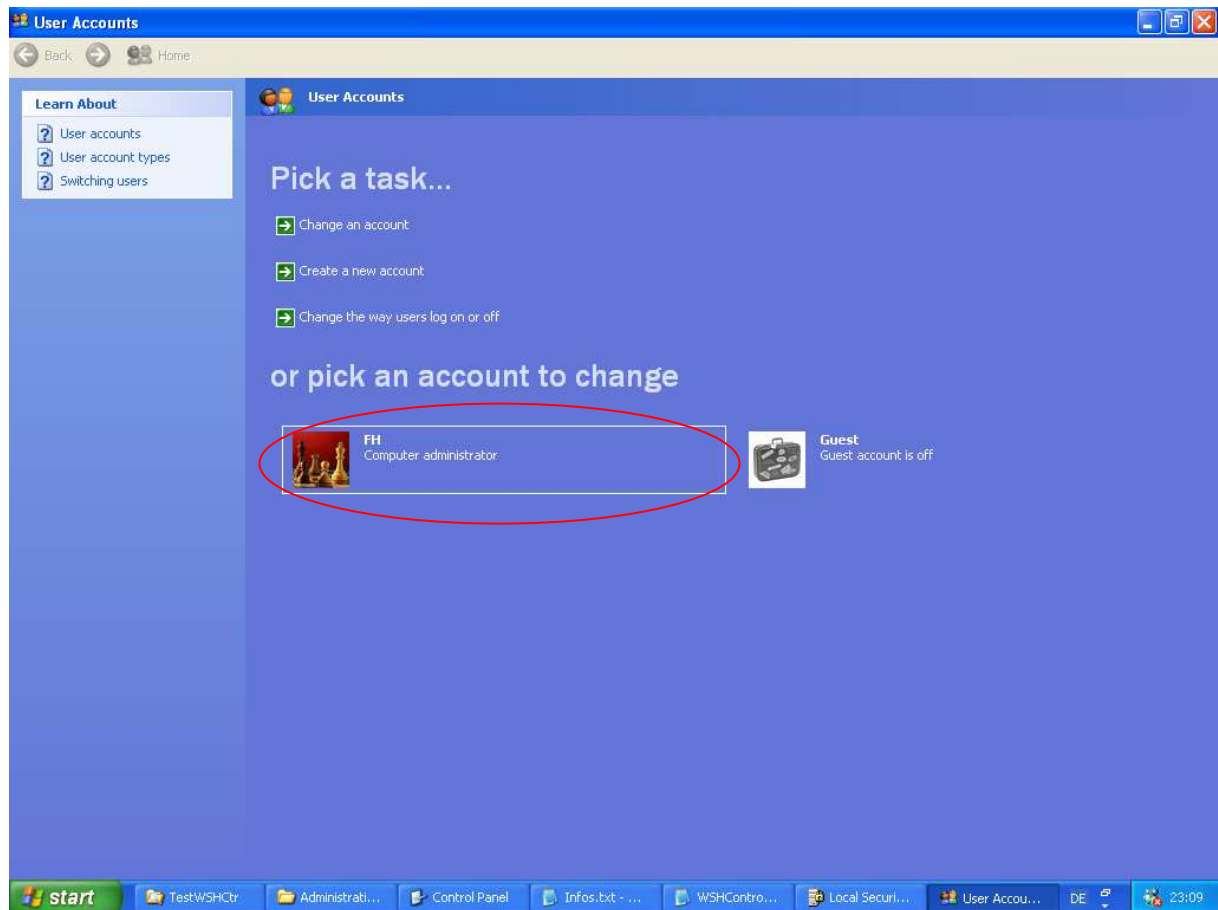
<sup>347</sup> C:\Documents and Settings\All Users\Start Menu\Programs\Administrative Tools



**Figure 46:** Snapshot of the Security Policy console.<sup>348</sup>

A further prerequisite is that the same password must be set on both machines for the administrator. Without setting a password it doesn't work. A password for the administrator is created by choosing Start->Control Panel->User Accounts. Double-click the computer administrator like in figure 47 [Sp03].

<sup>348</sup> Part of MS Windows XP Pro

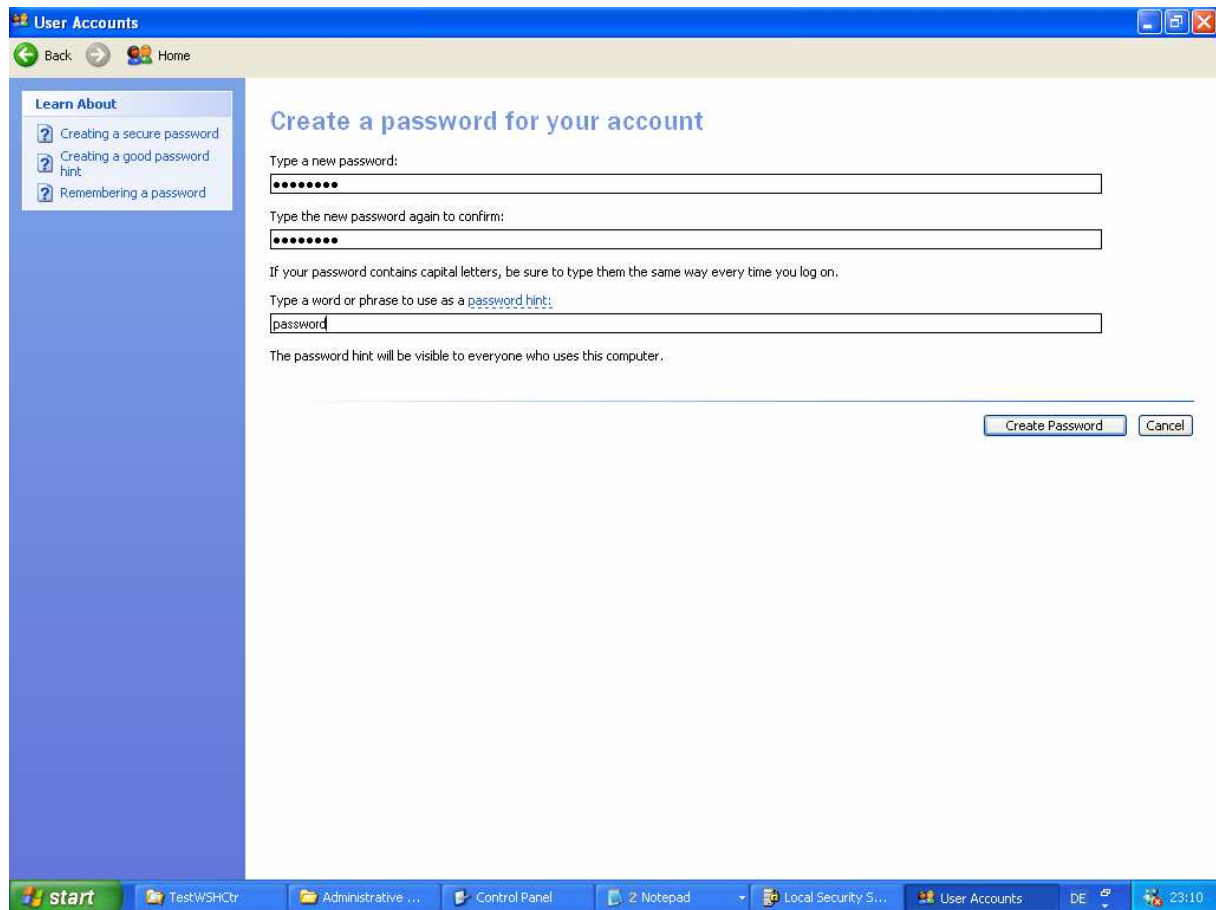


**Figure 47:** Snapshot of the start page of the User Accounts<sup>349</sup>

Then click Create a password and fulfill the form like in figure 48.

<sup>349</sup> Part of MS Windows XP Pro





**Figure 48:** Snapshot of the form for creating a password<sup>350</sup>

The control script code 32 contains additionally to the script code 30 the name of the machine (fhkcn) where the remote script should be executed. There is no folder specified because the remote script is located in the same folder as the control script but it is also possible to insert the folder [MS03]. The control script is on the control machine in the folder `c:\TestWSHCtrl` located. That is the same folder specification as the folder specification where the remote script will create the text file on the remote machine. If the control script is located in another folder on the control machine it doesn't work.

```

-----
-- WshController_OnMultipleMachines.rxs --
-----

-- Instantiation of the Controller Object
Controller = .OLEObject~New("WSHController")
-- The method CreateScript generates a WSHRemote object351

```

<sup>350</sup> Part of MS Windows XP Pro

<sup>351</sup> [MLWSHu]

```

RemoteScript = Controller~CreateScript("wshremote.vbs","fhkcn")
    -- The Execute method begins the execution of the remote script
RemoteScript~Execute -- object352.
    -- DO function which is executed until the value of the status is 2353
DO UNTIL RemoteScript~Status = 2
    CALL SysSleep 1
END

```

**Code 32:** WshController\_OnMultipleMachines.rxs<sup>354</sup>

This remote script OtherScript 9 is a Visual Basic Script file which is executed in the memory of the machine "fhkcn". If Object Rexx is not installed on the remote machine an Object Rexx remote script won't work. It creates a textfile with the name WSHDemo.txt in the folder C:\TestWSHCtrl of the remote machine. Into this text file the text "WSHController was here" is written.

```

Set fso = CreateObject("Scripting.FileSystemObject")
Set fsoFile = fso.CreateTextFile("C:\TestWSHCtrl\WSHDemo.txt", True)
fsoFile.WriteLine "WSHController was here."
fsoFile.Close

```

**OtherScript 9: Remote.vbs<sup>355</sup>**

### 13.7.2.3. WSHController and Events

The WSHController object has events. But these events can't be used with Object Rexx. Code 33 shows a control script which runs without error. It executes the remote script code 34 which launches the calculator on the same machine, but the Start event is not fired. The reason therefore is that the events of WSHController are not hand over to Object Rexx via the generic interfaces IDispatch and IUnknown. The events are hand over via special ID's. These ID's are not queried with Object Rexx [Doe03c].

<sup>352</sup> [MLWSHw]

<sup>353</sup> [MLWSHv]

<sup>354</sup> Modeled after [WSS02]

<sup>355</sup> Modeled after [WSS02]

```

-----
-- WshControllerWithEvents.rxs --
-----

-- Instantiation of the Controller Object
Controller = .EventsWithWSHctr~New("WSHController","WITHEVENTS")
-- The method CreateScript generates a WSHRemote object.356
RemoteScript = Controller~CreateScript("WSHRemoteCalc.rxs")
-- The ConnectObject method connects the object's event sources to
-- functions. Therefore the prefix "X" is used.357
WScript~ConnectObject(RemoteScript, "X")
-- The Execute method begins the execution of the remote script358.
RemoteScript~Execute
-- DO function which is executed until the value of the status is 2
DO UNTIL RemoteScript~Status = 2
    CALL SysSleep 1
END
-- Class which derives from OLEObject359
::CLASS EventsWithWSHctr SUBCLASS OLEObject
::METHOD XStart -- This method is invoked if the Start event is fired.
WScript~Echo("WSHRemote Start Event was fired!")

```

**Code 33: WshControllerWithEvents.rxs<sup>360</sup>**

The remote script code 34 executes the calculator with the `Exec` method<sup>361</sup>.

```

-----
-- WSHRemoteCalc.rxs --
-----

-- Instantiation of the Shell object
WshShell = .OLEObject~New("WScript.Shell")
WshShell~Exec("calc") -- Launches the Calculator

```

**Code 34: WSHRemoteCalc.rxs**

### 13.7.3. WshNetwork Object<sup>362</sup>

Code 30 describes the `WshNetwork` object. Thereby the computer, user and domain name are accessed with the equal named properties.

<sup>356</sup> [MLWSHu]

<sup>357</sup> [MLWSHcz]

<sup>358</sup> [MLWSHw]

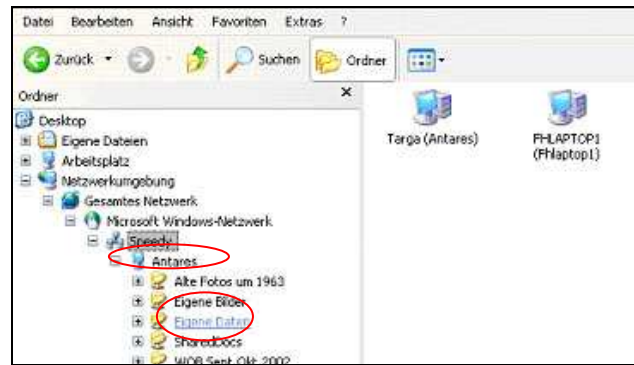
<sup>359</sup> c.p. 6.3.3.

<sup>360</sup> Modeled after [MLWSHda]

<sup>361</sup> c.p. 13.7.4.5.

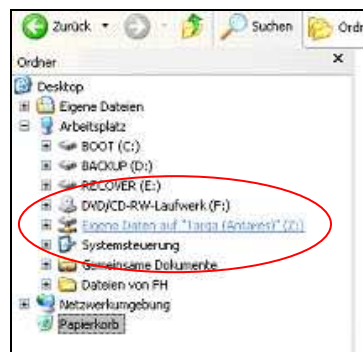
<sup>362</sup> [MLWSHx]

The available network drives are shown with the `EnumNetworkDrives` method [MLWSHy]. The `MapNetworkDrive("\\Server\Public")` [MLWSHz] and `RemoveNetworkDrive("name")` [MLWSHaa] methods create and delete a network drive. Figure 49 illustrates the network with the two network machines `Antares` that is the remote machine and `FHLAPTOP1`.



**Figure 49:** WSHNetwork1.JPG: \\Server =Antares\Public=Eigene Daten <sup>363</sup>

Figure 50 shows the created network drive.

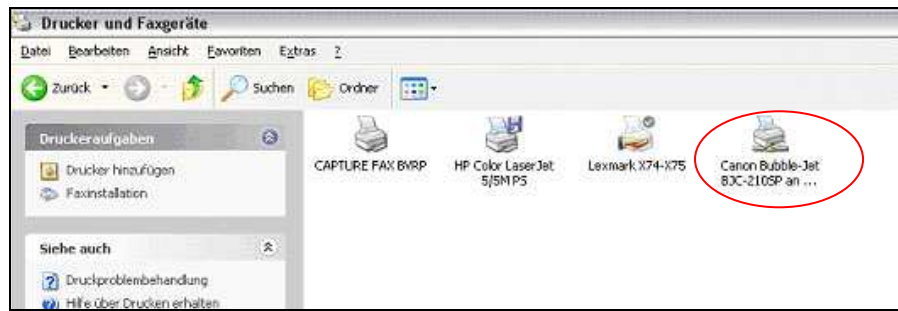


**Figure 50:** WSHNetwork2.JPG: Shows the new network drive" Z:"<sup>364</sup>

The `AddWindowsPrinterConnection("\\printserv\DefaultPrinter", "DriverName")` method [MLWSHab] enables the access to a remote network printer. The `EnumPrinterConnections` method [MLWSHac] shows the printers that are available on the machine or in the network for the user. At last the created printer connection is deleted with the `WshNetwork~RemovePrinterConnection` method [MLWSHad]. Figure 51 shows the new printer in the printer menu.

<sup>363</sup> Part of MS Windows XP

<sup>364</sup> Part of MS Windows XP



**Figure 51:** WSHNetwork3.JPG: Shows the network printer<sup>365</sup>

Code 35 demonstrates the WSHNetwork object.

```

-----
-- WshNetwork.rex --
-----

    -- To run this script a network connection is required, otherwise there
    -- is an error message. A printer must be installed on the remote
    -- machine. The printer needn't be plugged in.
    -- Instantiating of the Network object
WshNetwork = .OLEObject~New("WScript.Network")
SAY "Computer Name = " WshNetwork~ComputerName --Name of the user'sdomain366
SAY "User Name = " WshNetwork~UserName -- Name of a user367
SAY "Domain = " WshNetwork~UserDomain -- Name of the computer system368
    -- Tells how much network drives there are369.
SAY "There are " WshNetwork~EnumNetworkDrives~Length " network drives."
    -- Creates a new network drive with drive name "Z:". Works only if
    -- there is really a network connected
WshNetwork~MapNetworkDrive("Z:", "\\Antares\Eigene Daten")
    -- Message box
CALL RxMessageBox "Look up in the Windows Explorer to see the new" -
    " network drive!", "Information", "OK", "INFORMATION"
SAY "-----"
    -- Shows all networkdrives on the machine370
DO i = 0 to WshNetwork~EnumNetworkDrives~Length -1
SAY "Network drive: " WshNetwork~EnumNetworkDrives~Item(i)
END
SAY "-----"
WshNetwork~RemoveNetworkDrive("Z:") -- Deletes the NetworkDrive
    -- Message box

```

<sup>365</sup> Part of MS Windows XP

<sup>366</sup> [MLWSHae]

<sup>367</sup> [MLWSHaf]

<sup>368</sup> [MLWSHag]

<sup>369</sup> [MLWSHy]

<sup>370</sup> [MLWSHy]

```

CALL RxMessageBox "The new network drive is deleted!", "Information", "OK", "INFORMATION"
    -- Offers a remote Windows-based printer connection
WshNetwork~AddWindowsPrinterConnection("\\ANTARES\Canon Bubble-Jet BJC-210SP", -
    "Canon Bubble-Jet BJC-210SP")
SAY -- Blank line
    -- Offers with the odd-numbered items all networked printerUNC names371.
DO i = 1 to WshNetwork~EnumPrinterConnections~Length -1 by 2
SAY "Printer UNC names: " WshNetwork~EnumPrinterConnections~Item(i)
END
SAY -- Blank line
    -- Offers all printerports because the even-numbered items are the
    -- printer ports
DO i = 0 to WshNetwork~EnumPrinterConnections~Length -1 by 2
SAY "Printerport: " WshNetwork~EnumPrinterConnections~Item(i)
END
    -- Message box
CALL RxMessageBox "Look up in the Control Panel to see the new printer!", -
    "Information", "OK", "INFORMATION"
    -- Deletes the printer connection, .true means that the connection is
    -- removed whether if the user is connected or not
WshNetwork~RemovePrinterConnection("\\ANTARES\Canon Bubble-Jet BJC-210SP", .true)
SAY "The new printer is deleted."

```

**Code 35: WshNetwork.REX**

## 13.7.4. WshShell

This section discusses the `Run` method and the `Sendkeys` method, the access of the registry, the creation of shortcuts, `WSHEnvironment` and `WSHScriptExec`.

### 13.7.4.1. Run Method and SendKeys Method<sup>372373</sup>

Code 36 demonstrates the `Run` and `Sendkeys` methods. It is possible to execute applications and to simulate keystrokes with these methods.

The script is embedded in a HTML file. The file starts in the body. There the red background colour and a centred text are defined. A button is created. If the button is pressed the Object Rexx – described by “`language = 'Object Rexx'`”- routine “`doTheWork`” in the head is called. The tag `<br>` makes a line break.

<sup>371</sup> [MLWSHac]

<sup>372</sup> [MLWSHah]

<sup>373</sup> c.p. 9.1.3.

In the head the <title> tag writes the title. With <script language = "Object Rexx"> the Object Rexx code begins.

First the command window is opened with the Run method<sup>374</sup>. From this location the notepad is started with the SendKeys method. The notepad is remoted with the SendKeys method and then the notepad and the command window are closed.

With </script> the code ends.

```
<html>
<head>
<title>Embedding a script in HTML</title>
<script language="Object Rexx">
::routine doTheWork public
-----
-- Run method and Sendkeys method --
-----

    -- Instantiation of the Shell Object
Shell = .OLEObject~New("WScript.Shell")
Shell~Run("cmd") -- Opening of the command window with the run method
    -- The machine is sleeping for 2 seconds to see the command window
CALL SysSleep 2
    -- The path and the file name of notepad.exe are sent and prompted to
Shell~Sendkeys("c:\windows\system32\notepad.exe") -- the command window
    -- The machine is sleeping for 3 seconds for seeing the written text
CALL SysSleep 3
Shell~Sendkeys("~") -- An enter is sent to start the notepad
    -- Necessary because otherwise the machine writes the following text
CALL SysSleep 1 -- still to the command window
    -- Text sent to the notepad
Shell~Sendkeys("This text is written with the SendKeys method")
CALL SysSleep 2 -- Sleeping to read the text
Shell~Sendkeys("%d") -- The menu "File" is opened with the shortcut "ALT+d"
CALL SysSleep 2 -- Sleeping to see the file menu
    -- The item exit is chosen (in German it is named "Beenden" and so
Shell~Sendkeys("+b") -- it has the shortcut SHIFT+b)
CALL SysSleep 2 -- Sleeping to see the field
Shell~Sendkeys("{TAB}") -- With the tabulator the next button is activated
CALL SysSleep 2 -- Sleeping to see the field
Shell~Sendkeys("~") -- With the enter command the button is clicked
CALL SysSleep 2 -- Sleeping to see the command window
Shell~Sendkeys("exit") -- The exit command is prompted to the commandwindow
CALL SysSleep 2 -- Sleeping to see the command window with the exit command
```

---

<sup>374</sup> [MLWSHai]

```

    -- The exit command is executed and the command window is closed
Shell~Sendkeys( "~" )
</script>
</head>
<body bgcolor="red">
<center>
<font size=5>Press or click the button to start the script</font>
</center>
</br>
</br>
</br>
</br>
<center>
<input type=button value="press or click"
        language="Object Rexx"
        onmouseup="call doTheWork"
        onkeypress="call doTheWork">
</center>
</body>
</html>

```

**Code 36:**      **RunMethodAndSenkeysMethod.htm**

### 13.7.4.2.      Accessing the Registry

Code 37 demonstrates the RegWrite [MLWSHaj], RegRead [MLWSHak] and RegDelete [MLWSHal] methods. It is possible to access the registry with these methods.

First a new key and new values are created; after that they are accessed with the RegRead method and at last they are again erased<sup>375</sup>.

```

-----
-- WSHRegistry.rex --
-----

    -- Instantiating of the Shell object
Shell = .OLEObject~New("WScript.Shell")
    -- The "TESTKEY" key is created by the RegWrite method with the name
    -- keyname and it is specified by the final backslash. HKCR is the
    -- abbreviation for HKEY_CURRENT_USER
Shell~RegWrite("HKCU\TESTKEY\","keyname")
    -- The value with the name "name1" is created and has the value "1".
    -- It has the integer data type "REG_DWORD".
Shell~RegWrite("HKCU\TESTKEY\name1", "10", "REG_DWORD")

```

<sup>375</sup> c.p. 2.6.3.



```

-- The value with the name "name2" is created and has the value
-- "value". It has the string data type "REG_SZ".
Shell~RegWrite("HKCU\TESTKEY\name2", "value", "REG_SZ")
-- Message box
CALL RxMessageBox "Start the Registration Editor to watch the " -
  "created entries!", "Information", "OK", "INFORMATION"
-- Gives back the default value of the key
SAY Shell~RegRead("HKCU\TESTKEY\")
-- Gives back the value of the value-name name1
SAY Shell~RegRead("HKCU\TESTKEY\name1")
-- Gives back the value of the value-name name2
SAY Shell~RegRead("HKCU\TESTKEY\name2")
Shell~RegDelete("HKCU\TESTKEY\name1") -- The value name "name1" is deleted
-- Message box
CALL RxMessageBox "Watch the registry editor to see that "name1" is" -
  " deleted", "Information", "OK", "INFORMATION"
-- The key "TESTKEY" is deleted and accordingly also "name2"
Shell~RegDelete("HKCU\TESTKEY\")
-- Message box
CALL RxMessageBox "The created registry keys are deleted", -
  "Information", "OK", "INFORMATION"

```

**Code 37:** WSHRegistry.rex

### 13.7.4.3. Creation of Shortcuts

With the objects `WshShortcut` [MLWSHam], `WshUrlShortcut` [MLWSHan] and `WshSpecialFolders` [MLWSHao] shortcuts can be managed. The `WshSpecialFolders` object offers a comfortable access to folders like for example the Desktop, Programs, Favorites or Recent folder. With the methods of the `FileSystemObject`<sup>376</sup> object shortcuts can be copied and moved. Use the scripts in the sections 13.7.4.3.1. to 13.7.4.3.3. sequently, then the machine is in the end in the same state as in the beginning.

#### 13.7.4.3.1. Creation of a Shortcut

Code 38 creates a shortcut to the notepad and sets some parameters.

It is necessary to use the `Save` method in the last step, otherwise the changes are lost.

---

<sup>376</sup> c.p. 13.8.

```

-----
-- CreationOfAShortcut.rex --
-----

    -- Instantiation of the Shell Object
Shell = .OLEObject~New("WScript.Shell")
    -- Accesses the special folder "Desktop"377
DesktopPath = Shell~SpecialFolders("Desktop")
    -- Creation of a new shortcut with the description "Notepad"378
Shortcut = Shell~CreateShortcut(DesktopPath"\Notepad.lnk")
    -- Short description of the shortcut in the comment field379
Shortcut~Description = "Test Shortcut"
Shortcut~HotKey = "CTRL+ALT+n" -- Key-combination for the shortcut380
    -- Location of the icon of the shortcut and the index of the icon381
Shortcut~IconLocation = "notepad.exe,0"
    -- Location of the shortcut's executable file382
Shortcut~TargetPath = "c:\windows\notepad.exe"
    -- Activates the window and the value "3" displays it as a maximized
Shortcut~WindowStyle = 3 -- window383
    -- Assigns or identifies the working directory to/of a shortcut384
Shortcut~WorkingDirectory = DesktopPath
Shortcut~Save -- Saves the shortcut385

```

**Code 38:**      **CreationOfAShortcut.rex**<sup>386</sup>

### 13.7.4.3.2.      Creation of an UrlShortcut

Code 39 creates an UrlShortcut to the Internet browser and links to the Homepage of the University of Augsburg.

<sup>377</sup> [MLWSHap]

<sup>378</sup> [MLWSHq]

<sup>379</sup> [MLWSHar]

<sup>380</sup> [MLWSHas]

<sup>381</sup> [MLWSHat]

<sup>382</sup> [MLWSHau]

<sup>383</sup> [MLWSHav]

<sup>384</sup> [MLWSHaw]

<sup>385</sup> [MLWSHaq]

<sup>386</sup> Modelled after [MLWSHap]

```

-----
-- CreationOfAnUrlShortcut.rex --
-----

-- Instantiation of the Shell Object
WshShell = .OLEObject~New("WScript.Shell")
-- Accesses the special folder "Desktop"387
DesktopPath = WshShell~SpecialFolders("Desktop")
-- Creation of a new shortcut with the description "Uni-Augsburg"388
UrlLink = WshShell~CreateShortcut(DesktopPath"\Uni-Augsburg.url")
-- Location with the shortcut's URL389
UrlLink~TargetPath = "http://www.uni-augsburg.de"
UrlLink~Save -- Saves the shortcut390

```

**Code 39: CreationOfAnUrlShortcut.rex**

### 13.7.4.3.3. Deletion of a Shortcut

The last example code 40 deletes the both generated shortcuts.

Therefore, the `FileSystemObject` object is used.

```

-----
-- DeletionOfAShortcut.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
-- Instantiation of the Shell Object
WshShell = .OLEObject~New("WScript.Shell")

```

---

<sup>387</sup> [MLWSHap]

<sup>388</sup> [MLWSHaq]

<sup>389</sup> [MLWSHau]

<sup>390</sup> [MLWSHaq]

```

-- Accesses the special folder "Desktop"391
DesktopPath = WshShell~SpecialFolders("Desktop")
-- Deletion of the shortcuts "Notepad.lnk" and "Uni-Augsburg.lnk"392
fso~DeleteFile(DesktopPath"\Notepad.lnk")
fso~DeleteFile(DesktopPath"\Uni-Augsburg.url")

```

**Code 40:**      **DeletionOfAShortcut.rex**

#### 13.7.4.4.      WshEnvironment<sup>393</sup>

This object allows the access to environment variables. Therefore the Environment property<sup>394</sup> of the WshShell object is used.

The following code 41 demonstrates the values of several environment variables. In the end the Length property offers the number of variables in a specific environment.

```

-----
-- WshEnvironment.rex --
-----

-- Instantiation of the Shell Object
Shell = .OLEObject~New("WScript.Shell")
-- Determination of the location of the environment variable with the
-- Environment property, here "SYSTEM", and the Environment variable
-- determined with the item property395
Envsnop = Shell~Environment("SYSTEM")~item("NUMBER_OF_PROCESSORS")
Envspa = Shell~Environment("SYSTEM")~item("PROCESSOR_ARCHITECTURE")
Envspr = Shell~Environment("SYSTEM")~item("PROCESSOR_REVISION")
Envspi = Shell~Environment("SYSTEM")~item("PROCESSOR_IDENTIFIER")
Envutmp = Shell~Environment("USER")~item("TMP")
-- Results of the environment variables
SAY "There are " envsnop "processors on the machine"
SAY "The processor type is " envspa
SAY "The processor version is: " envspr
SAY "Processor ID: " envspi
SAY "The directory for storing temporary files is: " envutmp

```

<sup>391</sup> [MLWSHap]

<sup>392</sup> [MLWSHax]

<sup>393</sup> [MLWSHay]

<sup>394</sup> [MLWSHaz]

<sup>395</sup> [MLWSHba]

```

SAY -- blank line
    -- Location of the environment variable. Here "PROCESS"
envp = Shell~Environment("PROCESS")
    -- The length property returns the number of variables in the process
    -- environment396
SAY "There are" envp~length "environment variables in the PROCESS environment"

```

**Code 41: WshEnvironment.rex**

### 13.7.4.5. WshScriptExec<sup>397</sup>

Code 42 demonstrates the `Exec` method [MLWSHbd] with the `Status` property [MLWSHbe]. An application can be launched with the `Exec` method. The `Status` property shows if the application is running or not.

Note that here the `Shell` object is instantiated in another way. The WSH method `WScript~CreateObject("WScript.Shell")` [MLWSHbf] is used to instantiate the `Shell` object. The script runs the Notepad editor with the `Exec` method and then asks if the user wants to close the Notepad or not. With the `Status` property the action of the user is checked and it will be controlled if the Notepad editor is still running or not.

The first `SysSleep` function is required because otherwise the notepad is in front of the message box. This is because of the asynchronous start of the Notepad editor. The line that is sequently to the line that starts the notepad, is immediately executed without waiting for the notepad. But it takes more time to load the notepad than the message box. Therefore, the message box is earlier on the display and then the notepad that comes later is in front of it [Doe02a].

```

-----
-- WshScriptExec.rxs--
-----
    -- Instantiation of the Shell Object
Shell = WScript~CreateObject("WScript.Shell")
OExec = Shell~Exec("notepad") -- Runs the notepad
    -- Needed because otherwise the notepad is in front of the messagebox
CALL sys.sleep 1
    -- A message box occurs on the display
CALL RxMessageBox "You can close the notepad or you can leave it open!",-

```

<sup>396</sup> [MLWSHbb]

<sup>397</sup> [MLWSHbc]

```
"Result", "OK", "INFORMATION"
CALL sysssleep 5 -- The script pauses 5 seconds
OStatus = OExec~Status -- Handover of the status information
    -- Analysis if the application is running or not.
IF OStatus = 0 then answer = "The notepad is running"
ELSE answer = "The notepad is closed"
    -- Message with the result
CALL RxMessageBox answer, "Result", "OK", "INFORMATION"
```

**Code 42:** WshScriptExec.rxs

## 13.8. FileSystemObject Object

The `FileSystemObject` object makes it possible to create or delete folders, to get information about drives, to create and delete text files and other things. It is used for script control for applications developed with other languages, to create Web pages with HTML and for Windows Script Host. The object model is contained in the Scripting type library (`ScrRun.dll`) [MLWSHbg] and consist of the `FileSystemObject`, `Drive`, `File`, `Folder` and `TextStream` objects [MLWSHbh]. By instantiating, the `FileSystemObject` object the name of the type library is `Scripting` [MLWSHbi]. Execute the scripts demonstrated in the sections 13.8.3 to 13.8.7 sequentially, then the machine is in the end in the same state as in the beginning.

### 13.8.1. The AvailableSpace Property

Code 43 demonstrates the `AvailableSpace` property. The script shows how much memory is free on the drive [MLWSHbj].

```

-----
-- FSOAvailableSpace.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
-- The GetDriveName method398 offers a string with the name of the drive
-- of the folder and the GetDrive method399 gives back a Drive object
d = fso~GetDrive(fso~GetDriveName("c:\"))
-- The AvailableSpace property shows the free space on the drive
e = d~AvailableSpace
SAY e "bytes" -- The free space is prompted to the display

```

**Code 43: FSOAvailableSpace.rex**

### 13.8.2. DriveType Property

Code 44 shows the DriveType property [MLWSHbm]. The script checks the type of a drive.

```

-----
-- FSODriveType.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
d = fso~GetDrive("c:\") -- The GetDrive method gives back a Drive object400
a = d~DriveType -- Determining the DriveType Property
SELECT -- Corresponding to the DriveType value the type is determined401
WHEN a = 0 THEN b = "Unknown"
WHEN a = 1 THEN b = "Removable"
WHEN a = 2 THEN b = "Fixed"

```

---

<sup>398</sup> [MLWSHbl]

<sup>399</sup> [MLWSHbk]

<sup>400</sup> [MLWSHbk]

<sup>401</sup> [MLWSHbm]

```

WHEN a = 3 THEN b = "Network"
WHEN a = 4 THEN b = "CD-ROM"
WHEN a = 5 THEN b = "RAM Disk"
END
SAY "The drive type is " b -- The drive type is prompted

```

**Code 44:** FSODriveType.rex<sup>402</sup>

### 13.8.3. Creation of a Folder with a WSF File<sup>403</sup>

Code 45 creates two new folders. It is stored in a WSF file. Therefore it is structured in XML code<sup>404</sup>.

In the first line the XML version is described. Then how to handle errors and debugging is described. The <package> element implies the <job> element, which includes the script block. The script block starts with <script language="Object Rexx">. Inside the script the <![CDATA[ ...]]> section makes the entire <script> element opaque and ensures that characters in the <script> element are not handled as XML characters. There is an error message if the folder already exists.

```

<?xml version="1.0"?>
<?job error="true" debug="true" ?>
<package>
<job>
<script language="Object Rexx">
<![CDATA[
-----
-- FSOCreationOfANewFolder.wsf --
-----

    -- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
fso~CreateFolder("c:\Test1") -- Creation of the new folder named "Test1"

```

<sup>402</sup> Modelled after [MLWSHbm]

<sup>403</sup> [MLWSHbn]

<sup>404</sup> c.p. 9.1.4.



```
fso~CreateFolder("c:\Test2") -- Creation of a second folder names "Test2"
]]>
</script>
</job>
</package>
```

**Code 45:** FSOCreationOfANewFolder.wsf

### 13.8.4. Creating a Text File

Code 46 creates a text file and writes text into this file.

```
-----
-- FSOCreationOfATtextfile.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
-- Creation of the textfile "textfile.txt" in the folder "c:\Test"
-- "2" allows writing the file and ".true" allows the creation of the
-- filename if it doesn't exist405
textfile = fso~Opentextfile("c:\Test1\Textfile.txt", 2, .True)
-- Writing text into the textfile406
textfile~Writeline("This is a test text.")
```

**Code 46:** FSOCreationOfATtextfile.rex

### 13.8.5. Attribute Property

The Attribute property [MLWSHbq] shows the type of a file as in code 47 demonstrated.

```
-----
-- FSOFileAttribute.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
-- The File object is returned and the Attributes are found out 407
a = fso~getfile("c:\test1\textfile.txt")~Attributes
SELECT -- Corresponding to the Attribute value the type is determined
WHEN a = 0 THEN b = "Normal file";
WHEN a = 1 THEN b = "Read-only file";
WHEN a = 2 THEN b = "Hidden file";
WHEN a = 4 THEN b = "System file";
```

<sup>405</sup> [MLWSHbo]

<sup>406</sup> [MLWSHbp]

<sup>407</sup> [MLWSHbr]

```

WHEN a = 8 THEN b = "Disk drive volume label";
WHEN a = 16 THEN b = "Folder or directory";
WHEN a = 32 THEN b = "Archive";
WHEN a = 64 THEN b = "Link or shortcut";
WHEN a = 128 THEN b = "Compressed file";
END

    -- Text message of the file attribute
CALL RxMessageBox b, "Attribute", "OK", "INFORMATION"

```

**Code 47:**      **FSOFileAttribute.rex**

### 13.8.6. Copying a File

This text file is now copied to another folder with a new name with code 48.

```

-----
-- FSOCopyingAFileWithANewName.rex --
-----

    -- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
file1 = "c:\Test1\Textfile.txt" -- File name with folder name
file2 = "c:\Test2\Textfile_newname.txt" -- Target folder with new file name
fso~CopyFile(file1, file2) -- CopyFile method to copy the file408

```

**Code 48:**      **FSOCopyingAFileWithANewName.rex**

### 13.8.7. Deleting Files and Folders

In the last step in code 49 the files and folders are deleted with two different kinds. First with the `DeleteFile` [MLWSHbt] and `DeleteFolder` [MLWSHbu] method, and then the other folder with the `Delete` method [MLWSHbv].

---

<sup>408</sup> [MLWSHbs]

```

-----
-- FSODeletionOfTheFilesAndFolders.rex --
-----

-- Instantiation of the FileSystemObject
fso = .OLEObject~New("Scripting.FileSystemObject")
-- The file "Textfile_newname.txt" is deleted
fso~DeleteFile("c:\Test2\Textfile_newname.txt")
fso~DeleteFolder("c:\Test2") -- Deletion of the folder named "Test2"
-- Returns a folder object with the folder "c:\Test1" named "path"409
path = fso~GetFolder("c:\Test1")
path~Delete -- The folder "c:\Test1" is deleted inclusive all files

```

**Code 49:** FSODeletionOfTheFilesAndFolders.rex

## 13.9. Dictionary Object<sup>410</sup>

Item pairs and data keys are associated and stored with that object. Code 50 demonstrates the `Add` [MLWSHby] and the `Exists` methods [MLWSHbz]. For instantiating the type library `Scripting` is used.

```

-----
-- DictionaryObject.rex --
-----

-- Instantiation of the Dictionary Object
dic = .OLEObject~New("Scripting.Dictionary")
-- Some keys e.g."a"and some items e.g."Wirtschaftsinformatik"are added
dic~add("a", "Wirtschaftsinformatik")
dic~add("b", "Logistik")
dic~add("c", "Informatinsoekonomik")
a = dic~exists("a") -- The Exist method shows if an array occurs
-- If the Exist method has the value "1" then the array occurs and the
IF a = 1 then b = "Array a exists" -- message is "Array a exists"
-- If the value is different from "1" then the array doesn't exist
ELSE b = "Array a doesn't exist"
SAY b -- The message "b" is prompted

```

**Code 50:** DictionaryObject.rex

## 13.10. Security in Windows Script Host

Scripting manifests problems because of destroying, spying or changing of contents especially if there is an access via network. The problem is that all functions of the host application are available to a script. There is also the danger of viruses

<sup>409</sup> [MLWSHbw]

<sup>410</sup> [MLWSHbx]

operating with this technique and there is no “sandbox” available for WSH [Fla02d,p21f]. This section discusses how to create a test certificate and how to use the `Scripting.Signer` object with the methods `Sign`, `Verify`, `SignFile` and `VerifyFile`.

The user of a script can prove the authenticity of it [MLWSHca]. With the CryptoAPI tools certificates, certification revocation lists (CRLs) and certificate trust lists (CTLs) can be seen and managed and files can be digitally signed and used with Microsoft Authenticode [MLWSHcb]. To sign a script a certificate by a commercial certification authority or the administrator is necessary [MLWSHcc].

The `SelfCert.exe` tool<sup>411</sup> provided by the Windows SDK or the Microsoft Office enables the creation of a certificate for test purpose. This certificate has no root certificate authority [CI01]. Figure 52 shows the user-interface of `SelfCert.exe`.



**Figure 52:** Creation of a certificate<sup>412</sup>

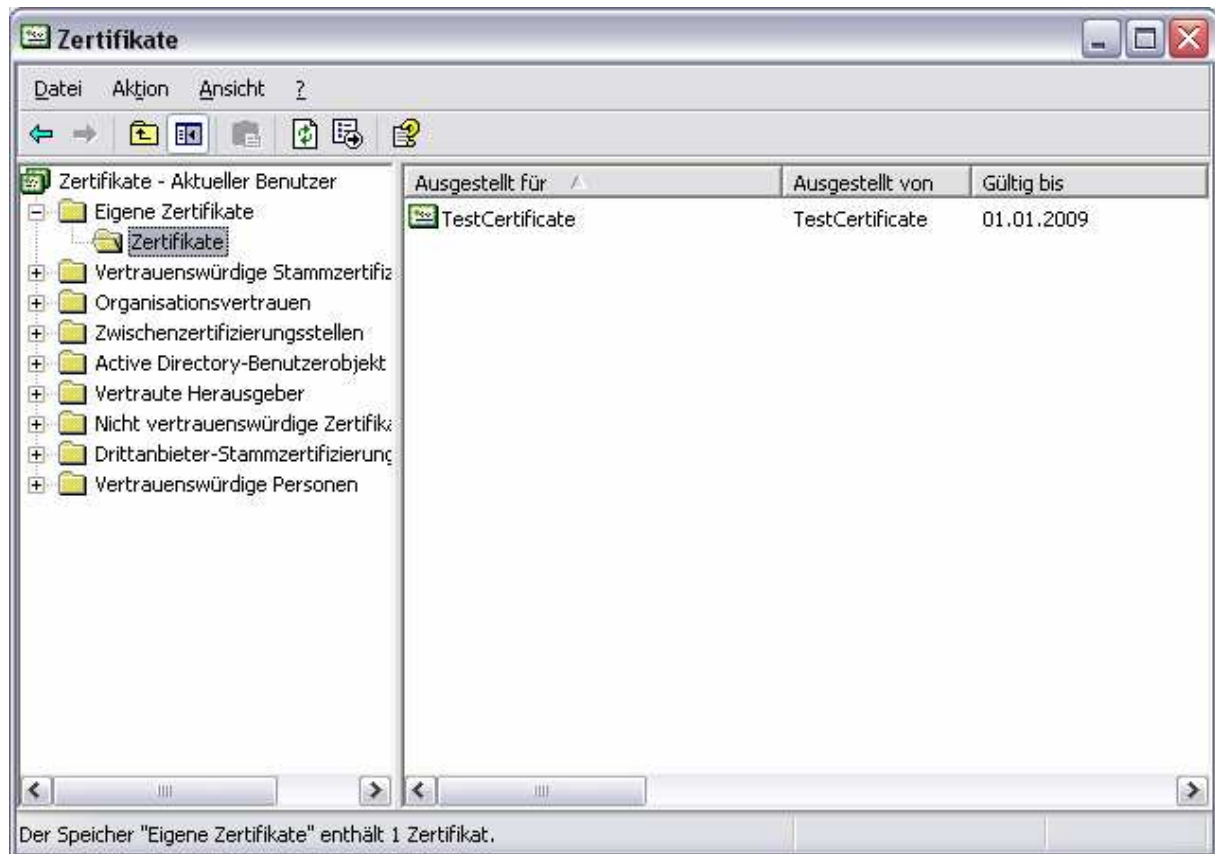
The Certificate-Snap-In shows in figure 53 the new created own certificate<sup>413</sup>.

---

<sup>411</sup> Start->Search->Files and Folders->Selfcert.exe

<sup>412</sup> Part of MS Windows XP

<sup>413</sup> Start->Run->Certmgr.msc



**Figure 53: Certificate Snap-In<sup>414</sup>**

An administrator can control scripts with so-called Software Restriction Policies. In this way the scripts can't implement any illegal activities in the operating system. A trust level requires the file content, the path from which the script is running, and any signature information in the script [MLWSHcd].

In the registry key `\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows Script Host\Settings\TrustPolicy` the verification policy is placed. With the verification policy it is defined if the signature verification is turned on or off by the administrator. If it is turned off, all scripts run on the system. If it is turned on, only scripts which are correctly signed or which have a special permission run on the system [MLWSHce].

With the `SignFile` method of the `Scripting.Signer` object (since WSH 5.6<sup>415</sup>) it is possible to sign scripts. The object is with Object Rexx instantiated in the following

<sup>414</sup> Part of MS Windows XP

<sup>415</sup> <http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp>

manner: `WshSigner = .OLEObject~New("Scripting.Signer")`. The `SignFile` method has the following syntax: `WshSigner~SignFile("File to sign", "name of the signing certificate", "name of certification store")`. The default store for the certificates is "my" [Es02].

A digital signature block of comments is written in a script when a script is signed. If there is an attempt to change the script, the signature is invalidated [MLWSHcc].

### 13.10.1. SignFile and VerifyFile Methods

This section discusses the `SignFile` and the `VerifyFile` methods. The `SignFile` method signs a file and the `VerifyFile` method checks if the file can be verified.

Code 51 is the script, which is to be signed. It is a simple Object Rexx program structured in XML<sup>416</sup>, which runs the calculator. It is also possible to sign a `.vbs` script, but it is not possible to sign an Object Rexx script, which has the extension `.rex` or `.rxs`.

```
<job>
<script language="Object Rexx">
WshShell = .OLEObject~new("WScript.Shell")
WshShell~Exec("calc")
</script>
</job>
```

**Code 51:**      **ScriptWhichIsToSign.wsf**

This script is signed with code 52 using the certificate created before.

---

<sup>416</sup> c.p. 9.1.4.

```

-----
-- WSH_Scripting_Signer_Signfile.rex --
-----

-- Instantiates the Scripting.Signer object
Signer = .OLEObject~New("Scripting.Signer")
-- Signs the file "ScriptWhichIsToSign" with the certificate
-- "TestCertificate" which is stored in "my"417
Signer~SignFile("ScriptWhichIsToSign.wsf", "TestCertificate", "my")

```

**Code 52:**      **WSH\_Scripting\_Signer\_SignFile.rex**

After signing code 51 the source code of code 51 looks like figure 54.

```

<job>
<script language="Object Rexx">
WshShell = .OLEObject~new("WScript.Shell")
WshShell~Exec("calc")
</script>

<signature>
** SIG ** MIID/AYJKoZIhvcNAQcCoIID7TCCA+kCAQExDjAMBggq
** SIG ** hkiG9w0CBQUAMGYGCisGAQQBgjcCAQSGWDBWMDIGCisG
** SIG ** AQQBgjcCAR4wJAIBAQQQcAVhGs44lBGiowAQS9NQkAIB
** SIG ** AAIABAAIBAAIBAAIBADAgMAwGCCqGSIb3DQIFBQAEECY3
** SIG ** uzM6FTS6ewTr6hHRLVmgggIhMIIHTCCAAYagAwIBAgIQ
** SIG ** wXM5sMaLvZ5CyF0Ifq2E1DANBgkqhkiG9w0BAQQFADAA
** SIG ** MRgwFgYDVQQDEw9UZXRhbnQ2VydGlmawWnhdGUwHhcNMDIx
** SIG ** MjMxMjMwMDAwWhcNMDgxMjMxMjMwMDAwWjAaMRgwFgYD
** SIG ** VQQDEw9UZXRhbnQ2VydGlmawWnhdGUwZ8wDQYJKoZIhvcN
** SIG ** AQEBBQADgY0AMIGJAoGBAK3Y9YvOmkB0kwjTbDeUO/2P
** SIG ** Fn6Urhpvngt7LSHJtaEO7d7dVGzlrMWhd0GcQ0VCDECW
** SIG ** EfOTGhFRKpyF+RbOQdlvYW0um0013PH7manCM0brGgRo
** SIG ** AjMjsg+d4Ml3lRswlGAX1VjJG4DK1XRaHUFvsrUkm5xY
** SIG ** konh8liwHTFMeGMXAgMBAAGjZDBiMBMGAlUdJQQMMAoG
** SIG ** CCsGAQUFBwMDMEsGA1UdAQREMEKAEPs0qW4wGdXwLRtf
** SIG ** 3p8laHehHDAaMRgwFgYDVQQDEw9UZXRhbnQ2VydGlmawWnhdGUwHhcNMDIx
** SIG ** dGwCEMFzObdGi72eQshdCH6thNQwDQYJKoZIhvcNAQEE
** SIG ** BQADgYEANx5bZnYOM4FBmMStOhmAqO8E6auR0bANxQ74
** SIG ** MBR+Q3WUjlrEXKfaC+FgHCdgYpV2zefA14g1k4g21Xl
** SIG ** bh6OT+RsSJjsaqimzJD107IpqdP8mNBTFHzZKpVXkw8m
** SIG ** D9g1v8FTbP0glz8gWw9gi8mZwpzV5V22Wi9pyXcDVSsA
** SIG ** UZoxggFFMIIBQQIBATAuMBoxGDAWBgNVBAMTD1Rlc3RD
** SIG ** ZXJ0aWZpY2F0ZQIQwXM5sMaLvZ5CyF0Ifq2E1DAMBggq
** SIG ** hkiG9w0CBQUAoGwwEAYKKwYBBAGCNwIBDDECMAAwGQYJ
** SIG ** KoZIhvcNAQkDMQwGCisGAQQBgjcCAQQwHAYKKwYBBAGC
** SIG ** NwIBCzEOMAwGCisGAQQBgjcCARUwHwYJKoZIhvcNAQkE

```

<sup>417</sup> [MLWSHcf]

```

** SIG ** MRIEEEDXAc3Kwc7F1vi/5epiXUcwDQYJKoZIhvcNAQEB
** SIG ** BQAEgYAg2BiyqKufwPeFwneIAeC6Nwnt2i8KfBjnu9Dw
** SIG ** MYoWvniJbzuWvF79M/szV0oVY2keaHwmWSRod+i0SRTy
** SIG ** h3Kp9edz9MRubzq42LUWVG5GE8N6Wv0G6R8KkPy6CZ0V
** SIG ** c2ZciCuF/s2Y+pMXPwjV92Rv/KhPFzz1Au3nSUia9vH3jg==
</signature>
</job>

```

**Figure 54: Signed source code of code 50**

With the `VerifyFile` method of the `Signer` object a script can be verified. This method checks if the signature is authenticated in the Trusted Publisher List, the legitimacy of the signature and if the script is manipulated [MLWSHcg].

The syntax is as follows: `WshSigner~VerifyFile("FileName", ShowUI)`. `ShowUI` is a boolean value which occurs on some operating systems if a dialog box is used to offer more trust information [MLWSHch].

Code 53 demonstrates the `VerifyFile` method.

```

-----
-- Signer_VerifyFile.rxs --
-----

-- Instantiates the Scripting.Signer object
WshSigner = .OLEObject~new("Scripting.Signer")
-- Accesses the argument with the name file418
File = WScript~Arguments~Named("file")
-- Checks if the argument with the name ui is present. Otherwise there
-- is the message that the script is not trusted419
UI = WScript~Arguments~Named~Exists("ui")
-- The script is verified and the result is given to "decision"
decision = WshSigner~VerifyFile(File, UI)
-- If decision has the value "1" then the message that the file is
If decision = 1 Then WScript~Echo(File " is trusted.") -- trusted is posted
-- Otherwise the message that the file is not trusted is posted
Else WScript~Echo(File " is untrusted.")

```

**Code 53: Signer\_VerifyFile.rxs<sup>420</sup>**

Figure 55 illustrates the command in the MS-DOS shell<sup>421</sup> to verify code 51.

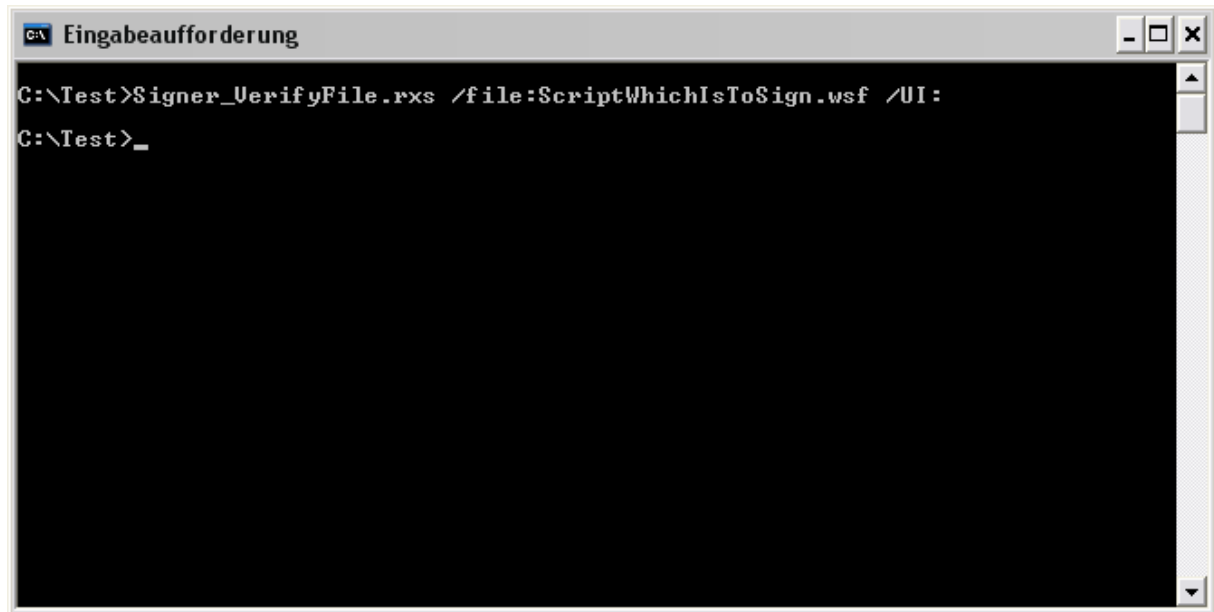
<sup>418</sup> [MLWSHch]

<sup>419</sup> [MLWSHs]

<sup>420</sup> Modelled after [MLWSHch]

<sup>421</sup> Start->Run->Command





**Figure 55:** MSDOS Shell with the command to sign the script<sup>422</sup>

After that the following question like in figure 56 appears. Here the user can decide on its own if the certificate is trusted or not.

---

<sup>422</sup> Part of MS Windows XP



**Figure 56:** Message box to verify the script.<sup>423</sup>

### 13.10.2. Sign and Verify methods

The next two scripts demonstrate the `Sign` method and the `Verify` method. The `Sign` method signs a script that is stored in a string [MLWSHci].

This is shown with OtherScript 10. Prerequisite is that a certificate is created with the name "TestCertificate"<sup>424</sup>. First, a Visual Basic Script file is shown which demonstrates the same issue.

```
Dim Signer, UnsignedText, SignedText
Set Signer = CreateObject("Scripting.Signer")
UnsignedText = _
    "Dim X " & vbCrLf & _
    "X = 123" & vbCrLf & _
```

<sup>423</sup> Part of MS Windows XP

<sup>424</sup> c.p. 13.10.

```

WScript.Echo X" & vbCrLf
SignedText = Signer.Sign(".VBS", UnsignedText, "Your Certificate Name _
Here")

```

**OtherScript 10: Sign method with Visual Basic Script**

Code 54 is the Object Rexx script that signs script text. Here is signed a script text which contains an Object Rexx script that is structured in XML. Code 54 contains another script text as OtherScript 10.

```

-----
-- WSH_ScriptingSigner_Sign.rex --
-----

-- Instantiates the Scripting.Signer object
Signer = .OLEObject~New("Scripting.Signer")
-- Script text which is to be signed. "0D0A"x is used for line break.
UnsignedText = ("<job>" "0D0A"x "<script language='Object Rexx'>" "0D0A"x ,
"WshShell = .OLEObject~new('WScript.Shell')" "0D0A"x ,
"WshShell~Exec('calc')" "0D0A"x "</script>" "0D0A"x "</job>")
-- The Sign method signs the script text. The first parameter is the
-- file extension which determines the type of the script file. The
-- second parameter is a string with the script text which is to be
-- signed. The third parameter contains the certificate name425.
SignedText = Signer~Sign(".WSF", UnsignedText, "TestCertificate")
SAY SignedText -- Prints the signed script code with the hash code

```

**Code 54: WSH\_ScriptingSigner\_Sign.rex**

The next script verifies a script text with the Verify method [MLWSHci].

First Visual Basic Script file OtherScript 11 is shown which demonstrates the same issue.

```

Dim Signer, UnsignedText, Trusted
Set Signer = CreateObject("Scripting.Signer")
UnsignedText = _
    "Dim X " & vbCrLf & _
    "X = 123" & vbCrLf & _
    "WScript.Echo X" & vbCrLf
Trusted = Signer.Verify(".VBS", UnsignedText, True)

```

**OtherScript 11: Verify method with Visual Basic Script**

Code 55 verifies the script text that contains Object Rexx script code that is structured in XML. Code 54 contains Object Rexx script text that is structured in XML and OtherScript 11 contains a VBS script text.

<sup>425</sup> [MLWSHci]

```

-----
-- WSH_ScriptingSigner_Verify.rex --
-----

-- Instantiates the Scripting.Signer object
Signer = .OLEObject~New("Scripting.Signer")
-- Script text which is to be verify. "0D0A"x is used for line break.
UnsignedText = ("<job>" "0D0A"x "<script language='Object Rexx'>",
"0D0A"x "WshShell = .OLEObject~new('WScript.Shell')" "0D0A"x ,
"WshShell~Exec('calc')" "0D0A"x "</script>" "0D0A"x "</job>")
-- The Verify method verifies the script text. The first parameter is
-- the file extension which determines the type of the script file.
-- The second parameter is a string with the script text which is to
-- be verify. The third parameter contains the ShowUI argument.
-- If it is set on true, dialog boxes are created if the trust
-- cannot be determined426.
Signer~Verify(".WSF", UnsignedText, .true)

```

**Code 55: WSH\_ScriptingSigner\_Verify.rex**

After that the following question, like in figure 57, appears. Here the user can decide on its own if the certificate is trusted or not.

---

<sup>426</sup> [MLWSHcj]



**Figure 57:** Dialog box which occurs after the Verify method is invoked.<sup>427</sup>

## 13.11. Starting Applications with WSH<sup>428</sup>

Code 56 which is embedded in HTML demonstrates the starting of MS Word instantiated by `.OLEObject~New("Word.Application")`. Starting that script runs the Microsoft Internet Explorer.

First to the second part of the script, to the body. There the yellow background colour is defined. Then a focused text is written and there are four line breaks. The following button includes the phrase `language="Object Rexx"`. This means that the phrase contains Object Rexx code, in this case the `doTheWork` routine is called.

The head of the file begins with the title. `<script language="Object Rexx">`, that means that here the script code is starting. Then the routine `doTheWork` starts. In that process MS Word is started, made visible and a new document is opened. The font name and size are chosen. The `Bold` property is set on `.true` and the `Italic` property is chosen by switching this property with the constant `"wdtoggle"`. The script ends with the `</script>` tag.

<sup>427</sup> Part of MS Windows XP

<sup>428</sup> c.p. 9.1.3.

```

<html>
<head>
<title>Embedding a script in HTML</title>
<script language="Object Rexx">
  ::routine doTheWork public
  -----
  -- WSH_EmbeddingAScriptInHTML.htm --
  -----

  Word = .OLEObject~New("Word.Application")
  Word~Visible = .TRUE
  Document = Word~Documents~Add
  Selection = Word~Selection
  Selection~Font~Name="Arial"
  Selection~Font~Size="24"
  Selection~Font~Bold = .TRUE
  Selection~Font~Italic = Word~GetConstant('wdToggle')
  Selection~TypeText("Hello World")
</script>
</head>
<body bgcolor="yellow">
<center>
<font size=5>Press or click the button to start MS Word</font>
</center>
</br>
</br>
</br>
</br>
<center>
<input type=button value="press or click"
      language="Object Rexx"
      onmouseup="call doTheWork"
      onkeypress="call doTheWork">
</center>
</body>
</html>

```

**Code 56:** WSH\_EmbeddingAScriptInHTML.htm

## 13.12. Windows Script Components

This section discusses how to generate own Windows Script components which can be used like a “normal” component. First the basics of Windows Script components are explained, then the structure is discussed with information how to register a component, how to expose the functions, how to excess the created component with the exposed functions and how to use remote instantiation of a component via a network. At the end the Windows Script Wizard is described.

### 13.12.1. Windows Script Components Basics

Windows script components make it possible to generate reusable [MLWSHck] COM components like Automation [MLWSHcl].

Thereby events, functions and attributes are used and embedded with any WSE language [Fla02d,p11]. Script components can be installed in applications like the Microsoft Windows Script Host or the Microsoft Internet Information Services (IIS). The used scripting language (e.g. Python, PScript, PERLScript, JScript and Microsoft Visual Basic Scripting Edition (VBScript)) must support the Microsoft ActiveX Scripting interfaces. The script component technology is extensible with DHTML behaviour [MLWSHcl]. The components can also be used via DCOM [Fla02d,p11].

WSC files can be created and be maintained with a simple text editor. The script components can be used in this way as COM components and all programs can use them [Fla02d,p11].

The Windows Script technology consists of the following parts [MLWSHcm]:

- Interface handlers to use COM interfaces
- The Script component file (.wsc) implies data about the COM component and it is in the XML format.
- Script component run-time (Scrobj.dll) to send out COM requests to the script.

### 13.12.2. Structure of Windows Script Components

WSC files are structured in XML.

The file starts with the declaration of the XML version and a declaration how to handle errors and debugging. If `error` is set on `true`, error messages are allowed and if `debug` is `true`, debugging is possible [MLWSHco]. The `<package>` element implies the `<component>` element, which includes the whole script component definition. It is possible that there are several `<component>` elements within a `<package>` element [MLWSHcn].

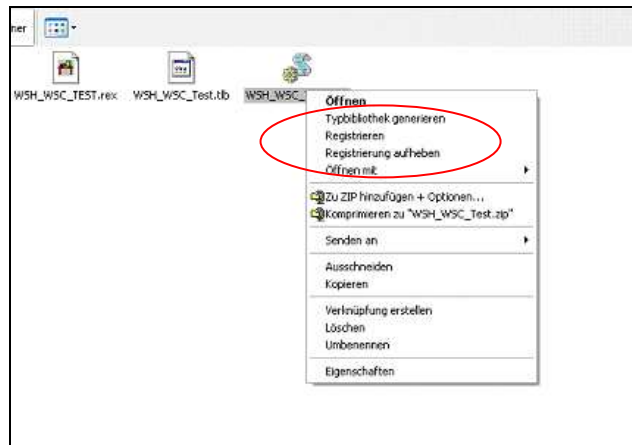
A WSC file consists of three parts: The registration, exposing the functions and the source code.

### 13.12.2.1. The Registration

This section describes the registration of components and how to create a type library.

The registration needs the ProgID and the CLSID (GUIGEN.EXE or UUIDGEN.EXE) of the component so that the component can be referenced. A short description and the version number are specified [MLWSHcp].

The component is registered by right-click on it in the Windows explorer and Register is chosen as shown in figure 58.



**Figure 58:** Snapshot of Registration field.<sup>429</sup>

Another way to register is to use the file Regsvr32.exe:

```
regsvr32 file:\\Location\ComponentName.wsc
```

The script component is together with the scrobj.dll registered on the machine. To unregister the component the component can be right-clicked in the Windows explorer and Unregister is chosen or Regsvr32.exe -u from the command prompt is started [MLWSHcq].

<sup>429</sup> Part of MS Windows XP



**Type Library**<sup>430431</sup>

For some host applications a type library is needed. Even if it is not needed it facilitates the work with the script and there are less errors. There is information about the functions and interfaces of the component.

There are two possibilities to create a type library. The first is to right-click in the explorer and to choose `Generate Type Library` like in figure 58 or it can be created dynamically from inside the script. There a file with a `.tlb` extension is created and it is registered in the Windows registry.

If the type library is created dynamically in code 57 the routine `Register` is called which is set public so that the data is accessible outside the script and an instance of the `GenerateTypeLib` object is generated. The `Shell` object is instanced. Then an instance of the `Component.GenerateTypeLib` object is generated to access the type library functionality. Next, a message box shows that the registration starts. The `AddUrl` method contains the location of the component file. The `Doc` property stores information about the component. With the `Path` property, the location and the name of the library file are stored. The internal name is described with the `Name` property. The `MajorVersion` and `MinorVersion` property contain integer values. The `GUID`<sup>432</sup> is different from the one for the script component. The `Write` method generates and registers the type library. The `Reset` method backspaces the settings so that a further type library can be generated. An additional message box, created with the `Popup` method, signals the end of the registering.

A further possibility to generate a type library is to use the file `Rundll32.exe` in the following manner:

```
Rundll32.exe path\scroobj.dll,GenerateTypeLib options
```

To end registration the routine `UnRegister` is called.

---

<sup>430</sup> [IBM01,p499ff]

<sup>431</sup> [MLWSHcr]

<sup>432</sup> c.p. 2.4.2.

### 13.12.2.2. Exposing the Functions

This section explains the exposing of properties, methods and events.

This part is bound with the `<public>` element [MLWSHcs].

*Properties* that are integrated with the `<property>` element [MLWSHcv] can consist of simple values (Property1) or of functions (Property2). The `<get>` element allows the reading of the property and the `<put>` element allows writing manifested with internal names [MLWSHct]. Properties are exposed in the following manner:

```
<property name="NameOfProperty">
    <get [internalName="GetName"] />
    <put [internalName="PutName"] />
</property>
```

“NameOfProperty” declares the name of the exposed property. “GetName” is obligatory and contains the name of the procedure that reads the value of the property. If there is only `<get>` element and no `<put>` element, the property is read-only. “PutName” is also obligatory. It includes the name of the procedure that writes the value of the property. If there is only a `<put>` element, the property is write-only.

Properties can be also declared in the following manner:

```
<property name=" NameOfProperty" [internalName="PropertyVariable"] />
```

Thereby “PropertyVariable” is obligatory and it is the name of the global variable in the scriptlet files `<script>` element. It holds the value for “NameOfProperty” [MLWSHcv].

With the `<method>` element [MLWSHcw] the *methods* are defined. They consist of their name and possibly of parameters [MLWSHcs].

*Events* are declared with the `<event>` element [MLWSHcx]. They need a method, here the method `FireTheEvent`, to be fired. They are launched with the `FireEvent` method. [MLWSHcu]

The functions can be used with dispatch identifiers (DISPID). This enables the functions to always have the same DISPID and the functions can get specific DISPIDS [MLWSHcu].

### 13.12.2.3. The Script Code<sup>433</sup>

The third part contains the script code of the component in code 57.

It begins with the <script> element where the script language is first mentioned. Then the default values for the Property1 and Property2 are given. The routine FireTheEvent is used to fire the event WSH\_WSC\_Event with the FireEvent method. The PropertyPut and PropertyGet routines for Property2 are noted. In this way the second property can be written and be read. Both methods takeover their values and calculate their formula. All routines are set on public because data is taken in from outside the script code. The file has the extension .wsc.

```
<?xml version="1.0"?>
<?component error="true" debug="true" ?>
<package>
<component>
<comment> Part 1 Registration</comment>
<registration
  progid="WSH_WSC_Test.WSHWSC"
  description="Demonstration of a WSC with Object REXX"
  version="1.0"
  clsid="{1C123B91-4F3E-4299-9064-26AC1F980C8A}" >
<script language="Object REXX"><![CDATA[
::Routine Register Public
  Shell = .OLEObject~New("WScript.Shell")
  Typelib = .OLEObject~New("Scriptlet.TypeLib")
  Shell~Popup("Start of registration")
  Typelib~AddURL("WSH_WSC_Test.wsc")
  Typelib~Doc = "WSH_WSC_TypeLib"
  Typelib~Path= "WSH_WSC_Test.tlb"
  Typelib~Name = "WSH_WSC_TypeLib"
  Typelib~MajorVersion = 1
  Typelib~MinorVersion = 0
  Typelib~GUID = "{CF53943D-428D-48f9-B8DD-5B0C4B850D9D}"
  Typelib~Write()
  Typelib~Reset()
  Shell~Popup("End of registering")
```

<sup>433</sup> [IBM01,p499ff]

```

::Routine Unregister Public
  Shell = .OLEObject~New("WScript.Shell")
  Shell~Popup("Ending of registration")
]]></script>
</registration>
<comment> Part 2: Exposing functions</comment>
<public>
<property name="property1">
</property>
<property name="property2">
  <get internalName = "propertyGet" />
  <put internalName = "propertyPut" />
</property>
<method name="km2Seamiles">
  <PARAMETER name="km"/>
</method>
<method name="FireTheEvent">
</method>
<method name="seamiles2km">
  <PARAMETER name="seamiles"/>
</method>
<event name="WSH_WSC_Event" >
</event>
</public>
<comment> Part3: Script Code </comment>
<script language="Object Rexx">
<![CDATA[
property1 = "This is the content of property1" -- Value of Property1
Property2value = "Defaultvalue" -- Defaultvalue of Property2
  -- Routine of the"FireTheEvent"method to fire the event "WSH_WSC_Event"
  -- Public because of transferring data from outside the script
::Routine FireTheEvent Public
  WSCEvent = fireEvent("WSH_WSC_Event")
  SAY "Event was fired"
  -- Routine to write the value of Property2. "WSHPROPERTY" ensures that
::Routine propertyPut Public -- the property is global in scope.
  use arg NewProperty2Value
  Return Value("Property2value",NewProperty2Value,"WSHPROPERTY")
::Routine propertyGet Public -- Retrieves the current value of Property2
  return Value("Property2value",,"WSHPROPERTY")
  -- Sets the number of kilometers and calculates and retrieves the
::Routine km2seamiles Public -- number of seamiles
  use arg km
  return km/1.852
  -- Sets the number of seamiles and calculates and retrieves the number

```

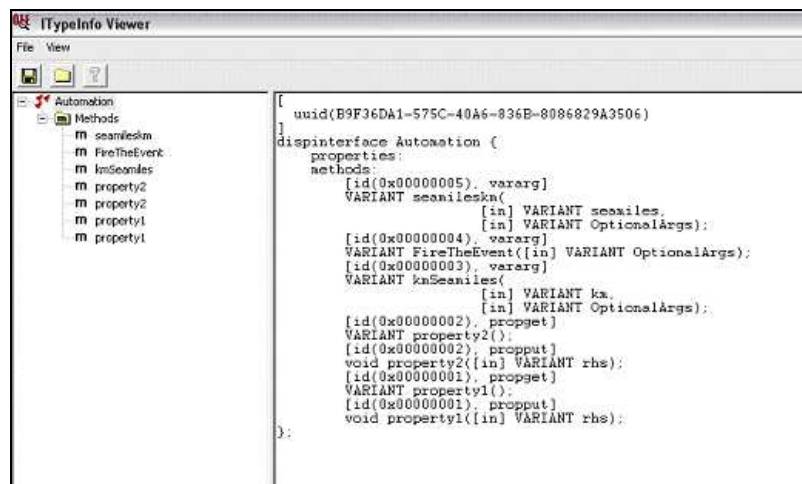
```

::Routine seamiles2km Public -- of kilometers
  use arg seamiles
  return seamiles*1.852
]]>
</script>
</component>
</package>

```

**Code 57:** WSH\_WSC\_Test.wsc<sup>434</sup>

Figure 59 illustrates the new generated functions in the OLE/COM Object Viewer.



**Figure 59:** ITypeInfo Viewer of the OLE/COM Object Viewer<sup>435 436</sup>

#### 13.12.2.4. Accessing the Functions of the Component

The following Object Rexx script code 58 accesses the methods and properties that are exposed. Important is that the new component is instantiated.

```

-----
-- WSH_WSC_Test.rex --
-----

-- Instantiate the WSH_WSC_Test.WSHWSC object
WSCComponent = .OLEObject~New("WSH_WSC_Test.WSHWSC")
-- Access the km2seamiles and seamiles2km methods and hand over of the
-- arguments "17" and "34"
SAY "17 kilometers corresponds to" WSCComponent~km2seamiles(17) "seamiles"
SAY "34 seamiles corresponds to" WSCComponent~seamiles2km(34) "kilometers"
SAY -- blank line
SAY wsccomponent~property1 -- Access the value of property1

```

<sup>434</sup> Modelled after [IBM01,p499ff] and [Ku02,20ff]

<sup>435</sup> c.p. 2.6.2.

<sup>436</sup> Can be downloaded from the Microsoft Homepage c.p. 2.6.2.

```

-- Shows the default value of Property2
SAY "This is the default value of Property2: " wsccomponent~property2
-- Sets a new value to Property2
wsccomponent~property2 = "NewValueOfProperty2"
-- Shows the current value of Property2
SAY "This is the new value of Property2: " wsccomponent~property2
SAY -- blank line
-- The FireTheEvent method is called to fire the Event
WSCComponent~FireTheEvent

```

**Code 58: WSH\_WSC\_TEST.rex**

### 13.12.2.5. Remote Instantiation of a Script Component

It is possible to remote instantiate a script component from another machine via a network. A script on a local machine accesses a component, which is stored and registered on a remote machine. The component's remotable attribute of the `<registration>` element must be set to "true" [MLWSHdb].

Prerequisite is that the script component is registered on the remote machine. It is necessary to register this component on the local machine where the script resides, which accesses this component. Therefore there must be created some registry entries on the local machine. There must be created this new key: `HKEY_CLASSES_ROOT\componentProgID`. As `componentProgID` write the `ProgID` of the component. In this example it is the `ProgID` `WSH_WSC_Test.WSHWSC`. Under this key a further key with the name `CLSID` must be generated. The value of the key `CLSID` is the `CLSID` of the component. In this example the `CLSID` is `{1C123B91-4F3E-4299-9064-26AC1F980C8A}`. Insert the `CLSID` with the brackets [MLWSHdc].

Code 58 is the script component that is located on the remote machine. The difference to code 56 is that here the line `remotable="true">` is added (this is necessary for remote instantiation) and the `propertyPut` and `propertyGet` routines are handled in another kind (this is not necessary for remote instantiation, but it shows another way to expose the properties that also works if the WSC is used on the same machine). Therefore, the Local Environment object (`.LOCAL`) is used. It is a directory of process-specific objects which are always available [IBM01,p294]. The default value of property 2 is hand over with `.local~Property2value = "Defaultvalue"`. The line `.local~property2value = arg(1)` defines an

entry "property2value" in the "LOCAL" directory of the runtime environment. `arg(1)` stores the first received argument. The phrase `IF .local~HasEntry("Property2Value") THEN RETURN .property2value ELSE RETURN .NIL` offers the value of the property [Fla03d]. `HasEntry("Property2Value")` is a method of the `DIRECTORY` class and gives back true if the directory contains an entry with the description "Property2Value" [IBM01,p129]. The `.NIL` object is an object which describes the nonexistence of an object [IBM01,p294].

```
<?xml version="1.0"?>
<?component error="true" debug="true" ?>
<package>
<component>
<comment> Part 1 Registration</comment>
<registration
  progid="WSH_WSC_Test.WSHWSC"
  description="Demonstration of a WSC with Object Rexx"
  version="1.0"
  clsid="{1C123B91-4F3E-4299-9064-26AC1F980C8A}"
  remotable="true">
<script language="Object Rexx"><![CDATA[
::Routine Register Public
  Shell = .OLEObject~New("WScript.Shell")
  Typelib = .OLEObject~New("Scriptlet.TypeLib")
  Shell~Popup("Start of registration")
  Typelib~AddURL("WSH_WSC_Test.wsc")
  Typelib~Doc = "WSH_WSC_TypeLib"
  Typelib~Path= "WSH_WSC_Test.tlb"
  Typelib~Name = "WSH_WSC_TypeLib"
  Typelib~MajorVersion = 1
  Typelib~MinorVersion = 0
  Typelib~GUID = "{CF53943D-428D-48f9-B8DD-5B0C4B850D9D}"
  Typelib~Write()
  Typelib~Reset()
  Shell~Popup("End of registering")
::Routine Unregister Public
  Shell = .OLEObject~New("WScript.Shell")
  Shell~Popup("Ending of registration")
]]></script>
</registration>
<comment> Part 2: Exposing functions</comment>
<public>
<property name="property1">
</property>
<property name="property2">
  <get internalName = "propertyGet" />
```

```

    <put internalName = "propertyPut" />
  </property>
  <method name="km2Seamiles">
    <PARAMETER name="km"/>
  </method>
  <method name="FireTheEvent">
  </method>
  <method name="seamiles2km">
    <PARAMETER name="seamiles"/>
  </method>
  <event name="WSH_WSC_Event" >
  </event>
</public>
<comment> Part3: Script Code </comment>
<script language="Object Rexx">
<![CDATA[
property1 = "This is the content of property1" -- Value of Property1
.local~Property2value = "Defaultvalue" -- Defaultvalue of Property2
    -- Routine of the"FireTheEvent"method to fire the event "WSH_WSC_Event"
    -- Public because of transferring data from outside the script
::Routine FireTheEvent Public
  WSCEvent = fireEvent("WSH_WSC_Event")
  SAY "Event was fired"
::Routine propertyPut Public -- Routine to write the value of Property2.
  .local~property2value = arg(1)
::Routine propertyGet Public -- Retrieves the current value of Property2
  IF .local~HasEntry("Property2Value")
  THEN return .Property2value
  ELSE return .NIL
    -- Sets the number of kilometers and calculates and retrieves the
::Routine km2seamiles Public -- number of seamiles
  use arg km
  return km/1.852
    -- Sets the number of seamiles and calculates and retrieves the number
::Routine seamiles2km Public -- of kilometers
  use arg seamiles
  return seamiles*1.852
]]>
</script>
</component>
</package>

```

**Code 59:**      **WSH\_WSC\_Remote.wsc**<sup>437</sup>

On the local machine is the OtherScript 12 located that accesses a method of the component WSH\_WSC\_Test.WSHWSC that is located on the machine with the name

<sup>437</sup> c.p. code 56



FHKCN. OtherScript 12 is written with Visual Basic Script code. This script calculates the seamiles which corresponds to 17 kilometers.

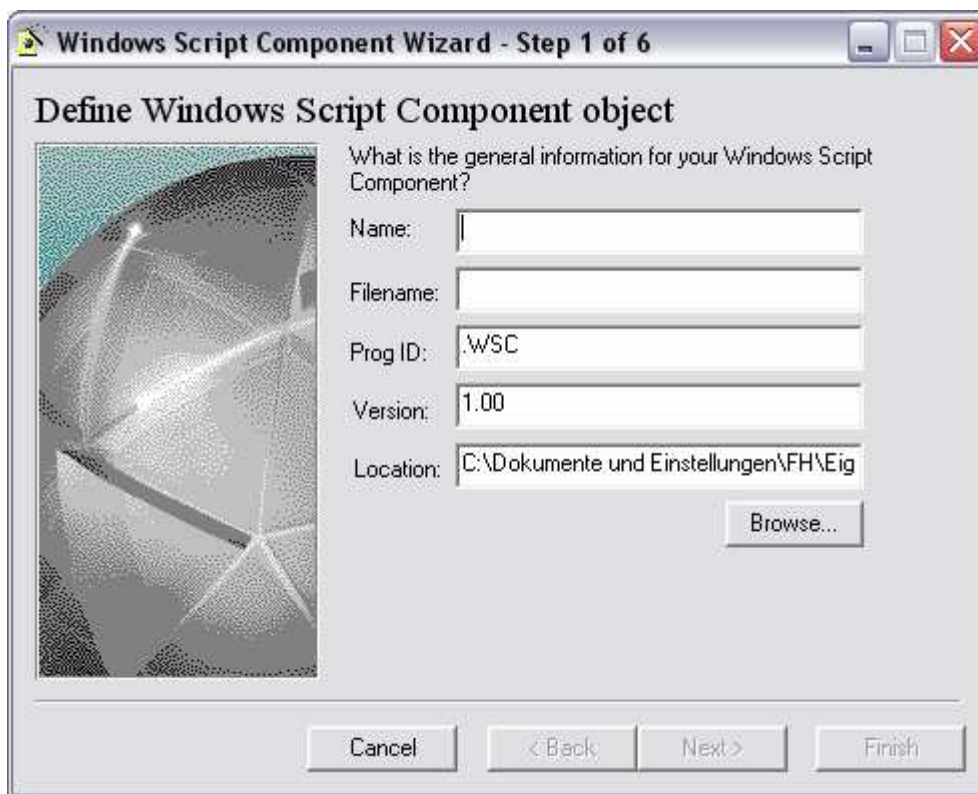
```
Set newS = CreateObject("WSH_WSC_Test.WSHWSC","FHKCN")
WScript.Echo "17 kilometers corresponds to" &newS.km2seamiles(17) &"seamiles"
```

**OtherScript 12: RemoteWSC.vbs**

### 13.12.2.6. Windows Script Component Wizard

The Script Component Wizard helps to generate a script component file, create the registration information, arranges the type of interface handler and helps to generate the events, methods and properties [MLWSHcy].

The wizard can be downloaded from the Microsoft Homepage<sup>438</sup>. Figure 60 shows the user interface of the Windows Script Component Wizard.



**Figure 60: Microsoft Windows Script Component Wizard.**<sup>439</sup>

<sup>438</sup><http://msdn.microsoft.com/downloads/default.asp?URL=/downloads/sample.asp?url=/MSDN-FILES/027/001/788/msdncompositedoc.xml>

<sup>439</sup> Can be downloaded from the Microsoft Homepage

## 14. MS.NET

Microsoft .NET (MS.NET or only called .NET) enables the usage and the generating of XML Web Services, XML-based applications and processes. The basics of MS.NET, the fields of application and the possibility to use MS.NET functions with Object Rexx are explained. This section discusses the items smart devices, Web Services, NET Framework, programming with the .NET Framework, building applications as well as Object Rexx with MS.NET.

MS.NET is a programming model that is developed for XML Web Services. It offers a multi-device support and a multi-language environment for developing and using XML Web Services [MS02a].

.NET can be downloaded from the Microsoft Homepage<sup>440</sup>. It enables the development and the usage of XML Web Services [MS02a].

Microsoft claims that MS.NET makes it possible to distribute computing power over several devices and enables the updating and the reusing of XML Web Services [MS02a].

MS.NET connects devices, systems and information. Applications can be created with developer tools like Microsoft Visual Studio .NET. Servers like Microsoft SQL Server, Microsoft Windows 2000 and Microsoft BizTalk Servers or Client Software like Microsoft Office XP, Windows CE or Windows XP enable the usage of XML applications and Web Services [MS02b].

The .NET Framework contains the class libraries and the Common Language Runtime<sup>441</sup>. The .NET Framework started in January 2002. There are compilers available for over 20 languages. Object Rexx is not part of them, because there is only an interpreter available for Object Rexx [Doe03b]. This makes it possible for developers to use the language that is suited for the respective purpose. The .NET Framework is and component-based [MS02b].

---

<sup>440</sup> <http://msdn.microsoft.com/library/default.asp?url=/downloads/list/netdevframework.asp>

<sup>441</sup> c.p. 14.3.1.

XML is an open standard technology for data transformation and exchange that enables integration and operability of services and applications. SOAP<sup>442</sup> and XML offers system interoperability and an administration of services and applications for .NET [MS02a].

For Pocket PCs the .NET Compact Framework is available [MS02e].

## **14.1. Smart Devices**

Smart devices are, for example handheld computers, tablet PCs, game consoles, smart phones, workstations, laptops or PCs [MS02a]. “Smart” imports the ability to work with digitally information [Ge00].

They enable the user to access data anywhere and anytime and they use the profile of the user. They can connect to other devices and interchange data over a network. They use XML, UDDI and SOAP [MS02a].

## **14.2. Web Services**

This section describes Web Services, the Simple Object Access Protocol, the Web Services Description Language, the Universal Discovery Description and Integration as well as the difference of Web Site and XML Web Service

Web Services are reusable and small applications. They are structured in XML. They facilitate the data exchange over the Internet. Peer2Peer connections are possible [MS03m].

XML Web Services enable the communication of programs that contain different languages and that are located on different platforms. They are running with standard Web protocols like TCP/IP, HTTP and XML [MS01c].

Supplier of Web Service technology are except to Microsoft for example IBM with IBM WebSphere SDK for Web Services (testing and creating Java-based Web services), IBM WebSphere Application Server (Web service applications and other e-business applications; embedded in IBM WebSphere SDK for Web Services) and IBM WebSphere Studio (deployment and development environment for building,

---

<sup>442</sup> c.p. 14.2.1.

testing and deploying on demand e-business applications) [IBM03], Sun with Sun Open Net Environment with server-software and development tools for e-commerce applications and services [BI02] or Apple with .Mac which provides an e-mail service and personnel services [Mue02].

### **14.2.1. Simple Object Access Protocol**

XML Services use the protocol SOAP (Simple Object Access Protocol) to offer their features. XML messages are exchanged.

SOAP can be compared with RPC, but HTTP is used as protocol and XML for coding (arguments, methods, returnvalues) [Fla03e]. Thereby a SOAP-Envelope includes the SOAP-Header (contains application specific information about the SOAP message and is optional [WS03]) and the SOAP-Body [MS01c], [Tu02,p231f]

### **14.2.2. Web Services Description Language**

The interfaces of the XML Web Services are described in WSDL (Web Services Description Language) document. Data types can be defined, messages are exchanged among so-called end points, these messages are specified, messages are bundled to operations, the connection to a SOAP message is made and the service is linked to an URL [MS01c], [Tu02,p234].

### **14.2.3. Universal Discovery Description and Integration**

UDDI (Universal Discovery Description and Integration) enables the registration of XML Web Services to simplify their discovering. It offers an infrastructure for the exchange of information about the Web Service. Therefore so-called white papers (information about the supplier of the service), yellow papers (categorization of the service) and green papers (technical information) are used. This infrastructure is also a Web Service [MS01c], [Tu02,p235].

### **14.2.4. Difference of Web Site and XML Web Service**

A Web Site is viewed with a browser and consists of pictures of data.

The XML Web Service can be utilized or joined with a software application or with another service. It communicates over a network like the Internet and uses therefore

standards like SOAP and XML [MS02a]. Figure 61 illustrates the differences of Web Site and Web Service.



**Figure 61:** Differences between Web Site and Web Service<sup>443</sup>

## 14.3. NET Framework

This section discusses the basics of .NET. The items Common Language Runtime, assembly, metadata, Cross-Language Interoperability and .NET Framework Class Library are explained.

The .NET Framework offers a platform in a distributed environment.

The .NET Framework consists of two core elements: the .NET Framework class library and the Common Language Runtime [MLNETa].

### 14.3.1. Common Language Runtime<sup>444</sup>

The Common Language Runtime organizes memory, thread and process management and the language integration [MS02b]. It is distributed via MS Windows Updates via the Internet since march 2003 [Fla03e]<sup>445</sup>.

<sup>443</sup> Taken from [MS02a]

<sup>444</sup> [MLNETa]

<sup>445</sup> E.g. MSIE->Extras->Windows Update

The Common Language Runtime makes it possible that objects programmed in different languages can correspond with each other. A Common Type System enables this cross-language integration [MLNETb].

So-called managed code is code built with a language compiler that addresses the runtime. It supports a model for component communication, debugging services, profiling services, cross-language integration, cross-language exception handling, and versioning. [MLNETb].

Managed data means that the runtime administrates objects. Thereby the references and releases of objects are organized by it. Managed and unmanaged data can be used alone or both together in a .NET Framework application [MLNETb].

The developers can generate its source code with different languages like Visual Basic or C#. To use the Common Language Runtime, code must be compiled with a language compiler [MLNETcj].

This compiler translates the managed code into Microsoft Intermediate Language (MSIL). MSIL is a CPU-independent set of commands that can be converted to native code. MSIL provides commands for calling, storing, loading and initializing methods. It is also possible to use instructions for logical and arithmetic operations, exception handling, direct memory access or control flow. A just-in-time (JIT) compiler converts the MSIL to CPU-specific code.

When MSIL is created metadata is also created. Both are contained in a portable executable (PE) file [MLNETck].

Metadata is offered by the language compilers and contains information about the references, types and members in the code. Metadata determinates run-time context boundaries, invokes methods, offers location of classes and loads them. So-called portable executable (PE) file includes metadata [MLNETb]. PE files are used for files to be linked together to form executable programs, and for executable programs [MLNETcl].

The metadata of the managed components stores information about them and the resources. State data and registration information are not saved in the registry. This information is stored as metadata [MLNETb].

The perception of features of the runtime can vary from environment to environment in which a program is written [MLNETb].

Features of the Common Language Runtime:

- Organization of compilation, code safety check, memory, thread and code execution and other system services
- Interoperability among unmanaged and managed code enables the further usage of COM components and DLLs.
- Server-side applications like Microsoft Internet Information Services (IIS) and Microsoft SQL Server can host the runtime
- Just-in-time (JIT) compiling makes it possible to run managed code in the native machine language of the system on which it is started. The memory manager turns away fragmented memory.

### **14.3.2. The Assembly**

The “Assembly” contains resources and types and it offers information for the Common Language Runtime<sup>446</sup>. Assemblies are the basis for .NET Framework applications with version control, reuse, security permissions, deployment and activation scoping [MLNETc]. This section discusses the features of an assembly, dynamic and static assemblies and the parts of static assemblies.

Features of an assembly [MLNETd]:

- It offers a type boundary with the type identity.
- Microsoft Intermediate Language (MSIL)<sup>447</sup> code can only be used if it is linked with an assembly
- A deployment unit enables that only these assemblies are present that were initially called.

---

<sup>446</sup> c.p. 14.3.1.

<sup>447</sup> c.p. 14.3.1.

- Allows side-by-side execution (several versions of the same assembly run simultaneously [MLNETe]).
- It offers a reference scope boundary to identify the resources and types, which are outside of the assembly.
- It offers a version boundary to define the version dependencies for any dependent assemblies.
- It offers a security boundary because an assembly is a unit at which permissions are granted and requested

Assemblies can be generated with Common Language Runtime<sup>448</sup> APIs like the Reflection Emit, with tools contained in the .NET Framework SDK or with development tools like the Visual Studio .NET [MLNETd].

Assemblies enable an infrastructure to use several software component versions at the same time (so-called side-by-side execution). Versioning rules can be enforced [MLNETf].

There are dynamic and static assemblies.

Static assemblies are saved on disk in PE files and they can contain resources for the assembly like JPEG files or bitmap files and .NET Framework types (classes and interfaces) [MLNETd].

Dynamic assemblies run directly from the memory and can be saved on disk after execution [MLNETd].

Normally a static assembly includes four parts like in figure 62 illustrated [MLNETg]:

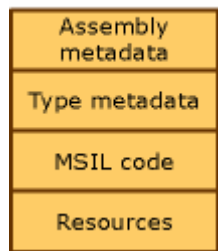
- The assembly manifest (required),
- The Type metadata,
- Microsoft Intermediate Language (MSIL)<sup>449</sup> code to use the types,

---

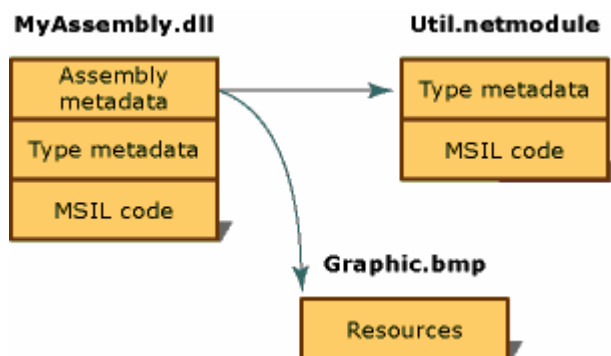
<sup>448</sup> c.p. 14.3.1.



- Resources.

**MyAssembly.dll****Figure 62:** All elements are united in a single file.<sup>450</sup>

It is also possible to store the elements of an assembly in multiple files. A multifile assembly allows optimizing downloading of an application and the combination of different language modules. The different files can be resources like graphic files, compiled code (.netmodule) or other necessary files. The files are only downloaded if they are referenced and they are linked by the assembly manifest [MLNETg]. Figure 63 shows a multifile assembly.

**Figure 63:** Multifile assembly<sup>451</sup>

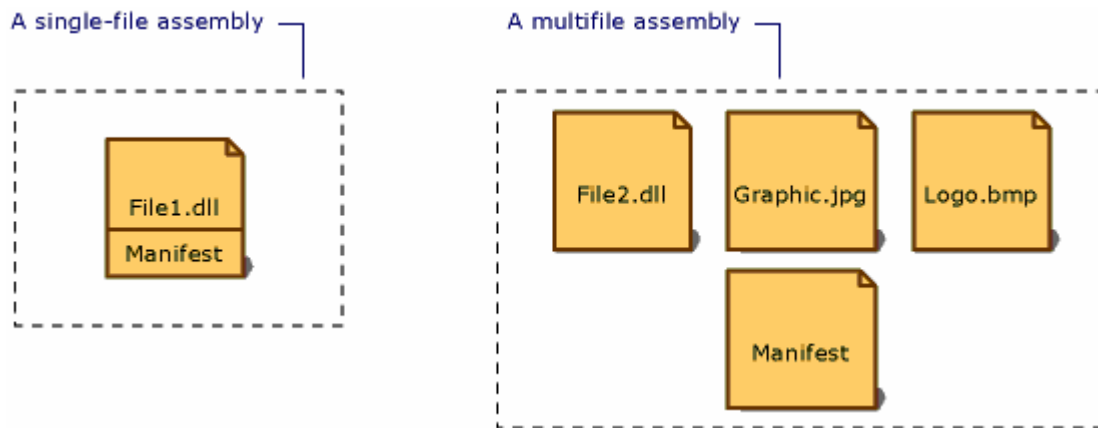
The assembly manifest includes the assembly metadata, which contains information about the relationships of the elements, the metadata with the security identity and the version needs of the assembly. It is saved in a standalone PE file with only assembly manifest information or in a portable executable PE file with Microsoft

<sup>449</sup> c.p. 14.3.1.

<sup>450</sup> Taken from [MLNETg]

<sup>451</sup> Taken from [MLNETg]

Intermediate Language<sup>452</sup> [MLNETh]. Figure 64 illustrates a single file assembly and a multifile assembly.



**Figure 64:** Single-file assembly and multifile assembly<sup>453</sup>

In the case of a single-file assembly the manifest is integrated together with the PE file. In the case of a multifile assembly the manifest can be integrated in one of the PE files or it can be stand-alone [MLNETh].

The assembly manifest lists all files of the assembly, lists all other related assemblies, offers a self-description of the assembly and administrates the references to the resources and types of the assembly. There is information about the identity of the assembly, namely the language or culture the assembly can use, the assembly name, the version number and strong name information (the public key) [MLNETh]. A strong name is a name that consists of the identity of an assembly. It is a text name, version number, culture information. This is increased by a digital signature and by a public key created over the assembly. Assemblies are identical if they have the same strong name [MLNETcn].

A side-by-side assembly must have a version. The version number has the four parts `major.minor.build.revision`. The `major` or `minor` parts must be modified if an assembly is made incompatible with existing versions by a change to the assembly. An assembly that is backward compatible with prior versions is modified only in the `build` or `revision` parts [MLNETcm].

<sup>452</sup> c.p. 14.3.1.

<sup>453</sup> Taken from [MLNETh]

### 14.3.3. Metadata

Metadata offers the possibility that components can communicate because the .NET Framework enables compilers to give additional declarative information into assemblies and modules [MLNETi]. Metadata with its features and PE files are explained.

The metadata is saved in the memory or in a PE file of the Common Language Runtime<sup>454</sup>. If the code is compiled the code is translated to the Microsoft Intermediate Language (MSIL)<sup>455</sup>. The metadata and the code are stored in the same file in separate parts of it. The appropriate metadata is loaded if the code is started. Metadata contains information about the types (members and description), the assembly (identity, related assemblies, exported types and security permissions) and the attributes [MLNETj].

Features of metadata [MLNETj]:

- Microsoft claims that language interoperability enables the generating of any class in any managed language that can be used by the Common Language Runtime<sup>456</sup>.
- Attributes can be exposed. They are a type of metadata in the compiled file that can control the program behaviour at run time.
- Description of .NET languages in a language-neutral kind
- Self-describing files enable modules to communicate with each other. The metadata offers all required information [MLNETj].

Metadata and the common type system together enable cross-language inheritance [MLNETk].

---

<sup>454</sup> c.p. 14.3.1.

<sup>455</sup> c.p. 14.3.1.

<sup>456</sup> c.p. 14.3.1.

### *PE File*

A PE file (portable executable file) consists of the PE header, the MSIL instructions and the Metadata. The PE header contains the address of the entry point and the index of the main sections of the PE file. The code consists of MSIL instructions. The metadata consists of heaps and tables (information about the elements of the program). It offers information about custom attributes and security members and it records data about types to the runtime.

Metadata tokens are used to reference rows of the metadata table [MLNETI].

### **14.3.4. Cross-Language Interoperability**

Content of this section are the cross-language interoperability and the Common Language Specification. Language interoperability is available for the Common Language Runtime<sup>457</sup>. This is enabled by the Common Language Specification (CLS) that defines some rules and features for the language [MLNETm].

Language interoperability makes it possible that one code can communicate with a code that is programmed in another language and the reuse of code is supported. The Common Language Runtime defines a common type system with rules for types of all languages and provides metadata with rules for the management of the information of the types. In this way, it is the basis for language interoperability. This enables the running of multilanguage applications by the runtime [MLNETn].

Microsoft claims the following advantages of language interoperability for managed code [MLNETn]:

- Consistent exception handling across languages,
- With metadata and the Microsoft Intermediate Language (MSIL)<sup>458</sup> for the Common Language Runtime only one environment is necessary for profilers, debugger or other tools,
- Regardless of the language types, objects or methods can be used.

---

<sup>457</sup> c.p. 14.3.1.

<sup>458</sup> c.p. 14.3.1.

Nevertheless, it is possible that the functionality of generated types cannot be fully used by other languages. The problem is that the language compiler uses the metadata and the type system to maintain the own language features and that can differ from other language features. To solve that problem the Common Language Specification is used. It sets the rules for the language features [MLNETn].

### *Common Language Specification*

The Common Language Specification (CLS) consists of a set of language features to enable language interoperability. The CLS rules are a subset of the Common Type System. Components that only apply CLS features in the API are fully available from CLS supporting languages. These are so-called CLS-complaint components. All CLS-compliant languages are able to create verifiable code [MLNETo].

The `CLSCompliantAttribute` function allows characterizing assemblies, modules, types and members as CLS-compliant or as non-CLS-compliant. A CLS-compliant assembly is marked as CLS-compliant. If it is not marked, it is considered as not CLS-compliant. If there is no CLS-compliant attribute to a type, it has the same CLS-compliant attribute as its assembly. If there is no CLS-compliant attribute to a member, it has the same CLS-compliant attribute as its type.

Nevertheless if several parts of the assembly, module or type are not CLS-compliant, assembly, module or type could be CLS-compliant if there is supplied for each non-CLS-compliant member a comparable CLS-compliant alternative member and if all parts which are CLS-compliant or non-CLS-compliant are manifested as such a part. Languages that can access all features that are supplied by CLS-compliant libraries can be used with languages, which are called CLS-compliant consumer tools. Languages, which enable to use types that are specified in CLS-compliant libraries, are called CLS-compliant extender tools [MLNETp].

## **14.3.5. .NET Framework Class Library**

The .NET Framework Class Library is object-orientated, offers the possibility for third-party components to be used with .NET Framework classes and it is a compilation of

reusable types, which are included with the Common Language Runtime<sup>459</sup> [MLNETa].

Functions of the .NET Framework types:

- Performance of I/O
- Invocation of security checks of the .NET Framework
- Information of loaded types
- Summarize data structures
- Description of base data exceptions and types
- Supply of server-controlled and a client-side GUI and data access

It is possible to use the interfaces and the classes inclusive derivation of the classes of the .NET Framework.

The .NET Framework naming schema is divided into two parts, the namespace name and the type name. The namespace name consists of all to the rightmost dot and all what is right of that dot is the type name. An example is: `System.Collections.ArrayList`. Here `System.Collections` is the name of the namespace and `ArrayList` is the name of the type. Library developer should name their namespaces in the following manner: `CompanyName.TechnologyName` like `Microsoft.Word`. It is possible that one assembly includes types from several namespaces. One namespace can be split into several assemblies.

The root namespace is the `System` namespace, which contains classes like for example `Object`, `Char`, `Int32`, `String`, `Array` and `Byte`, which are the fundamental data types for all applications. Furthermore, this namespace includes approximately 100 classes and many second-level namespaces [MLNETq].

---

<sup>459</sup> c.p. 14.3.1.

## **14.4. Programming with the .NET Framework**

This section describes some fields of application of the .NET Framework. Here is information about the key programming concepts discussed. The following items are described: Microsoft .NET Passport, .NET Remoting, accessing the Internet, Active Directory-Components, CodeDOM, components development, developing world-ready applications, asynchronous calls, creation of messaging components, Windows Management Instrumentation, processing transactions, security items, system monitoring components and ADO.NET.

### **14.4.1. ADO.NET**

Content of this section is an introduction to ADO.NET, the objects of ADO.NET and the ADO.NET architecture. ADO.NET means ActiveX Data Objects for .NET Framework. It offers classes that allow data access to application data, XML and relational data. Microsoft claims that it is suited for the creation of data sharing and distributed applications and for middle-tier business objects and front-end database clients [MLNETr].

Microsoft also says that with ADO.NET consistent data access is possible and data-sharing consumer applications can be used to manage data [MLNETs].

This data access technology is based on the .NET Framework. In this way it offers communication with a database, a common data representation, integration with XML and disconnected data architecture. ADO.NET and ADO can exist with each other and the programming model of ADO.NET is similar to ADO. ADO.NET enables the usage of disconnected, n-tier programming environments and supports XML [MLNETt].

ADO.NET enables scalable data access and platform interoperability. With XML ADO.NET can remote data among clients and tiers. XML supports ADO.NET by hierarchical queries, data transformation and by validation [Ro01].

#### **14.4.1.1. Objects of ADO.NET**

The `DataReader` object offers a quick, read-only, forward-only access to query results. With the `DataSet` object, an in-memory relational representation of data is available and a common, completely disconnected data representation is possible.

The `DataAdapter` object is a connection between the data source and the `DataSet` object. ADO.NET enables the return of results in their native data type [Ro01].

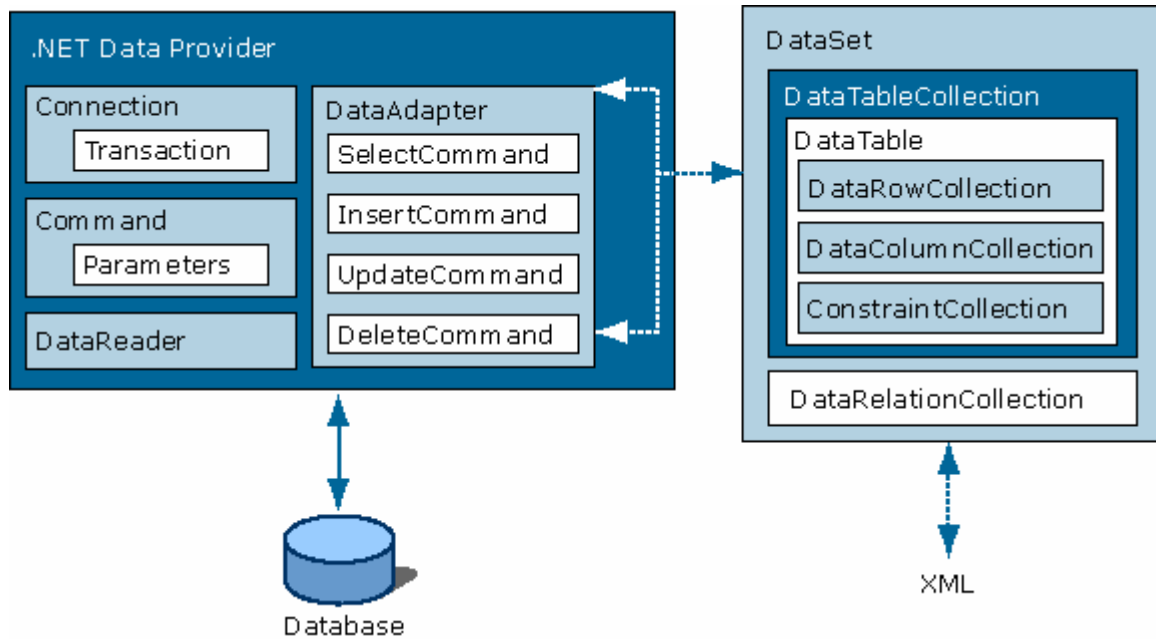
#### **14.4.1.2. ADO.NET Architecture**

The ADO.NET architecture consists of two components. These are the .NET Data Provider and the Data Set. This section explains the ADO.NET architecture.

The .NET Data Provider manipulates and accesses the data. This is a read-only and forward-only data access. With the `Command` object the data can be managed. The `Connection` object enables the connection to the data source. The `DataReader` offers from the data source a high-performance data stream. `DataAdapter` is the connection among data source and `DataSet` object. The .NET Framework uses the OLE DB .NET Data Provider and the SQL Server .NET Data Provider. It is possible to create .NET Data Provider for any data source.

The `DataSet` is the main component. It is possible to use it with XML data, with managed data local to the application or it can be used with different and several data sources. The `DataSet` includes the so-called `DataTableCollection` that contains some objects to manage row, column, relation and constraint information of data. These objects cannot be moved via XML web services. The `DataSet` is used if the data is locally cached in the application, if data processing is expanded and an open connection to the data source is not necessary, if remote data is used from a XML web service or among tiers or if there is a dynamically data communication. If this is not the case Microsoft claims that the performance can be increased if `DataReader` is used [MLNETu]. Figure 65 illustrates the ADO.NET architecture.





**Figure 65:** ADO.NET architecture<sup>460</sup>

## 14.4.2. .NET Remoting

.NET Remoting is a generic interprocessing system. It can be used to generate XML web services. This section provides information about .NET Remoting and its architecture.

All security features are available for .NET Remoting if an HTTP-based application that hosts in IIS (Internet Information Services) is used. Scalability is possible.

Features of .NET Remoting:

- Object lifetimes and activation are checked directly
- Through taking part in the communication process the wanted functionality can be generated
- Support of third-party protocols and channels
- Presents an object by a reference and gives it back to a definite object in a definite application domain

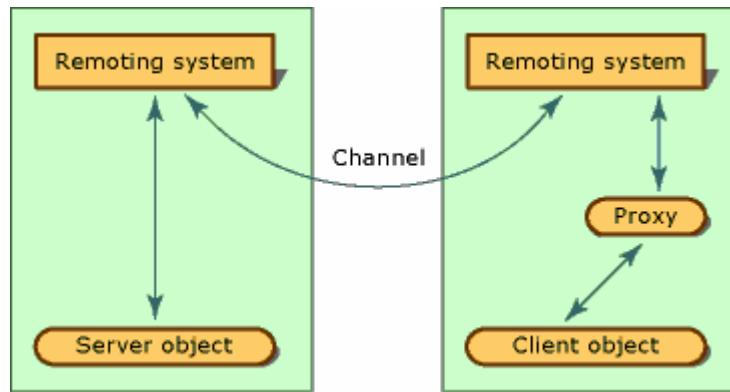
<sup>460</sup> Taken from [MLNETu]

- Services of each type of application domain are available
- A managed-code type-system integrity in binary formatted communications is kept [MLNETv]

Microsoft claims that through .NET Remoting different applications can interact with one another independent of their location and operation system. It is a generic system of interprocess communication. So-called formatters encrypt and decrypt the messages before transmitting them. An object that can be handed over by value can be automatically handed over among applications in variant domains or machines. It is nearly always possible to interrupt and change the communication process. Transaction of objects in different application domains and processes is available for different object generating modes, object lifetimes schemes, serialization formats or transportation protocols.

#### *Architecture of .NET Remoting*

The core component of the remoting is the object reference to interact among server objects and clients. With the function `New` a new instance of the remote object is generated, the client gets a reference to the object and the methods can be called. So-called proxy objects are stand-in objects, which are generated if the client creates an instance of the remote type. For the client the proxy object is like the original object. A call on the proxy object is routed by the remoting system to the server process where the call is worked on and then the return value is given back to the client over the proxy. So-called transport channels have the function of a particular protocol for sending the data and as technology for network connections [MLNETw]. Figure 66 illustrates the remoting process [MLNETw].



**Figure 66:** Remotingprocess<sup>461</sup>

### 14.4.3. Accessing the Internet

With the .NET Framework an administrated implementation of Internet services is enabled.

The .NET Framework uses a Uniform Resource Identifier (URI) to locate an Internet resource. The URI contains the scheme identifier (identification of the communication protocol), the server identifier (TCP address or DNS host name), the path identifier (location on the server) and possibly a query string (information from client to the server).

The .NET Framework offers the `Uri` class (URI of the Internet resource), the `WebResponse` class (container for incoming response) and the `WebRequest` class (request for the resource). The `WebClient` class (requires the `WebRequest` class) offers methods for up- and downloading of data. The classes `TCPListener`, `TCPClient` and `UDPCClient` of the `System.Net.Sockets-Namespace` enable functions for creating connections [MLNETx].

Pluggable protocols and an administrated implementation of the Windows socket interface can be used [MLNETy].

<sup>461</sup> Taken from [MLNETw]

#### 14.4.4. Active Directory-Components

ADSI (Active Directory Services Interfaces) is an interface that makes the communication of applications with directories on a network possible. This section presents the features and components of ADSI.

Features of ADSI [MLNETz]:

- Protocols enable the usage of a single application programming interface (API) to be carried out on several directory systems.
- Construction of applications which access printers, manage user accounts or back up databases.
- Insertion of directory information in databases.
- The Active Directory tree structure enables the management of a hierarchical, single structure for network configurations.
- To handle different directories only one log on is necessary.
- Querying for directory systems is supplied with LDAP (Lightweight Directory Access Protocol) and SQL.

The two component classes `DirectorySearcher` and `DirectoryEntry` of the namespace `System.DirectoryServices` are supplied by the .NET Framework for the Active Directory Services Interfaces (ADSI) technology. ADSI is a technology to administrate resources on a network [MLNETaa].

An Active Directory hierarchy can be searched and queries can be made with the `DirectorySearcher` class by using the Lightweight Directory Access Protocol.

The `DirectoryEntry` component supports administrative task like monitoring or changing of properties. Therefore, this component can be associated with an object in the directory.

To use ADSI with the components `DirectorySearcher` and `DirectoryEntry` the ADSI runtime or ADSI SDK are necessary. On Windows NT version 5.0,

Windows 2000 and Windows XP ADSI is installed. For other Windows platforms the ADSI SDK can be downloaded from the Microsoft Homepage<sup>462</sup> [MLNETz].

Microsoft says that the ADSI is a Windows directory service which decrements the amount of namespaces and directories that the developer must take in consideration. The Active Directory is organized like a hierarchical tree. On the one hand, a directory service is an information source and, on the other hand, it offers the information to the user. A so-called Active Directory schema contains information about attributes for directory objects, which is used for searches for members, and they contain information about network nodes. The Active Directory schema is saved in the Active Directory hierarchy. Schemas are used by the `DirectoryEntry` and `DirectorySearcher` components to get information.

An alterable ADSI COM object is returned if a valid directory path is offered to the `DirectoryEntry` component. The `DirectoryEntry` component allows including new nodes to the hierarchy, the properties of a node from a hierarchy can be managed and an object or service can be found in the Active Directory hierarchy.

With ADSI it is possible to access all directory protocols (so-called service provider) with a single interface to manage the directories contents. With a service provider, objects with associated behaviour and data are usable [MLNETab].

### **14.4.5. CodeDOM**

The CodeDOM (Code Document Object Model) mechanism of the .NET Framework offers the possibility to have output of source code in several programming languages at run time.

Therefore, the CodeDOM provides the architecture, classes and interfaces to manifest the structure of source code. It is possible to use an external compiler for the compilation of the source code. A so-called CodeDOM graph or tree shows the structure of the source code and is created with CodeDOM elements. Elements that describe code elements are provided by the `System.CodeDom` namespace.

---

<sup>462</sup> <http://www.microsoft.com/NTServer/nts/downloads/other/ADSI25/default.asp>

Classes, which support compiling created code at run time, are defined with the `System.CodeDOM.Compiler` namespace [MLNETac].

CodeDOM can be used for dynamic compilation, which means code compilation in single or multiple languages. It can also be used for templated code creation. This is code creation for code wizards, designers, ASP.NET, XML-based Web services or other code-emitting mechanism. The core types of elements of programming languages that can be used with the Common Language Runtime<sup>463</sup> can be used by CodeDOM [MLNETad].

Source code graphs helps to create source code in supported programming languages with so-called code generators. With code compilers a source code can be created in a supported language [MLNETae].

#### **14.4.6. Components Development**

This section discusses components in the .NET Framework. A component in the .NET Framework is a class that is used with the `System.ComponentModel.IComponent` interface or it is a class that comes indirect or direct from a class using this interface. A component must be created with a Common Language Specification<sup>464</sup> compliant language and all members must be CLS-compliant [MLNETaf].

A characteristic of a component is that it can communicate with other objects and it can be used again. For .NET Framework components, further features like design-time support and control over external resources are available.

.NET Framework enables classes that are components to be handled with an rapid application development (RAD) environment like Visual Studio .NET and enables in this way a design-time support.

It is possible that a component is hosted by a container and can receive services.

---

<sup>463</sup> c.p. 14.3.1.

<sup>464</sup> c.p. 13.3.4.

There are nonremotable and remotable components, remotable components are marshalled<sup>465</sup> by value or by reference. Marshalling by reference means that a proxy is generated for the interaction. The base class is in this case `System.ComponentModel.Component`. This is recommended for components which exist as single instance or that summarize system resources. Marshalling by value means that a serialized copy of the object is transmitted and it is recommended for components, which simply hold state. The base class is here `System.ComponentModel.MarshalByValueComponent`. A nonremotable component is used directly with the `IComponent` interface.

There are two base classes for controls in the .NET Framework from which all controls derive. The one base class is the ASP.NET server control and the other is the client-side Windows Forms control. Controls are components and make user-interface (UI) skills possible.

A container could include only one or multiple components and it is a class using the `System.ComponentModel.IContainer` interface [MLNETag].

To display a control and its members at design time so-called design-time attributes are necessary to show information for visual design tools [MLNETah].

The .NET Framework enables a developer to license its controls to preserve the intellectual property [MLNETco].

### **14.4.7. Developing World-Ready Applications**

This section defines the term “World-ready applications”. So-called “World-ready applications” are developed in three steps: Globalization, Localizability and Localization.

An application that is globalized is language-neutral and culture-neutral and can use regional data for all users and localized user interfaces [MLNETai].

In the *Globalization* part the executable code of the application is created [MLNETaj].

---

<sup>465</sup> c.p. 2.4.3.

The second step *Localizability* separates the parts of the application, which need translation from the rest of the application's code [MLNETai].

That means that the executable code of the application is separated from its resources. Normally it is not necessary to vary the source code while the localization [MLNETaj].

*Localization* means that the application is customized for the particular regions or cultures [MLNETai]. Here the user interface is translated [MLNETaj].

An application which is really global is language- and culture-neutral. Microsoft claims that “World-ready applications” help to support new cultures, because the application can be used in more cultures [MLNETaj].

The executed code is stored in the main assembly of the application.

Prerequisite for the Localization is the Localizability. The Common Language Runtime<sup>466</sup> makes the separation of resources and executable code possible [MLNETak].

### **14.4.8. Asynchronous Calls**

This chapter describes so-called asynchronous calls that are supported by MS.NET.

Thereby a .NET class method is called during the runtime of the program, until blocking or calling away or waiting for the call to complete (if callback is not provided), or until the particular callback is carried out (if callback is provided) [MLNETaj].

Asynchronous calls are used with .NET in the following cases:

- Poll Completed (offers the `IAsyncResult.IsCompleted` property)
- Use Callbacks (offers the callback delegate at the start of the asynchronous call)
- Begin Invoke, Wait Handle, End Invoke (waiting for `IAsyncResult`)

---

<sup>466</sup> c.p. 14.3.1.



- Begin Invoke, End Invoke (early finishing of the operation)

With .NET there are two parts for an asynchronous operation. The first part is responsible for the input from the client that starts the asynchronous operation. This part returns a waitable object that is used by the server to preserve any state related with asynchronous operation. The second part offers the results for the client of the operation by supplying the waitable object [MLNETam].

### **14.4.9. Creation of Messaging Components**

The `MessageQueue` component makes it possible to generate, delete and explore queues and to transmit and receive messages. Content of this section are the features of messages and messaging, basic knowledge of messaging and types of queues.

The developer can integrate message-based communication in the application [MLNETan].

To use this technology Message Queuing must be installed on the client machine<sup>467</sup> [MLNETao].

It offers a mechanism for interaction among components of a server-based application.

Features of messages and messaging [MLNETao]:

- Messages stay in a queue until they are guaranteed processed
- The message queuing uses Windows security for access control, encoding, auditing and authentication
- Messages have an offline capability. That means that they can be transmitted to temporary queues and stay there until they are sent.
- Transactional messaging allows bundling of messages into one single transaction
- Message prioritization is possible

---

<sup>467</sup> <http://www.microsoft.com/msmq/>

#### 14.4.9.1. Basic Knowledge of Messaging

A *message queue* is like a container that stores messages until they are transmitted. The so-called *Message Queuing* enables messaging between Microsoft Windows machines. Different computers that are able to send messages to each other are called a *Message Queuing network*. The single computer in such a network is called *site* and the computers are connected with so-called *site links*. The function of *routing servers* is to determine the most efficient and fastest way for sending messages [MLNETap]. Figure 67 illustrates the message routing between sites.

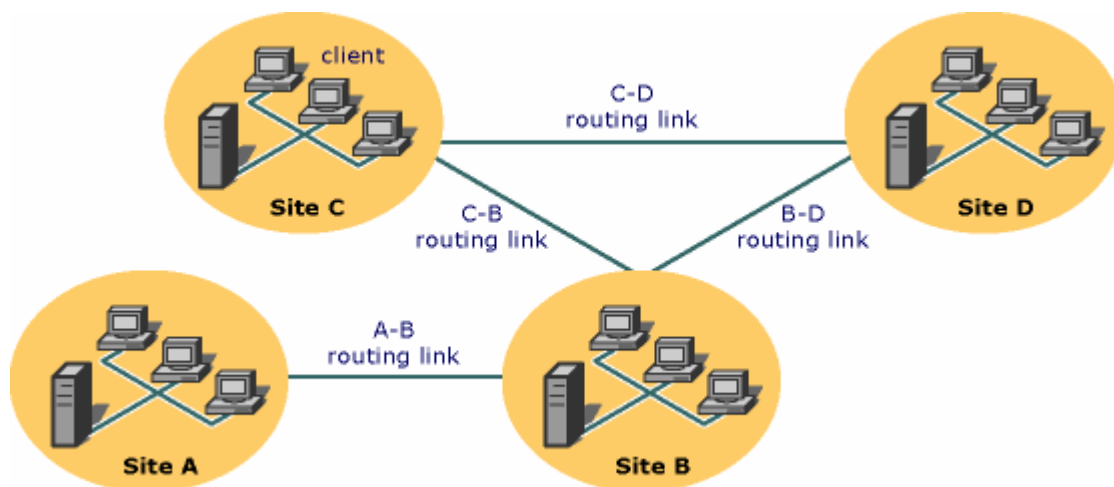


Figure 67: Messagerouting between sites<sup>468</sup>

#### 14.4.9.2. Types of Queues

There are so-called user-created queues and system queues. The user-created queues include private queues (only accessible by the local machine), administration queues (confirm the receive of a message), response queues (response message which is sent back to the transmitting application) and public queues (accessible by all sites). The system queues include private system queues (private queues needed for process messaging), report queues (show the route of a message or test messages), dead-letter queues (copies of not sent or expired messages) and journal queues (supplies copies of sent and removed messages).

The messages are sent asynchronously [MLNETap].

<sup>468</sup> Taken from [MLNETap]

### **14.4.10. Windows Management Instrumentation**

The Windows Management Instrumentation (WMI) offers system management services for the Microsoft Windows operating systems [MLNETaq].

An administrator can use the WMI for the following features:

- Monitor application because of errors
- Configuration and management of applications
- Implementation of remote or local management operations
- Discover errors and bottlenecks
- Query of the data of the application

The tiers of the WMI architecture are the clients (software components that implements operations with WMI), the Object Manager (agent among clients and providers) and the providers (give back live data to the client application, call methods from the client, associate the client to the administrated infrastructure). For older Windows versions, WMI has to be installed with the .NET Framework [MLNETar].

### **14.4.11. Processing Transactions**

Transaction processing systems enables a data-orientated system to be updated only if all operations are fulfilled. Microsoft claims that this technology can be used for exchange.

Therefore, some related operations are combined. The transaction succeeds only if the whole bundle of operations succeeds. Such a system requires a software and a hardware component. An example is an airline reservation that only succeeds if it is paid before [MLNETas].

Therewith a transaction can commit, all parts of the system have to ensure that all modifications of data will be durable; otherwise the whole transaction fails [MLNETat].

Automatic and manual transaction models are supported by the Common Language Runtime<sup>469</sup> [MLNETau].

A manual transaction enables the developer to determine the start, the result and the ending of the transaction, to manage each connection and resource enlistment inside the transaction boundary [MLNETav].

A .NET Framework class, a XML Web service method or an ASP.NET page is carried out automatically in the range of a transaction if it is declared to join in a transaction [MLNETaw].

### **14.4.12. Security**

With the .NET Framework and the Common Language Runtime<sup>470</sup> it is possible to use role-based security and cryptography [MLNETax]. This section discusses security items with MS.NET. Content are basic security terms, security policy management, cryptography, role-based security, access security and security tools.

Access security is supplied by the Microsoft .NET Framework. Role-based security and access security use the Common Language Runtime [MLNETay].

The Common Language Runtime permits only activities for which the code has the authorization. Thereby the code can demand that its caller has a permission to run, permission can be given during the runtime or the code itself can demand for the necessary permission. There are role-based security permissions, identity permissions and code access permissions [MLNETaz].

#### **14.4.12.1. Basic Security Terms**

This section presents some basic security terms.

---

<sup>469</sup> c.p. 14.3.1.

<sup>470</sup> c.p. 14.3.1.

- *Type-safe* code means that only that memory location is accessed for which it is authorized and the Common Language Runtime<sup>471</sup> could isolate assemblies from each other [MLNETba].
- The *security policy* is a guideline for the Common Language Runtime that determines what the code is allowed to do [MLNETbb].
- A *principal* is a kind deputy of the user. The .NET Framework offers custom principals, windows principals and generic principals [MLNETbc].
- The *authentication* checks the identity of a principal [MLNETbd].
- The *authorization* checks if a principal has the permission to carry out a requested action [MLNETbe].

#### **14.4.12.2. Access Security**

The .NET Framework offers the so-called code access security [MLNETbf].

#### **14.4.12.3. Role-based Security**

Role-based security is used by the Common Language Runtime<sup>472</sup> with the custom identity or on a Windows account as fundament. Thereby the role of a user is checked [MLNETbg].

Therefore information about the principal is arranged. In the .NET Framework it is possible to make authorization decisions for reason of the role membership (group of principals with the same rights) or of the identity of the principal or both parts. A principal can be a member of multiple roles. The role is used to check if somebody is authorized to implement a particular operation. The .NET Framework enables the usage of the role-based security on a server or on a client [MLNETbh].

#### **14.4.12.4. Cryptography**

The task of cryptography is to make communication secure if it is implemented over insecure channels.

---

<sup>471</sup> c.p. 14.3.1.

<sup>472</sup> c.p. 14.3.1.

For this cryptography should lead to authentication (assurance that data comes from a particular source), data integrity (no data modification) and confidentiality (identity or data protection). Microsoft claims that the .NET Framework provides cryptographic random number generators to create unpredictable numbers [MLNETbj].

To achieve these targets the .NET Framework enables the following cryptographic core functions [MLNETbj]:

- *Cryptographic hashes*: Data of any length are classified to a byte-sequence with fixed length and hashes are statistically unique.
- *Cryptographic signing*: Assurance that the data comes from a particular source by making an unambiguous digital signature. Therefore, a hash function is used.
- *Public-key encryption*: This so-called asymmetric cryptography uses a pair of private- and public-keys to encode or decode the data to prevent that the data is seen by unauthorized eyes.
- *Private-key encryption*: This so-called symmetric cryptography uses a single, secret key which is commonly used to encode and decode the data to prevent that the data is seen by unauthorized eyes.

The .NET Framework provides an extendible cryptography model with object inheritance, cryptographic configuration and stream design.

This object model uses a pattern of derived class inheritance. The top of the hierarchy is the abstract level algorithm type class. From this class the abstract level algorithm class is inherited. The next level is fully implemented and is inherited from the algorithm class.

To employ hash algorithm and symmetric algorithm a stream-orientated design is used [MLNETbj].

#### **14.4.12.5. Security Policy Management**

The so-called security policy are regulations of the Common Language Runtime<sup>473</sup> that settle the permissions for the code. The runtime takes care that the code only uses sources that are permitted.

Therefore, the security policy arranges code groups that classify code by characteristics and give them permissions [MLNETbga].

For Microsoft the security policy model consists of the security policy levels (enterprise, machine, user), the hierarchy of the code groups (inside the enterprise, machine and user levels), the named permissions sets (classified by the code groups), the evidence (information about the conformity of the code) and the application domain hosts (offers evidence to the Common Language Runtime to check code with regard the code group) [MLNETbha].

#### **14.4.12.6. Security Tools**

To check applications and components and to carry out security-related tasks the .NET Framework SDK offers command-line tools.

These tools enables for example the configuration of the security policy<sup>474</sup>, the management of certificates, the viewing of the assembly's demanded permissions or modifying the registry [MLNETbia].

#### **14.4.13. System Monitoring Components**

System monitoring components enable the monitoring and modification of system resources [MLNETbja]. This section provides information about the `Process`, `EventLog` and `ServiceController` components.

Microsoft claims that so-called Windows performance counters enable components and applications to analyze the performance data like the application performance, the fine-tune system or system bottlenecks. The .NET Framework offers the possibility to use performance counters on remote or local machines, to generate

---

<sup>473</sup> c.p. 14.3.1.

<sup>474</sup> c.p. 14.4.12.1.

custom counters on Windows systems and to put values to with .NET generated custom counters on the local machine [MLNETbk].

The `EventLog` component of the .NET Framework enables the connection to Windows event logs on a remote or on a local machine to create new usage patterns, to solve problems and to check the access to the system [MLNETbl].

To monitor Windows services the `ServiceController` component is used for custom commands on the service, to get back lists of available services and to implement administrative tasks [MLNETbm].

Many Windows process tasks are administrated with the `Process` component [MLNETbn].

#### **14.4.14. Microsoft .NET Passport<sup>475</sup>**

The .NET Passport started in 1999. It offers a Kids Passport, so-called .NET Passport express purchase and single sign in (SSI).

The single sign in (SSI) is an Internet participation mechanism among Web sites to protect the integrity of interaction.

The Kids Passport is part of the single sign in of the .NET Passport and is not a web filter. According to Microsoft it supports the parents by changing profile exceptions which are detailed in the Passport Privacy Statement and by choosing a consent level [MS02d].

The .NET Passport express purchase offers a so-called .NET Passport wallet with shipping and billing data. Therefore, the credit card numbers are protected with a Triple Data Encryption Standard algorithm and the data transaction uses a SSL encoding.

It is possible to use the same authentication system for all used sites. The .NET Passport data can be used with many devices. The password and the sign-in name have to be inserted per session only one time. It is possible to buy with fewer clicks.

---

<sup>475</sup> This section uses [MS02c]



Therefore, with one click the wallet is signed in and with a second click the data with the shipping and billing information is transmitted to the shop.

Because of a complaint of US-consumer protection organizations the US FTC (Federal Trade Commission) enacted that for the next 20 years the safety of Passport will be regularly checked [AP02,p19].

## **14.5. Building Applications**

This section provides an overview of some programming concepts. Content are ASP.NET, Windows Forms, Windows Service Applications and Design-Time Support.

### **14.5.1. ASP.NET**

ASP.NET is provided by the .NET Framework and it is a unified Web development platform [MLNETbo]. This section introduces this theme.

Microsoft claims that applications can be written with any .NET compatible language. ASP.NET is used in a .NET-based environment that is compiled. ASP.NET is almost completely syntax compatible with ASP. ASP.NET can co-operate with WYSIWIG (What You See Is What You Get) editors.

There are two features possible for generating ASP.NET application supported by the same infrastructure. The features are Web services and Web forms or a combination of both. A XML Web service has the advantage of being able to use a server functionality remotely. Web Forms offer the possibility to have forms-based Web pages [MLNETbp].

Features of ASP.NET [MLNETbq]:

- Database management is possible.
- Custom events can be implemented which take part in every demand to the application.
- The session-state and application facilities of ASP are still usable with the .NET Framework.

- ASP.NET code is compiled, modules are removable, performance counters are available and caching of services is possible.
- Custom debug statements are available.
- The configuration settings are saved in XML files.
- ASP.NET is not completely backward compatible.
- Default schemes for authentication and authorization are available.
- To write the logic, implemented at the application level, the code can be written in a compiled class or in the `global.asax` text file (includes code for responding to application level events fired by HTTP modules or by ASP.NET [MLNETcp]).
- Communication with low-level request and response services of the IIS Web server is possible.

### **14.5.2. Windows Service Applications**

The .NET Framework enables to generate services by generating an application that is installed as service. Windows Service Applications can be automatically started. This part gives a short overview.

They do not contain a user interface and they are suited for the usage on servers. Services are created by generating the application and then installing them with the utility `InstallUtil.exe`. With the so-called Services Control Manager the service can be administrated and managed.

Characteristics of Windows Service applications:

- On another Windows station the Windows Service applications are executed than other application
- Installation components are required which register and install the service.
- A service must be installed and executed.
- The Windows Service applications are executed in an own security context.

- The `Run` method of the `Main` method loads the service into the Services Control Manager.

There are the following basic states for services: Running, Paused and Stopped [MLNETbr].

### **14.5.3. Windows Forms**

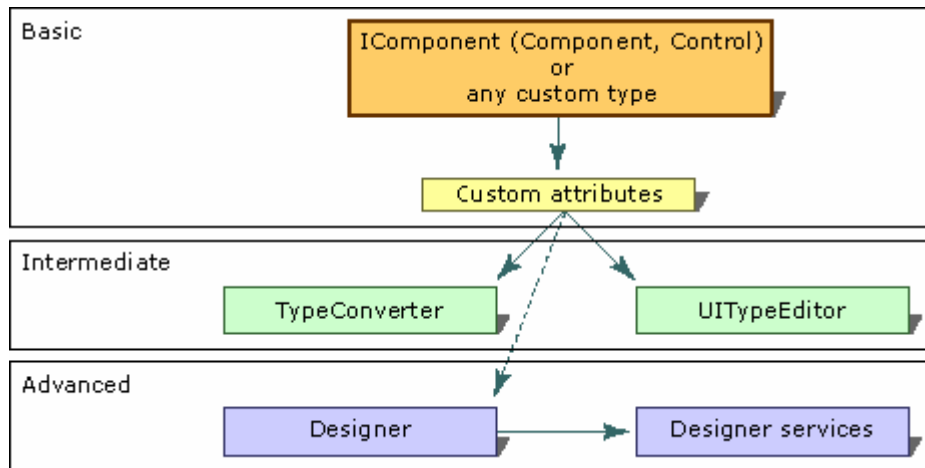
Windows Forms are a development platform that enables the creation of Windows applications.

Therefore, the .NET Framework offers object-orientated classes. Windows Forms can be used as multi-tier distributed function or as local user interface.

A Form is an object with events, properties and methods. It is an instance of a class and there is the possibility of inheritance. Forms can be dialog boxes, display surfaces or graphical routines, multiple document interface (MDI) windows or standard windows. Forms are also controls. Forms are used for communication with the user [MLNETbs].

### **14.5.4. Design-Time Support**

The design-time support is offered by the .NET Framework through a built-in design-time architecture. Design-time functionality pertains to the behaviour and display of a control or component in a visual designer [MLNETbt].



**Figure 68:** Levels of design-time support<sup>476</sup>

With the .NET Framework three levels of design-time support can be offered by control or component developers [MLNETbu] as shown in figure 68:

- The basic level enables the .NET Framework to offer classes with design-time functionality can be related with a component and its members.
- The intermediate level enables the usage of classes with type conversion and of classes with a custom user interface for managing properties.
- The advanced level enables a component developer to use classes (designer) that adapt the design-time representation of a component.

## 14.6. Object Rexx and MS.NET

To program the MS.NET classes there is a language specific compiler necessary. There is no compiler available for Object Rexx [MS02b]. Here is a possibility described to create .NET components which can be accessed via the ActiveX interface with Object Rexx or another language that can implements this interface.

This section discusses the exposing of .NET Framework components for usage with COM, the Assembly Registration Tool (Regasm.exe), COM Interop, COM Wrappers, COM Callable Wrapper with an example and the conclusion.

<sup>476</sup> Taken from [MLNETbu]

### 14.6.1. Exposing of .NET Framework Components for Usage with COM

.NET Framework Components can be exposed for usage with COM [MLNETbv]. This section discusses the prerequisites and the possibilities.

Prerequisites for .NET Framework Components when used with COM [MLNETbw]:

- Managed types have to be public so that they are visible to COM because only public types in an assembly are registered and used with a type library
- Types are not abstract
- A public default constructor enables the activation from COM
- Fields, properties, methods and events have to be public
- Interfaces should be implemented explicitly by classes

Information about managed types for COM developers to package an assembly [MLNETbx]:

- Versioning instructions
- Deployment instructions describe that unsigned assemblies are installed as private assembly on the user's machine and by a publisher signed strong-named<sup>477</sup> assemblies are installed into the global assembly cache.
- List of types which show if the managed types are for COM visible and creatable, visible but not creatable or invisible. It is possible that an assembly can include creatable, non-creatable, visible and invisible types
- Type library is necessary for the most types

Options for the creation of a type library (only the public types of the assembly are integrated to the type library) [MLNETbx]:

- *Type Library Exporter*. Command-line tool which creates a COM type library with the classes and interfaces of an assembly.

---

<sup>477</sup> c.p. 14.3.2.

- *TypeLibConverter Class*. This tool works like the Type Library Exporter and is stored in the `System.Runtime.InteropServices` namespace.
- *.NET Services Installation Tool* (Regsvc.exe) can load and register an assembly and additionally install the type library into a COM+1.0 application.
- *Assembly Registration Tool*. This tool can create and register a type library with the option `/tlb`. If this option is not selected the types are only selected in an assembly.

### 14.6.2. Assembly Registration Tool (Regasm.exe)<sup>478</sup>

This tool enables COM to generate .NET Framework classes by reading the metadata in an assembly and by inserting the entries to the registry. This section contains a description and an example with Object Rexx of this tool.

Syntax:

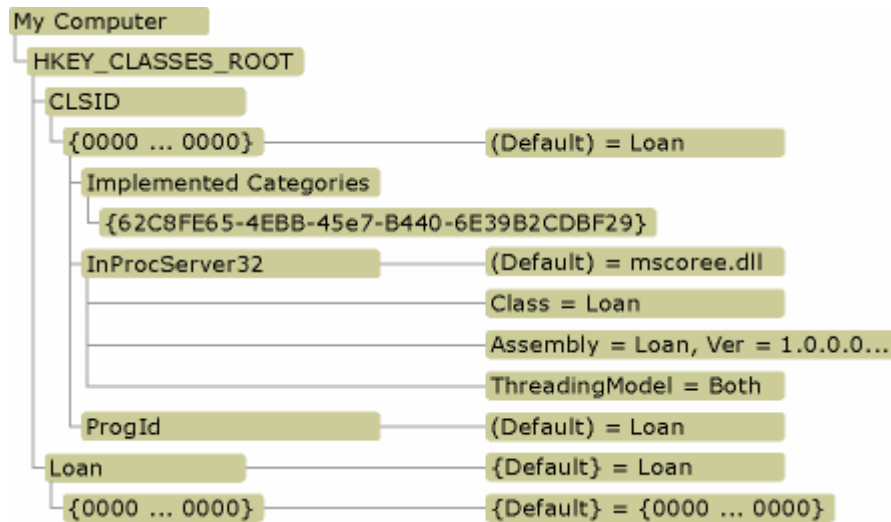
```
regasm assemblyFile [options]
```

With the option `/regfile` a `.reg` file is created. Then the entries are not inserted to the registry but to this file. With the `/tlb` option a type library which includes the types of the assembly is created and registered.

In the registry new entries for the CLSID are inserted to the registry key `HKCR/CLSID`. Under the key `HKCR\CLSID\{0000...0000}` the default value is set to the ProgID of the class. The values `Assembly` and `Class` are also added. As subkey of the key `HKCR\CLSID\{0000...0000}` the key `InProcServer32` is generated whose default value is the name of the DLL [MLNETbz]. Figure 69 shows a registry entry of `Mscoree.dll`.

---

<sup>478</sup> [MLNETby]



**Figure 69:** Illustration of a registry entry with a reference to Mscoree.dll<sup>479</sup>

*Example:*

This command inserted in the command prompt registers all public classes, which are contained in Mscorlib.dll. This is the DLL of the System.Object namespace, which is the root namespace [MLNETca].

```
Regasm.exe Mscorlib.dll
```

After that, many .NET classes are visible for COM. Only a few can be referenced with the ActiveX interface of Object Rexx. Most classes are visible but they can't be used via ActiveX<sup>480</sup>.

<sup>479</sup> Taken from [MLNETbz]

<sup>480</sup> c.p. 14.6.2.

Click or enter application PROGID/CLSID to reveal OLEObject properties		
2119	<b>System.MissingFieldException</b>	System.MissingFieldException
2120	<b>System.MissingMemberException</b>	System.MissingMemberException
2121	<b>System.MissingMethodException</b>	System.MissingMethodException
2122	<b>System.MTAThreadAttribute</b>	System.MTAThreadAttribute
2123	<b>System.MulticastNotSupportedException</b>	System.MulticastNotSupportedException
2124	<b>System.Net.WebClient</b>	System.Net.WebClient
2125	<b>System.Net.WebHeaderCollection</b>	System.Net.WebHeaderCollection
2126	<b>System.NonSerializedAttribute</b>	System.NonSerializedAttribute
2127	<b>System.NotFiniteNumberException</b>	System.NotFiniteNumberException
2128	<b>System.NotImplementedException</b>	System.NotImplementedException
2129	<b>System.NotSupportedException</b>	System.NotSupportedException
2130	<b>System.NullReferenceException</b>	System.NullReferenceException
2131	<b>System.Object</b>	System.Object
2132	<b>System.ObsoleteAttribute</b>	System.ObsoleteAttribute
2133	<b>System.OutOfMemoryException</b>	System.OutOfMemoryException
2134	<b>System.OverflowException</b>	System.OverflowException
2135	<b>System.ParamArrayAttribute</b>	System.ParamArrayAttribute
2136	<b>System.PlatformNotSupportedException</b>	System.PlatformNotSupportedException
2137	<b>System.Random</b>	System.Random
2138	<b>System.RankException</b>	System.RankException
2139	<b>System.Reflection.AmbiguousMatchException</b>	System.Reflection.AmbiguousMatchException
2140	<b>System.Reflection.AssemblyName</b>	System.Reflection.AssemblyName
2141	<b>System.Reflection.AssemblyNameProxy</b>	System.Reflection.AssemblyNameProxy
2142	<b>System.Reflection.CustomAttributeFormatException</b>	System.Reflection.CustomAttributeFormatException
2143	<b>System.Reflection.InvalidFilterCriteriaException</b>	System.Reflection.InvalidFilterCriteriaException
2144	<b>System.Reflection.TargetException</b>	System.Reflection.TargetException
2145	<b>System.Reflection.TargetParameterCountException</b>	System.Reflection.TargetParameterCountException
2146	<b>System.Resources.MissingManifestResourceException</b>	System.Resources.MissingManifestResourceException
2147	<b>System.Runtime.CompilerServices.CallConvCdecl</b>	System.Runtime.CompilerServices.CallConvCdecl
2148	<b>System.Runtime.CompilerServices.CallConvFastcall</b>	System.Runtime.CompilerServices.CallConvFastcall
2149	<b>System.Runtime.CompilerServices.CallConvStdcall</b>	System.Runtime.CompilerServices.CallConvStdcall

**Figure 70:** RGF\_OLEINFO.HTA with the new created ProgIDs of .NET classes<sup>481</sup>

For example the root class `System.Object` or the class `System.Random` can be referenced as illustrated in figure 70. Code 60 demonstrates the access of some methods of `System.Random` with the methods `Next` and `NextDouble` which calculate some random numbers [MLNETcb].

<sup>481</sup> c.p. 7.3.



```

-----
-- MS_NET_System_Random.rex --
-----

    -- Instantiation of an object of System.Random
NETObject = .OLEObject~New("System.Random")
SAY NETObject~Next -- Returns a random number
SAY NETObject~NextDouble -- Returns a random number between 0,0 and 1,0

```

**Code 60: MS\_NET\_System\_Random.rex**

### 14.6.3. COM Interop

COM interop enables forward and backward compatibility. Thereby managed code can be accessed by COM clients. Metadata can be exported from an assembly to a type library and the managed component can be registered as COM component [MLNETcc]. This section explains COM Wrappers and COM Callable Wrappers.

#### 14.6.3.1. COM Wrappers<sup>482</sup>

Clients of .NET use reflection to get a description of the functionality of an object, whereat the clients of COM objects request an interface to get back an interface pointer or not to get information if a service is available.

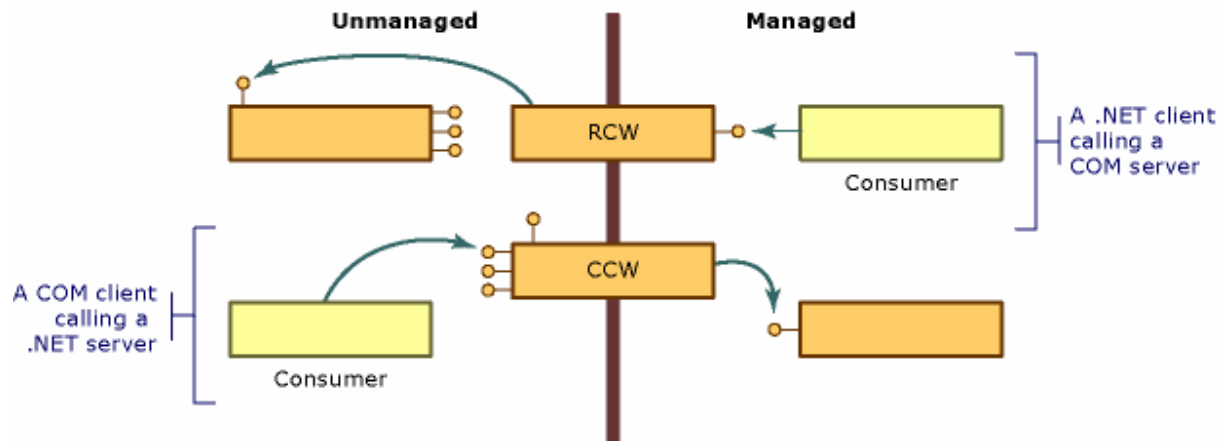
The Common Language Runtime<sup>483</sup> administrates the lifetime of objects on the one side and on the other side the client of COM objects must administrate the lifetime of those objects.

The .NET objects are in the memory which is administrated by the .NET Framework execution environment.

So-called wrappers solve these problems by letting unmanaged and managed code think that they are calling objects that are in the corresponding environment. There are two kinds of wrapper: The runtime callable wrapper (RCW) which handles calls of a managed (.NET) client on a COM object and the COM callable wrapper (CCW) which works in the other direction and handles calls of a COM client on a .NET object. This principle is explained in figure 71.

<sup>482</sup> [MLNETcd]

<sup>483</sup> c.p. 14.3.1.



**Figure 71: Principle of RCW and CCW<sup>484</sup>**

### 14.6.3.2. COM Callable Wrapper<sup>485</sup>

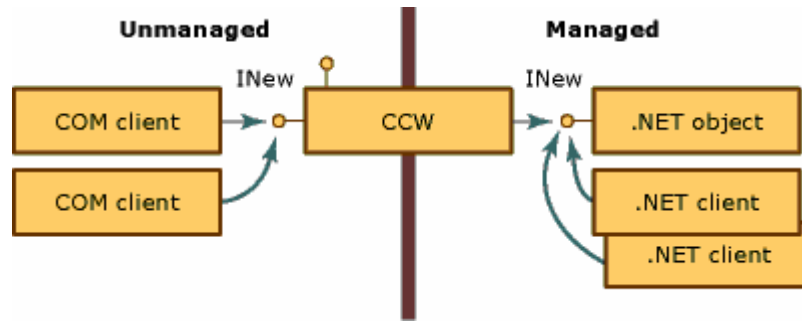
The CCW is a kind of proxy for the managed object. This section defines the term COM Callable Wrapper and there is an example with Object Rexx that illustrates the usage of a CCW.

It marshals references between unmanaged and managed code as illustrated in figure 72. In this way if a COM client wants to reference .NET object the managed object and a COM callable wrapper are generated by the Common Language Runtime<sup>486</sup>. The figure 72 shows that there is exactly one CCW generated for a managed object although there are multiple COM clients that want to reference the .NET object. The CCW has a single reference to the managed object that is garbage collected and that implements the interface. It is possible that .NET and COM clients reference the same object at the same time.

<sup>484</sup> Taken from [MLNETcd]

<sup>485</sup> [MLNETce]

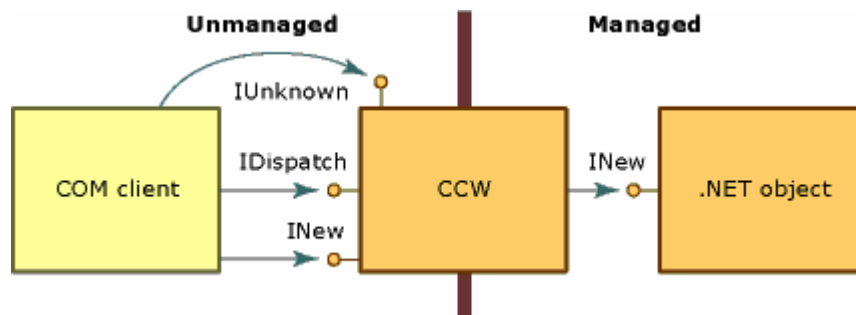
<sup>486</sup> c.p. 14.3.1.



**Figure 72: Access of a CCW**<sup>487</sup>

The CCW also handles the object lifetime (reference-counted like COM) and object identity (the Common Language Runtime<sup>488</sup> offers memory to the CCW so that the COM client can call the wrapper directly) [MLNETce].

The invocation of methods of .NET objects is equal to the invocation of COM object methods. Therefore, the CCW enables that all public, COM-visible data types, return values and interfaces are usable for COM clients. In this way the CCW exposes traditional COM interfaces like the `IUnknown` and `IDispatch`. The figure 73 demonstrates a single reference on the .NET object by the CCW. The .NET object and the COM client communicate via the CCW stub and proxy construction [MLNETcf]



**Figure 73: COM interfaces for the CCW**<sup>489</sup>

The .NET Framework provides the following implementations of COM interfaces: `IDispatch`, `IErrorInfo`, `IUnknown`, `ISupportErrorInfo`, `ITypeInfo` and `IProvideClassInfo`.

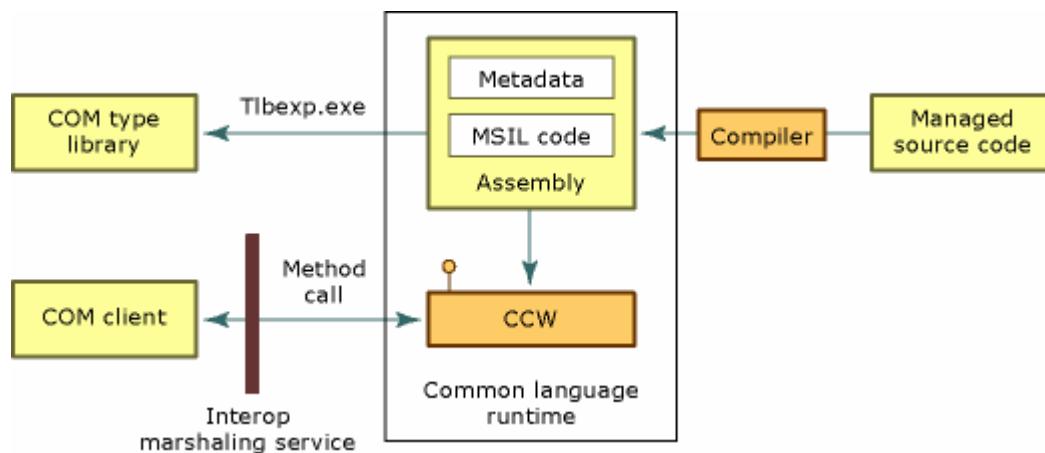
<sup>487</sup> Taken from [MLNETce]

<sup>488</sup> c.p. 14.3.1.

<sup>489</sup> Taken from [MLNETcf]

Managed classes can supply the following COM interfaces: The (*\_classname*) class interface, IEnumVARIANT, IDispatchEx, IConnectionPoint and IConnectionPointContainer [MLNETcf].

Each type in an assembly is described by metadata that is created by compiling a managed project to an assembly DLL. If a COM client references a managed object, the metadata is utilized to create a CCW. In the figure 74 the managed types are transformed to COM with the tool Tlbexp.exe [MLNETcg].



**Figure 74: CCW method call**<sup>490</sup>

The usage of attributes regulated by the interop marshalling service enables the management of data and interface marshalling behaviour like the controlling if an assembly is exposed to COM [MLNETcg].

*Example* [MM03]:

In this example from the Homepage of Andy McMullan<sup>491</sup> first a .NET component is created which is then accessed via a CCW with Object Rexx as COM client.

First, this C# file with the name `testcomserver.cs` is created (OtherScript 13).

<sup>490</sup> Taken from [MLNETcg]

<sup>491</sup> <http://www.eponymous.eclipse.co.uk/dotnetfaq.htm>

```
using System;
namespace AndyMc
{
    public class CSharpCOMServer
    {
        public CSharpCOMServer() {}
        public void SetName( string name ) { m_name = name; }
        public string GetName() { return m_name; }
        private string m_name;
    }
}
```

**OtherScript 13:**      **testcomserver.cs**

This .cs file is compiled from the command prompt with the following command:

```
Csc /target:library testcomserver.cs
```

Csc is the compiler [MLNETch] and the option /target:library means that a DLL is created [MLNETci].

Then a DLL is generated which is then registered with the tool Regasm.exe<sup>492</sup>:

```
regasm testcomserver.dll /tlb:testcomserver.tlb /codebase
```

Here the option /tlb means that a type library is created and the option /codebase creates a codebase entry to the registry [MLNETby]. Figure 75 shows the new created ProgID.

---

<sup>492</sup> c.p. 14.6.2.

Click or enter application PROGID/CLSID to reveal OLEObject properties		
49	ADOX.Table.2.7	ADOX.Table.2.7
50	ADOX.User.2.7	ADOX.User.2.7
51	ADs	ADs Provider Object
52	ADsDSOObject	ADsDSOObject
53	ADsNamespaces	ADs Namespaces Object
54	ADsSecurityUtility	%ADS_SECURITY_UTILITY_OBJECT%
55	ADSystemInfo	AD System Info Object
56	Agent.Control	Microsoft Agent Control 1.5
57	Agent.Server	Microsoft Agent Server 2.0
58	Alg.AlgSetup	
59	AMOVIE.ActiveMovieControl	ActiveMovieControl Object
60	AMtoolbar.AMtoolbar	AMtoolbar Class
61	AndyMc.CSharpCOMServer	AndyMc.CSharpCOMServer
62	APHandler.Handler	Handler Class
63	AppBarCom.AppBarInvoker	AppBarInvoker Class
64	AppExport.AppExport	AppExport Class
65	AppImport.AppImport	AppImport Class
66	AppWizard6.SubWizard	AppWizard6.SubWizard
67	AppWizard6.Wizard	AppWizard6.Wizard
68	ArticleData.ArticleData	ArticleData Class
69	ArticleLayout.ArticleLayout	ArticleLayout Class
70	ASControls.InstallEngineCtl	InstallEngineCtl Object
71	ASFSession.ASFSession	ASFSession Class
72	ASFSessionProp.ASFSessionProp	ASFSessionProp Class
73	ASP.HostEncode	ASP Host Encode Object
74	ATL.Registrar	Registrar Class
75	Attr.Attr	Attr Class
76	AUDIOCONTROL.CurveEditCtrl.1	CurveEdit Control
77	AUDIOCONTROL.KnobCtrl.1	Knob Control
78	AUDIOCONTROL.LevelSliderCtrl.1	LevelSlider Control
79	AudioVBScript	DirectMusic Audio VB Script Language

**Figure 75:** RGF\_OLEInfo.hta: New ProgID “AndyMc.CSharpCOMServer”<sup>493</sup>

This newly created .NET component can be referenced with the ActiveX interface of Object Rexx like in code 61.

<sup>493</sup> c.p. 7.3.

```
-----  
-- MS_NET_AndyMc_CSharpCOMServer.rex --  
-----  
-- Instantiation of an object of AndyMc.CSharpCOMServer  
NETObject = .OLEObject~New("AndyMc.CSharpCOMServer")  
NETObject~SetName("Hello World") -- Inserts the text "Hello World"  
SAY NETObject~GetName -- Prints the content of GetName to the display
```

**Code 61:**      **MS\_NET\_AndyMc\_CSharpCOMServer.rex**

Output:

Hello World

### 14.6.4. Conclusion

This is a self-created component. In the author's opinion this principle can also be used to expose .NET namespaces with its classes, methods, properties and events for COM. In this way these objects can be used via the ActiveX interface of Object Rexx or another language like Visual Basic Script or JScript.

For example, a COM object `XSystem.XConsole` with a method `XWriteLine` can simulate the real .NET object `System.Console` with the real .NET method `WriteLine`. Then in Object Rexx the object `XSystem.XConsole` is instantiated. This object has a method `XWriteLine` and a text is hand over. This text is then hand over to the real .NET object `System.Console` with the real .NET method `WriteLine` over CCW. Then this text is printed to the display.

In this way it should be possible to reference .NET objects via the ActiveX interface of Object Rexx.

## 15. Examples of Use

This chapter presents some selected projects of a seminar at the University of Augsburg that show how to use the Automation technology with Object Rexx in a business. The projects *Döner Dome Restaurant*, *High Value Customers Consultancy* and *Tourplanning* are described.

### 15.1. Döner Dome Restaurant<sup>494</sup>

This project describes a franchise business model with multiple fast food restaurants, which sell Döner Kebab.

In this project a cash registering system is developed. This system should be cheap, prevent media breaks, prevent double work, count automatically the daily sales revenues, implement automatically the late order of the sold products and have a user-friendly user interface.

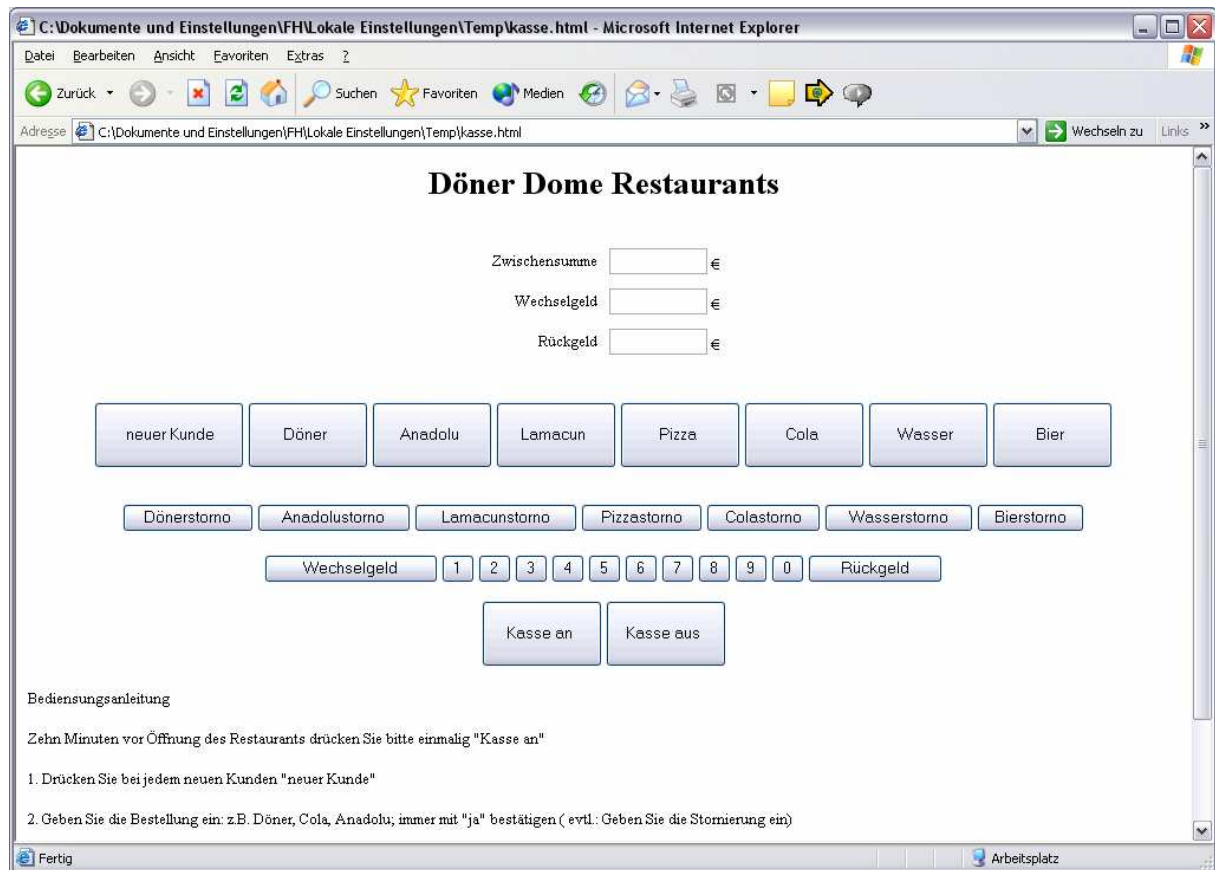
This task was solved by a system consisting of PCs with touch screens. The user interface was implemented with HTML. The daily sales revenues are computed with Object Rexx, and documented with MS Excel. The automatically late order of the sold products is transmitted with the e-mail program Eudora to the warehouse.

To apply this concept a PC, a touch screen, the MS Internet Explorer, Qualcomm Eudora, Object Rexx, MS Office and Windows as an operating system are required. Figure 76 shows the user interface of the cash registering system.

---

<sup>494</sup> This section uses [HeLu03]





**Figure 76:** User interface of the cash registering system of project Döner Dome<sup>495496</sup>

## 15.2. High Value Customers Consultancy<sup>497</sup>

This project shows the possibilities of Automation for a consultant.

The consultant has only a few “High Value Customers”. These customers are stored in a customer database where the addresses and portfolio of the customers is saved. The bond department offers recommendations in the intranet. The task of the consultant is to send the buy or sell recommendations to the customers. The customer data is stored in an MS Excel sheet. If there is a recommendation, each customer is informed with a text written with MS Word. The addresses are pasted with Copy&Paste. The stock history is attached in tabulate form. This takes a lot of time and there is less time to consult the customers. The letters are sent to the customers by a central post office.

<sup>495</sup> MS Internet Explorer is part of MS Windows XP

<sup>496</sup> Taken from [HeLu03]

<sup>497</sup> This section uses [MeSc03]

Automation would increase efficiency.

The user interface, which handles the Automation, is an Object Rexx file, which is embedded in a HTML document. This interface can access and manage the customer data by Automation; it can receive the stock tables. The text of a serial letter can be inserted. This serial letter with the stock data and customer data is sent automatically via the automated e-mail program Qualcomm Eudora to the central post office. Figure 77 shows the user interface.

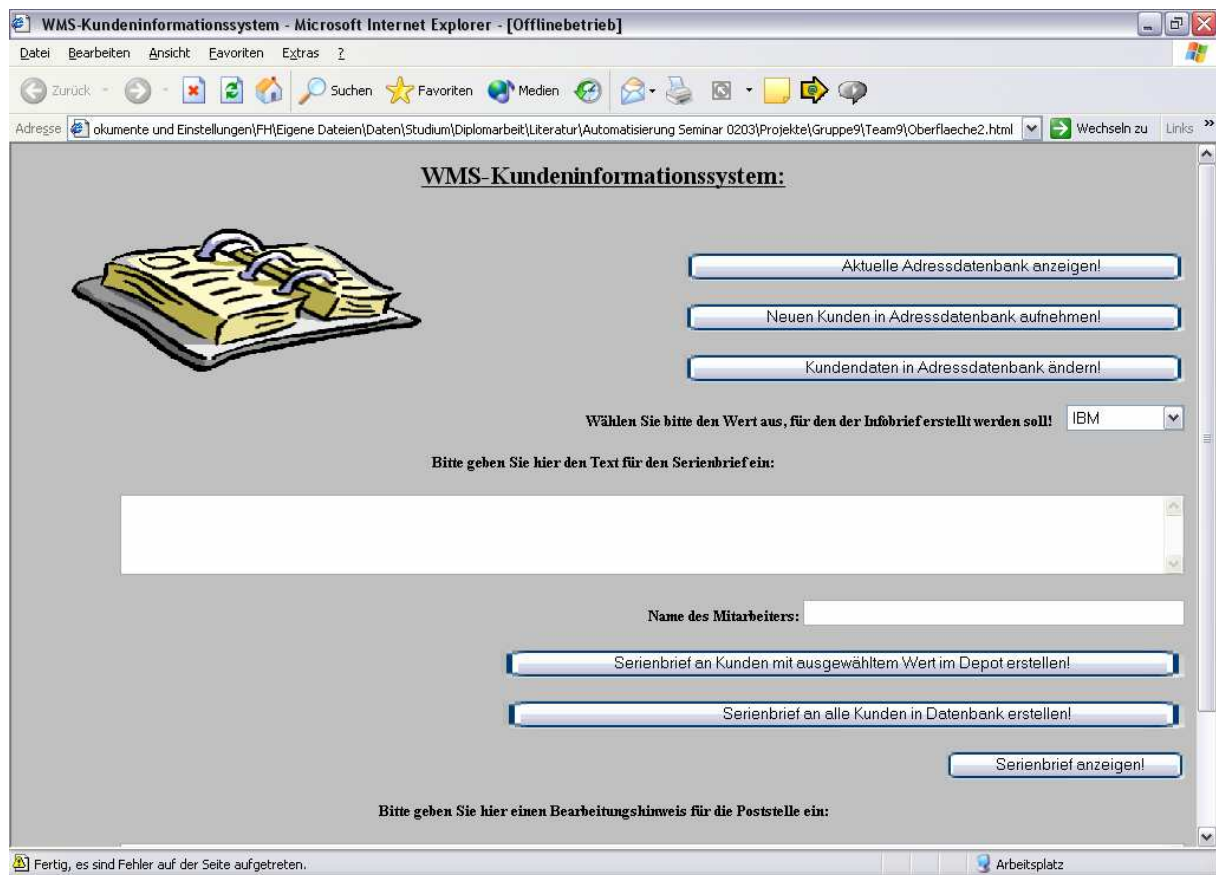


Figure 77: User interface of High-Value Customers consultancy<sup>498499</sup>

## 15.3. Tourplanning<sup>500</sup>

This project describes an enterprise in the transport and logistic field.

<sup>498</sup> MS Internet Explorer is part of MS Windows XP

<sup>499</sup> Taken from [MeSc03]

<sup>500</sup> This section uses [BuHe03]

The enterprise has each day a varying number of customers and a varying order amount. In this way it is necessary to create each day new delivery notes, new delivery certifications, new schedules for the drivers and new computing of the optimal tour. This problem is now automated. Therefore a user interface is generated with HTML. All data is stored in MS Excel. The optimal tour is computed with Object Rexx. The delivery notes and schedules are created with MS Word and the delivery notes are sent with Qualcomm Eudora as e-mail.

The user interface enables the management of the customers, the management of the fleet of cars, the registering of the orders, the beginning of the computing and the presentation of the optimal tour. To compute the optimal tours the savings proceeding is used in MS Excel. Then a table with all tours is generated. The required data is read from MS Excel to create the schedules with addresses of the customers and total load. The delivery notes are also created in this way with the respective data. A confirmation of order gets its data from MS Excel and is sent via the e-mail program Qualcomm Eudora.

Figure 78 illustrates the user interface for the tour planning.

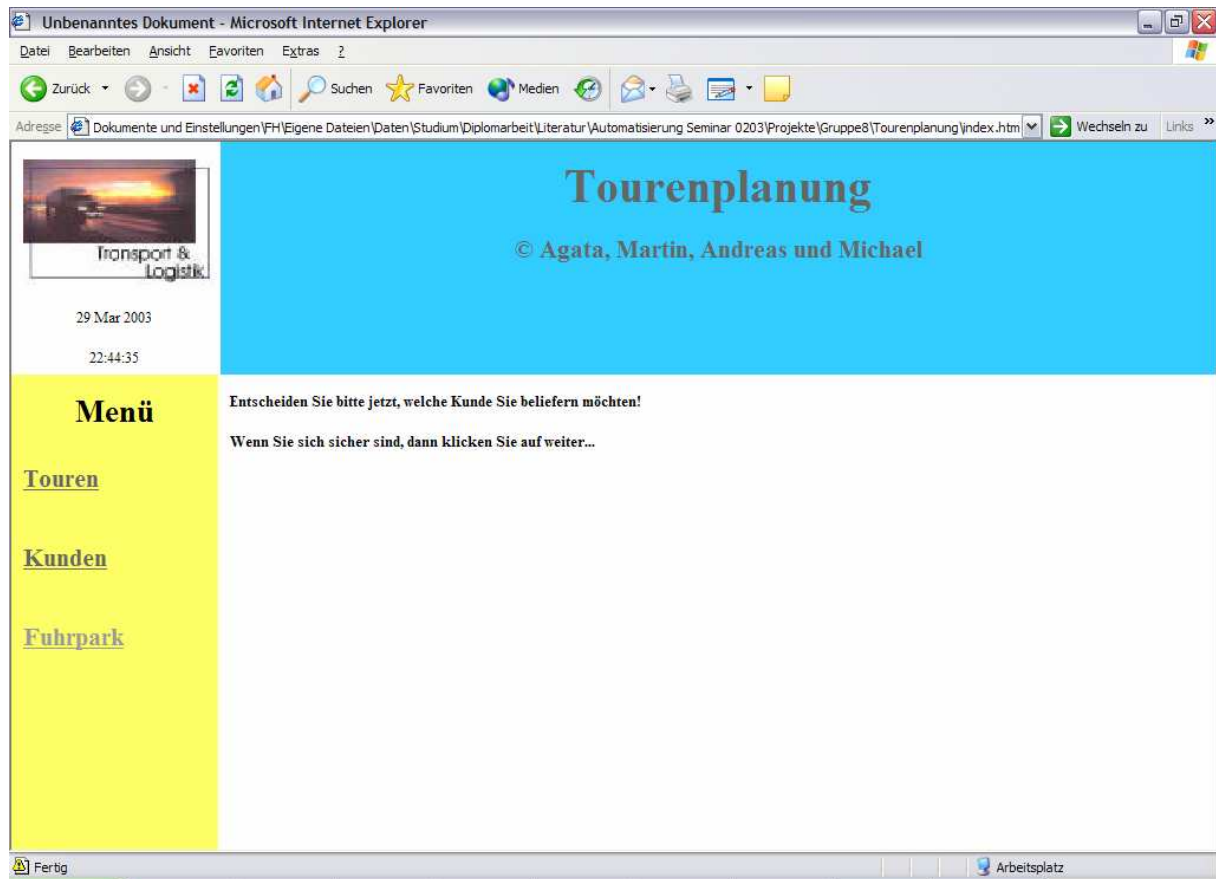


Figure 78: Tourplanning<sup>501502</sup>

<sup>501</sup> MS Internet Explorer is part of MS Windows XP

<sup>502</sup> Taken from [BuHe03]

## Summary

First, this paper explained the Component Object Model. Then this master thesis discussed the usage of ActiveX Automation theoretically and practically with examples like MS Office components, Windows Management Instrumentation (WMI), Windows Script Host (WSH), MS Agent technology and MS Speech technology. One section described how to get Object Rexx code. Other chapters showed some useful tools, the embedding of Object Rexx in HTML/XML or gave an overview over MS.NET and the possibility to access MS.NET functions with Object Rexx via so-called COM Callable Wrappers. The .NET part is to emphasize, because Object Rexx cannot use .NET in the same kind like so-called .NET languages because there is no language compiler for the .NET Framework available. At the end there were some examples which show how to use Automation with Object Rexx in a business context.

Object Rexx is a very powerful instrument for Automation. Compared with other Automation codes like Visual Basic Script code, the Object Rexx code is often more direct. Sometimes Object Rexx code works in another kind as Visual Basic Script. For example, this problem occurs if the `Copy` message is used. Then the `UNKNOWN` method of `OLEObject` class of Object Rexx must be invoked. Another example is macros of MS Office tools which are sometimes programmed in another kind. Then it is necessary to understand the system by trial and error. Helpful is here section 5, which offers instructions to get the Object Rexx code. Section 6 describes the `OLEObject` class of Object Rexx with its methods. This class enables the usage of OLE with Object Rexx.

There were sometimes problems with the implementation of events. These problems occurred with the `WSHController` object<sup>503</sup>, the MS Speech technology<sup>504</sup> and the MS Agent technology<sup>505</sup> (for this technology is now a new `OREXXOLE.DLL` available). These problems arise because Object Rexx is called in this case with a special ID by `QueryInterface` and not like normally by `IID_IUnknown` respectively

---

<sup>503</sup> c.p. 13.7.2.3.

<sup>504</sup> c.p. 12.2.

<sup>505</sup> c.p. 11.3.

IID\_IDispatch. For Object Rexx it is not possible to create for each MS OLE object a particular call. Probably there will be another solution for Object Rexx in the future [Doe03].

The documentation for OLE, COM and Windows Script Host from Microsoft is in some cases wrong [En03] or incomplete [Fla03e]. This makes it more difficult for non-Microsoft languages to use these parts of Microsoft technologies. This probably is a reason for some developers not to use these languages, but a Microsoft language. Object Rexx is for the most part compatible with new user interfaces like the MS Agent technology<sup>506</sup> and the MS Speech technology<sup>507</sup>. These user interfaces can be used for a help service<sup>508</sup> of an application (MS Agent) or to read out texts like MS Word documents<sup>509</sup> or e-mails (MS Speech technology).

Object Rexx provides access to Windows Script Host. This technology enables for example interaction among ActiveX components, the access of the registry<sup>510</sup> or the implementation of scripts on another machine in a network<sup>511</sup>. WMI (Windows Management Instrumentation)<sup>512</sup> can be also used with Object Rexx and provides control and management information, for example the free space on a hard disk<sup>513</sup>.

A great part takes the chapter over MS.NET. This technology is one of various technologies that can be used for Web Services. The significance of Web Services will increase in the future. A market research of Forrester Research has the result that about 84% of the asked European enterprises plan to increase investments in the Web Service area in 2003 and Cap Gemini means that standardization for Web Services will be finished in 2006 [DPA03]. Unsolved problems are billing systems for this technology [DPA03a].

---

<sup>506</sup> c.p. 11.

<sup>507</sup> C.p. 12.

<sup>508</sup> c.p. code 21

<sup>509</sup> c.p. 12.1.3.

<sup>510</sup> c.p. 13.7.4.2.

<sup>511</sup> c.p.13.7.2.

<sup>512</sup> c.p. 10.

<sup>513</sup> c.p. 10.4.

The MS.NET technology can be used with Object Rexx. Therefore, a detour via the COM Callable Wrapper must be gone, because Object Rexx has no language compiler that converts the data to Microsoft Intermediate Language (MSIL).

For MS.NET is the MSIL an intermediate goal. There will be for .NET named languages the same semantic [Fla03]. A problem that occurs with .NET is the compatibility. An example therefore is Visual Basic .NET that is an object-oriented programming language [MLNETcq]. There are many changes between Visual Basic 6.0 and Visual Basic .NET. A lot of elements of Visual Basic 6.0 are reclassified, renamed, or combined with other programming elements for Visual Basic .NET [MLNETcr]. That means that there are problems with the backward compatibility of Visual Basic.NET to Visual Basic [Dr03]. This can be in the future a lock-in trap [Fla03]. Not until 2005/2006, there will be pure .NET applications [Fla03].

Finally is to say that Object Rexx has compared with other languages, a simple syntax and is easy to learn. It is suited for the usage in a business, it is compatible with other MS technologies and it is an efficient alternative to other programming languages.

## Bibliography

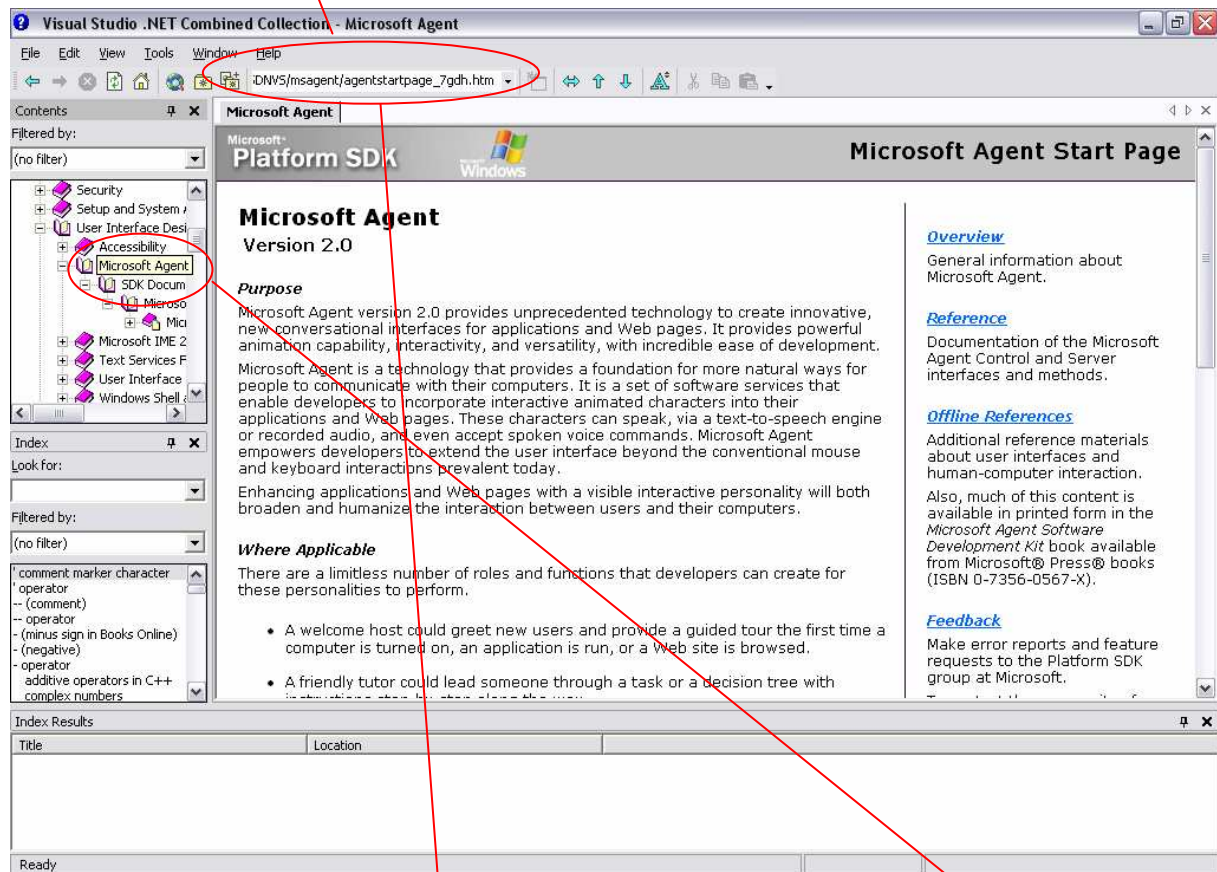
The Bibliography consists of five sections. The short references are [MLXXXx] (Microsoft Library Sources), [MLNETxx] (Microsoft Library .NET Framework Sources), [SPXX] (Microsoft Speech SDK 5.1 Documentation Sources), [SGX] (System Administration Scripting Guide Sources). Short references that are not part of these references are described in the part Other Sources.



[MLXXXx]

### MS Library Sources

The folder specifications refer to the MS Library. In this case the MS Library was installed on the machine as part of Visual Studio .NET. Thereby the Microsoft Document Explorer was installed on the machine and the folder specification is inserted in the folder line.



**Figure 79: Microsoft Document Explorer**

The short reference is inserted in brackets. The first two letters are ML for MSDN Library. The next three or four letters describe the chapter. After that there follow one or two small letters for the further distinction. In the phrase [MLAGTa] this is AGT which refers to the chapter "Microsoft Agent". The description of this short reference contains if available the name of the author, if not there occurs "N.A." for "No Author". Then the headline of the referenced text is written and at least the location is printed.

[MLAGTa]	N.A.: Microsoft Agent ms-help://MS.VSCC/MS.MSDNVS/msagent/agentstartpage_7gdh.htm
[MLAGTaa]	N.A.: Microsoft Agent. ShowDefaultCharacterProperties Method ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4egc.htm
[MLAGTb]	N.A.: Microsoft Agent. Connected Property ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_354c.htm
[MLAGTc]	N.A.: Microsoft Agent. Load Method

	<a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_0odm.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_0odm.htm</a>
[MLAGTd]	N.A.: Microsoft Agent. Top Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3luy.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3luy.htm</a>
[MLAGTe]	N.A.: Microsoft Agent. Left Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_95re.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_95re.htm</a>
[MLAGTf]	N.A.: Microsoft Agent. LanguageID Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4w2y.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4w2y.htm</a>
[MLAGTg]	N.A.: Microsoft Agent. Show Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_97ca.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_97ca.htm</a>
[MLAGTh]	N.A.: Microsoft Agent. Play Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3isq.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3isq.htm</a>
[MLAGTi]	N.A.: Microsoft Agent. Speak Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_6cbu.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_6cbu.htm</a>
[MLAGTj]	N.A.: Microsoft Agent. Unload Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4b56.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4b56.htm</a>
[MLAGTk]	N.A.: Microsoft Agent. Show Event <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_7wrw.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_7wrw.htm</a>
[MLAGTI]	N.A.: Microsoft Agent. Height Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_85pm.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_85pm.htm</a>
[MLAGTm]	N.A.: Microsoft Agent. MoveTo Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_03u2.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_03u2.htm</a>
[MLAGTn]	N.A.: Microsoft Agent. Hide Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_9eqy.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_9eqy.htm</a>
[MLAGTo]	N.A.: Microsoft Agent. Think Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_1alm.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_1alm.htm</a>
[MLAGTp]	N.A.: Microsoft Agent. ShowPopupMenu Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3ffu.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3ffu.htm</a>
[MLAGTq]	N.A.: Microsoft Agent. Speed Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_55gq.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_55gq.htm</a>
[MLAGTr]	N.A.: Microsoft Agent. GUID Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_323u.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_323u.htm</a>
[MLAGTs]	N.A.: Microsoft Agent. FontName Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_35uy.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_35uy.htm</a>
[MLAGTt]	N.A.: Microsoft Agent. Unload Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4b56.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_4b56.htm</a>
[MLAGTu]	N.A.: Microsoft Agent. Add Method <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_8kfe.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_8kfe.htm</a>
[MLAGTv]	N.A.: Microsoft Agent. Caption Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_0wyy.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_0wyy.htm</a>
[MLAGTw]	N.A.: Microsoft Agent. Voice Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_02qy.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_02qy.htm</a>
[MLAGTx]	N.A.: Microsoft Agent. Visible Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_91ii.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_91ii.htm</a>
[MLAGTy]	N.A.: Microsoft Agent. Visible Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3ega.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3ega.htm</a>
[MLAGTz]	N.A.: Microsoft Agent. Enabled Property <a href="ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3l2i.htm">ms-help://MS.VSCC/MS.MSDNVS/msagent/pacontrol_3l2i.htm</a>
[MLAPIa]	N.A.: What is an API. <a href="ms-help://MS.VSCC/MS.MSDNVS/modcore/html/deovrwhatisapi.htm">ms-help://MS.VSCC/MS.MSDNVS/modcore/html/deovrwhatisapi.htm</a>
[MLAUTa]	N.A.: Platform SDK - Automation. Container Application <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obn.htm#_oa96_container_application">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obn.htm#_oa96_container_application</a>
[MLAUTaa]	N.A.: Returning Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0mer.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0mer.htm</a>
[MLAUTab]	N.A.: Shutting Down Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_8z8z.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_8z8z.htm</a>
[MLAUTac]	N.A.: Creating the Programmable Interface <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0aat.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0aat.htm</a>
[MLAUTad]	N.A.: Creating the IUnknown Interface

	<a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_97s5.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_97s5.htm</a>
[MLAUTae]	N.A.: Creating the IDispatch Interface <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_6l5x.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_6l5x.htm</a>
[MLAUTaf]	N.A.: Implementing Dual Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_40fn.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_40fn.htm</a>
[MLAUTag]	N.A.: Registering Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0dv7.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0dv7.htm</a>
[MLAUTah]	N.A.: ProxyStubClsid32 <a href="ms-help://MS.VSCC/MS.MSDNVS/com/reg_83ua.htm">ms-help://MS.VSCC/MS.MSDNVS/com/reg_83ua.htm</a>
[MLAUTai]	N.A.: Creating Class Identifiers <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9vg3.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9vg3.htm</a>
[MLAUTaj]	N.A.: Implementing the IEnumVARIANT Interface <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7cmd.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7cmd.htm</a>
[MLAUTak]	N.A.: Implementing the _NewEnum Property <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_2ws9.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_2ws9.htm</a>
[MLAUTal]	N.A.: What Is a Type Library? <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_5mzj.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_5mzj.htm</a>
[MLAUTam]	N.A.: Type Libraries <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7jxv.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7jxv.htm</a>
[MLAUTan]	N.A.: Creating a Type Library <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7i49.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_7i49.htm</a>
[MLAUTao]	N.A.: Building a Type Library <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_1xq1.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_1xq1.htm</a>
[MLAUTap]	N.A.: Registering a Type Library <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_89o9.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_89o9.htm</a>
[MLAUTaq]	N.A.: HRESULT <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obs.htm#_oa96_hresult">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obs.htm#_oa96_hresult</a>
[MLAUTar]	N.A.: Returning an Error <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9po2.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9po2.htm</a>
[MLAUTas]	N.A.: Accessing ActiveX Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap3_0mlv.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap3_0mlv.htm</a>
[MLAUTat]	N.A.: Type Description Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap9_49pv.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap9_49pv.htm</a>
[MLAUTau]	N.A.: Type Building Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap10_725v.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap10_725v.htm</a>
[MLAUTav]	N.A.: Error Handling Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap11_0fqr.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap11_0fqr.htm</a>
[MLAUTaw]	N.A.: IClassFactory <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ctin_c_9mk9.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ctin_c_9mk9.htm</a>
[MLAUTax]	N.A.: IDataObject <a href="ms-help://MS.VSCC/MS.MSDNVS/com/oin_d_8cl0.htm">ms-help://MS.VSCC/MS.MSDNVS/com/oin_d_8cl0.htm</a>
[MLAUTay]	N.A.: IServiceProvider Interface <a href="ms-help://MS.VSCC/MS.MSDNVS/ICom/workshop/components/com/reference/ifaces/IServiceProvider/IServiceProvider.htm">ms-help://MS.VSCC/MS.MSDNVS/ICom/workshop/components/com/reference/ifaces/IServiceProvider/IServiceProvider.htm</a>
[MLAUTaz]	N.A.: IPictureDisp <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ctin_p_4gfk.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ctin_p_4gfk.htm</a>
[MLAUTb]	N.A.: Platform SDK - Automation. In-place activation <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obt.htm#_oa96_in_place_activation">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obt.htm#_oa96_in_place_activation</a>
[MLAUTba]	N.A.: IPicture <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ctin_p_482t.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ctin_p_482t.htm</a>
[MLAUTbb]	N.A.: IOleContainer <a href="ms-help://MS.VSCC/MS.MSDNVS/com/oin_oc_68mq.htm">ms-help://MS.VSCC/MS.MSDNVS/com/oin_oc_68mq.htm</a>
[MLAUTbc]	N.A.: IOleClientSite <a href="ms-help://MS.VSCC/MS.MSDNVS/com/oin_oc_5l2d.htm">ms-help://MS.VSCC/MS.MSDNVS/com/oin_oc_5l2d.htm</a>
[MLAUTbd]	N.A.: IFontDisp <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ctin_a2o_7mls.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ctin_a2o_7mls.htm</a>
[MLAUTbe]	N.A.: IRecordInfo Interface

	<a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap12_69wl.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap12_69wl.htm</a>
[MLAUTbf]	N.A.: ISpecifyPropertyPages <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ctin_q2z_49ir.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ctin_q2z_49ir.htm</a>
[MLAUTbg]	N.A.: Dispatch identifier (DISPID) <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obo.htm#_oa96_dispatch_identifier_dispid_">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4obo.htm#_oa96_dispatch_identifier_dispid_</a>
[MLAUTc]	N.A.: Automation <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/autoportal_7l45.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/autoportal_7l45.htm</a>
[MLAUTd]	N.A.: Overview of Automation <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_3r1q.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_3r1q.htm</a>
[MLAUTe]	N.A.: What Is An ActiveX Client? <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7wtb.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7wtb.htm</a>
[MLAUTf]	N.A.: What Is An ActiveX Object? <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_8xr3.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_8xr3.htm</a>
[MLAUTg]	N.A.: Why Expose Objects? <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_4g4f.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_4g4f.htm</a>
[MLAUTHh]	N.A.: How Do Clients and Objects Interact? <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_2bmn.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_2bmn.htm</a>
[MLAUTi]	N.A.: Accessing an Object Through the IDispatch Interface <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7b1h.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7b1h.htm</a>
[MLAUTj]	N.A.: Accessing an Object Through the VTBL <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7ncc.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_7ncc.htm</a>
[MLAUTk]	N.A.: In-Process and Out-of-Process Server Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_5rw3.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap1_5rw3.htm</a>
[MLAUTl]	N.A.: Exposing ActiveX Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9nqr.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9nqr.htm</a>
[MLAUTm]	N.A.: Exposing Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_1d9v.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_1d9v.htm</a>
[MLAUTn]	N.A.: Initializing Exposed Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_07hv.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_07hv.htm</a>
[MLAUTo]	N.A.: OleInitialize <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ofn_oa2k_7w85.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ofn_oa2k_7w85.htm</a>
[MLAUTp]	N.A.: RegisterActiveObject <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap5_2k38.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap5_2k38.htm</a>
[MLAUTq]	N.A.: running object table (ROT) <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4oc2.htm#_oa96_running_object_table_ROT_">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/gloss_4oc2.htm#_oa96_running_object_table_ROT_</a>
[MLAUTr]	N.A.: Implementing Exposed Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_3m2b.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_3m2b.htm</a>
[MLAUTs]	N.A.: Implementing a Class Factory <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9ng9.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9ng9.htm</a>
[MLAUTt]	N.A.: Exposing the Application Object <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_54fo.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_54fo.htm</a>
[MLAUTu]	N.A.: Creating a Registration File <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_6msl.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_6msl.htm</a>
[MLAUTv]	N.A.: Registering the Application <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9smm.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_9smm.htm</a>
[MLAUTw]	N.A.: Registering Classes <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_71o3.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_71o3.htm</a>
[MLAUTx]	N.A.: Releasing OLE and Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_5sz7.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_5sz7.htm</a>
[MLAUTy]	N.A.: OleUninitialize <a href="ms-help://MS.VSCC/MS.MSDNVS/com/ofn_ol2z_13vp.htm">ms-help://MS.VSCC/MS.MSDNVS/com/ofn_ol2z_13vp.htm</a>
[MLAUTz]	N.A.: Retrieving Objects <a href="ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0f77.htm">ms-help://MS.VSCC/MS.MSDNVS/automat/htm_hh2/chap2_0f77.htm</a>
[MLAXCa]	Cluts, Nancy: Microsoft ActiveX Controls Overview <a href="ms-help://MS.VSCC/MS.MSDNVS/dnaxctrl/html/msdn_actxcont.htm">ms-help://MS.VSCC/MS.MSDNVS/dnaxctrl/html/msdn_actxcont.htm</a>
[MLAXCb]	N.A.: ActiveX Control Tutorial

	ms-help://MS.VSCC/MS.MSDNVS/ActiveX/workshop/components/activex/tutorial.htm
[MLAXCc]	N.A.: Introduction to ActiveX Controls. Persistence interfaces ms-help://MS.VSCC/MS.MSDNVS/ActiveX/workshop/components/activex/intro.htm#Persistence_Interface
[MLAXCd]	N.A.: Introduction to ActiveX Controls. Properties, Methods (through IDispatch and other dispinterfaces) ms-help://MS.VSCC/MS.MSDNVS/ActiveX/workshop/components/activex/intro.htm#Properties
[MLAXCe]	N.A.: Introduction to ActiveX Controls. Events ms-help://MS.VSCC/MS.MSDNVS/ActiveX/workshop/components/activex/intro.htm#Events
[MLAXCf]	N.A.: Licensing ActiveX Controls. Design-Time Licensing ms-help://MS.VSCC/MS.MSDNVS/ActiveX/workshop/components/activex/licensing.htm#design_lic
[MLCO+a]	N.A.: COM+ (Components Service) ms-help://MS.VSCC/MS.MSDNVS/cosssdk/htm/complusportal_9o9x.htm
[MLCO+b]	N.A.: What's new in COM+ ms-help://MS.VSCC/MS.MSDNVS/cosssdk/htm/whatsnewcomplus_350z.htm
[MLCO+c]	N.A.: COM+ Programming Overview ms-help://MS.VSCC/MS.MSDNVS/cosssdk/htm/pgintro_programmingoverview_9kjb.htm
[MLCOMa]	N.A.: The Component Object Model. ms-help://MS.VSCC/MS.MSDNVS/com/com_757w.htm
[MLCOMaa]	N.A.: Registering Components ms-help://MS.VSCC/MS.MSDNVS/com/registry_8hrn.htm
[MLCOMab]	N.A.: Classifying Components ms-help://MS.VSCC/MS.MSDNVS/com/registry_5s37.htm
[MLCOMac]	N.A.: Registry Editor ms-help://MS.VSCC/MS.MSDNVS/com/registry_3pgy.htm
[MLCOMad]	N.A.: IClassFactory ms-help://MS.VSCC/MS.MSDNVS/com/cmi_c_9mk9.htm.
[MLCOMae]	N.A.: IClassFactory2 ms-help://MS.VSCC/MS.MSDNVS/com/cmi_c_641e.htm
[MLCOMaf]	N.A.: IConnectionPointContainer ms-help://MS.VSCC/MS.MSDNVS/com/cmi_c_5h2q.htm
[MLCOMag]	N.A.: IDataObject ms-help://MS.VSCC/MS.MSDNVS/com/oin_d_8cl0.htm
[MLCOMah]	N.A.: Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_1xpv.htm
[MLCOMai]	N.A.: OLE Moniker Implementations ms-help://MS.VSCC/MS.MSDNVS/com/monikers_3boz.htm
[MLCOMaj]	N.A.: File Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_5gqc.htm
[MLCOMak]	N.A.: Composite Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_0sj7.htm
[MLCOMal]	N.A.: Item Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_5tgz.htm
[MLCOMam]	N.A.: Anti-Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_2lbn.htm
[MLCOMan]	N.A.: Pointer Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_1fxv.htm
[MLCOMao]	N.A.: Class Monikers ms-help://MS.VSCC/MS.MSDNVS/com/monikers_7r03.htm
[MLCOMb]	N.A.: COM Objects and Interfaces. ms-help://MS.VSCC/MS.MSDNVS/com/com_0alv.htm
[MLCOMc]	N.A.: Interfaces and Interface Implementations. ms-help://MS.VSCC/MS.MSDNVS/com/com_2r5f.htm
[MLCOMd]	N.A.: Interface Pointers and Interfaces.

	<a href="ms-help://MS.VSCC/MS.MSDNVS/com/com_37w3.htm">ms-help://MS.VSCC/MS.MSDNVS/com/com_37w3.htm</a>
[MLCOMe]	N.A.: IUnknown and Interface Inheritance. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/com_9v6t.htm">ms-help://MS.VSCC/MS.MSDNVS/com/com_9v6t.htm</a>
[MLCOMf]	N.A.: IUnknown::QueryInterface. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_7fvp.htm">ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_7fvp.htm</a>
[MLCOMg]	N.A.: Managing Object Lifetimes Through Reference Counting. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/com_63fr.htm">ms-help://MS.VSCC/MS.MSDNVS/com/com_63fr.htm</a>
[MLCOMh]	N.A.: IUnknown::AddRef. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_3rja.htm">ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_3rja.htm</a>
[MLCOMi]	N.A.: IUnknown::Release <a href="ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_59np.htm">ms-help://MS.VSCC/MS.MSDNVS/com/cmi_q2z_59np.htm</a>
[MLCOMj]	N.A.: The COM Library. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/com_1fuh.htm">ms-help://MS.VSCC/MS.MSDNVS/com/com_1fuh.htm</a>
[MLCOMk]	N.A.: Processes, Threads, and Apartments <a href="ms-help://MS.VSCC/MS.MSDNVS/com/aptnthrd_8po3.htm">ms-help://MS.VSCC/MS.MSDNVS/com/aptnthrd_8po3.htm</a>
[MLCOMl]	N.A.: COM Clients and Servers. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_8p2r.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_8p2r.htm</a>
[MLCOMm]	N.A.: Getting a Pointer to an Object. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_1gvo.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_1gvo.htm</a>
[MLCOMn]	N.A.: Class identifier (CLSID) <a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Class_identifier_CLSID_">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Class_identifier_CLSID_</a>
[MLCOMo]	N.A.: COM Class Objects and CLSIDs. <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_2s4z.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_2s4z.htm</a>
[MLCOMp]	N.A.: COM Server Responsibilities <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_99df.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_99df.htm</a>
[MLCOMq]	N.A.: Registering COM Servers <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_05pv.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_05pv.htm</a>
[MLCOMr]	N.A.: GUID Creation and Optimizations <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_7w1f.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_7w1f.htm</a>
[MLCOMs]	N.A.: Inter-Object Communication <a href="ms-help://MS.VSCC/MS.MSDNVS/com/comext_6h7y.htm">ms-help://MS.VSCC/MS.MSDNVS/com/comext_6h7y.htm</a>
[MLCOMt]	N.A.: Defining COM Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/com/custintf_1777.htm">ms-help://MS.VSCC/MS.MSDNVS/com/custintf_1777.htm</a>
[MLCOMu]	N.A.: Interface Design Rules <a href="ms-help://MS.VSCC/MS.MSDNVS/com/custintf_7rn7.htm">ms-help://MS.VSCC/MS.MSDNVS/com/custintf_7rn7.htm</a>
[MLCOMv]	N.A.: Designing Remotable Interfaces <a href="ms-help://MS.VSCC/MS.MSDNVS/com/custintf_31o3.htm">ms-help://MS.VSCC/MS.MSDNVS/com/custintf_31o3.htm</a>
[MLCOMw]	N.A.: Registering COM Applications <a href="ms-help://MS.VSCC/MS.MSDNVS/com/registry_32er.htm">ms-help://MS.VSCC/MS.MSDNVS/com/registry_32er.htm</a>
[MLCOMx]	N.A.: Registry Hierarchy <a href="ms-help://MS.VSCC/MS.MSDNVS/com/registry_9k3d.htm">ms-help://MS.VSCC/MS.MSDNVS/com/registry_9k3d.htm</a>
[MLCOMy]	N.A.: Classes and Servers <a href="ms-help://MS.VSCC/MS.MSDNVS/com/registry_933n.htm">ms-help://MS.VSCC/MS.MSDNVS/com/registry_933n.htm</a>
[MLCOMz]	N.A.: Checking Registration <a href="ms-help://MS.VSCC/MS.MSDNVS/com/registry_5b8u.htm">ms-help://MS.VSCC/MS.MSDNVS/com/registry_5b8u.htm</a>
[MLDCOa]	N.A.: DCOM Technical Overview <a href="ms-help://MS.VSCC/MS.MSDNVS/dndcom/html/msdn_dcomtec.htm">ms-help://MS.VSCC/MS.MSDNVS/dndcom/html/msdn_dcomtec.htm</a>
[MLGLOa]	N.A.: Glossary <a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos</a>
[MLGLOb]	N.A.: Glossary. Container Application <a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Container_application">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Container_application</a>
[MLGLOc]	N.A.: Glossary. Uniform Data Transfer <a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Uniform_data_transfer">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Uniform_data_transfer</a>
[MLGLOd]	N.A.: Glossary. Moniker

	<a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Moniker">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Moniker</a>
[MLGLOe]	N.A.: Glossary. Class factory <a href="ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Class_factory">ms-help://MS.VSCC/MS.MSDNVS/com/newgloss_8xvd.htm#_oleglos_Class_factory</a>
[MLMIDLa]	N.A.: The Header Files <a href="ms-help://MS.VSCC/MS.MSDNVS/midl/mi-cmpil_6ofn.htm">ms-help://MS.VSCC/MS.MSDNVS/midl/mi-cmpil_6ofn.htm</a>
[MLMMGa]	Prosise, Jeff: Windows 2000: Asynchronous Method Calls Eliminate the Wait for COM Clients and Servers. <a href="ms-help://MS.VSCC/MS.MSDNVS/dnmag00/html/async.htm">ms-help://MS.VSCC/MS.MSDNVS/dnmag00/html/async.htm</a>
[MLMMGb]	N.A.: Automating COM+ Administration. <a href="ms-help://MS.VSCC/MS.MSDNVS/dnmag00/html/instincts0900.htm">ms-help://MS.VSCC/MS.MSDNVS/dnmag00/html/instincts0900.htm</a>
[MLMTSa]	N.A.: Microsoft Transaction Server) <a href="ms-help://MS.VSCC/MS.MSDNVS/mts/mtsportal_1lwl.htm">ms-help://MS.VSCC/MS.MSDNVS/mts/mtsportal_1lwl.htm</a>
[MLOLEa]	Brockschmidt, Kraig: What is OLE really about. <a href="ms-help://MS.VSCC/MS.MSDNVS/dnolegen/html/msdn_aboutole.htm">ms-help://MS.VSCC/MS.MSDNVS/dnolegen/html/msdn_aboutole.htm</a>
[MLOLEb]	Brockschmidt, Kraig: OLE Integration Technologies: A Technical Overview <a href="ms-help://MS.VSCC/MS.MSDNVS/dnolegen/html/msdn_ddjole.htm">ms-help://MS.VSCC/MS.MSDNVS/dnolegen/html/msdn_ddjole.htm</a>
[MLSRla]	N.A.: VarEnum Enumeration <a href="ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfSystemRuntimeInteropServicesVarEnumClassTopic.htm">ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfSystemRuntimeInteropServicesVarEnumClassTopic.htm</a>
[MLWMIa]	N.A.: About WMI <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/aboutwmi_1lpl.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/aboutwmi_1lpl.htm</a>
[MLWMib]	N.A.: List All Services on the System <a href="ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic2">ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic2</a>
[MLWMic]	N.A.: Win32_OperatingSystem <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os4_0h7x.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os4_0h7x.htm</a>
[MLWMId]	N.A.: Win32_DiskPartition <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os2_9l7y.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os2_9l7y.htm</a>
[MLWMle]	N.A.: Listing All Drive Partitions with Less Than 20 Percent Free Space <a href="ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic4">ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic4</a>
[MLWMIf]	N.A.: Win32_LogicalDisk <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32hard3_43vv.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32hard3_43vv.htm</a>
[MLWMIg]	N.A.: Launch Notepad Through WMI <a href="ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic8">ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic8</a>
[MLWMih]	N.A.: Reboot a Remote Machine <a href="ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic7">ms-help://MS.VSCC/MS.MSDNVS/dnwmi/html/mngwmi.htm#mngwmi_topic7</a>
[MLWMIi]	N.A.: Win32Shutdown Method in Class Win32_OperatingSystem <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os4_5gdp.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/r_32os4_5gdp.htm</a>
[MLWMIj]	N.A.: Creating a WMI Script Using VBScript <a href="ms-help://MS.VSCC/MS.MSDNVS/wmisdk/us_approg_2ruc.htm">ms-help://MS.VSCC/MS.MSDNVS/wmisdk/us_approg_2ruc.htm</a>
[MLWSHa]	N.A.: Scripts and Automating Windows <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWhatIsWSH.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWhatIsWSH.htm</a>
[MLWSHaa]	N.A.: RemoveNetworkDrive Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthremovenetworkdrive.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthremovenetworkdrive.htm</a>
[MLWSHab]	N.A.: AddWindowsPrinterConnection Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthaddwindowsprinterconnection.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthaddwindowsprinterconnection.htm</a>
[MLWSHac]	N.A.: EnumPrinterConnections Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthenumprinterconnections.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthenumprinterconnections.htm</a>
[MLWSHad]	N.A.: RemovePrinterConnection Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthremoveprinterconnection.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthremoveprinterconnection.htm</a>
[MLWSHae]	N.A.: ComputerName Property <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprocomputername.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprocomputername.htm</a>
[MLWSHaf]	N.A.: UserName Property <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprouername.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprouername.htm</a>
[MLWSHag]	N.A.: UserDomain Property <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprouerdomain.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprouerdomain.htm</a>
[MLWSHah]	N.A.: SendKeys Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthSendKeys.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthSendKeys.htm</a>
[MLWSHai]	N.A.: Run Method

	ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthrun.htm
[MLWSHaj]	N.A.: RegWrite Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthRegWrite.htm
[MLWSHak]	N.A.: RegRead Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthregread.htm
[MLWSHal]	N.A.: RegDelete Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthregdelete.htm
[MLWSHam]	N.A.: WshShortcut Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsObjWshShortcut.htm
[MLWSHan]	N.A.: WshUrlShortcut Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsObjWshUrlShortcut.htm
[MLWSHao]	N.A.: WshSpecialFolders Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsObjWshSpecialFolders.htm
[MLWSHap]	N.A.: SpecialFolders Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprospecialfolders.htm
[MLWSHaq]	N.A.: CreateShortcut Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthCreateShortcut.htm
[MLWSHar]	N.A.: Description Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProDescription.htm
[MLWSHas]	N.A.: Hotkey Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProHotkey.htm
[MLWSHat]	N.A.: IconLocation Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProIconLocation.htm
[MLWSHau]	N.A.: TargetPath Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProTargetPath.htm
[MLWSHav]	N.A.: WindowStyle Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProWindowStyle.htm
[MLWSHaw]	N.A.: WorkingDirectory Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProWorkingDirectory.htm
[MLWSHax]	N.A.: DeleteFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmthDeleteFile.htm
[MLWSHay]	N.A.: WshEnvironment Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsObjWshEnvironment.htm
[MLWSHaz]	N.A.: Environment Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsProEnvironment.htm
[MLWSHb]	N.A.: Hosting Environments and Script Engines ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconHostingEnvironments.htm
[MLWSHba]	N.A.: Item Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsproitem.htm
[MLWSHbb]	N.A.: length Property (WshEnvironment object) ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrlengthpropertywshenvironmentobject.htm
[MLWSHbc]	N.A.: WshScriptExec Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfScriptExecObject.htm
[MLWSHbd]	N.A.: Exec Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfExecMethod.htm
[MLWSHbe]	N.A.: Status Property (WshScriptExec) ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfStatusProperty.htm
[MLWSHbf]	N.A.: CreateObject Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthcreateobject.htm
[MLWSHbg]	N.A.: The FileSystemObject Object Model ms-help://MS.VSCC/MS.MSDNVS/script56/html/sgFileSystemObjectModel.htm
[MLWSHbh]	N.A.: FileSystemObject Objects ms-help://MS.VSCC/MS.MSDNVS/script56/html/sgFileSystemObjects.htm
[MLWSHbi]	N.A.: Programming the FileSystemObject ms-help://MS.VSCC/MS.MSDNVS/script56/html/sgProgrammingFileSystemObject.htm
[MLWSHbj]	N.A.: AvailableSpace Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsproAvailableSpace.htm
[MLWSHbk]	N.A.: GetDrive Method



	ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithGetDrive.htm
[MLWSHbl]	N.A.: GetDriveName Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithGetDriveName.htm
[MLWSHbm]	N.A.: DriveType Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsproDriveType.htm
[MLWSHbn]	N.A.: CreateFolder Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithCreateFolder.htm
[MLWSHbo]	N.A.: OpenTextFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithOpenTextFile.htm
[MLWSHbp]	N.A.: WriteLine Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithWriteLine.htm
[MLWSHbq]	N.A.: Attributes Property ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsproAttributes.htm
[MLWSHbr]	N.A.: GetFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithGetFile.htm
[MLWSHbs]	N.A.: CopyFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithCopyFile.htm
[MLWSHbt]	N.A.: DeleteFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithDeleteFile.htm
[MLWSHbu]	N.A.: DeleteFolder Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithDeleteFolder.htm
[MLWSHbv]	N.A.: Delete Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithDelete.htm
[MLWSHbw]	N.A.: GetFolder Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithGetFolder.htm
[MLWSHbx]	N.A.: Dictionary Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsobjDictionary.htm
[MLWSHby]	N.A.: Add Method (Dictionary) ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithadddictionary.htm
[MLWSHbz]	N.A.: Exists Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/jsmithexists.htm
[MLWSHc]	Cluts, Nancy: What Scripting Is, and Why and When to Use It ms-help://MS.VSCC/MS.MSDNVS/dnsrpt/html/allabout.htm#allabout_topic2
[MLWSHca]	N.A.: Security and Windows Script Host ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconSecurityWindowsScriptHost.htm
[MLWSHcb]	N.A.: CryptoAPI Tools ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWinTrust.htm
[MLWSHcc]	N.A.: Signing a Script ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconSigningScript.htm
[MLWSHcd]	N.A.: Software Restriction Policies ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWinSafer.htm
[MLWSHce]	N.A.: Signature Verification Policy ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconTrustPolicy.htm
[MLWSHcf]	N.A.: SignFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfSignFileMethod.htm
[MLWSHcg]	N.A.: Verifying a Script: ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconverifyingscript.htm
[MLWSHch]	N.A.: VerifyFile Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfverifyfilemethod.htm
[MLWSHci]	N.A.: Sign Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfSignMethod.htm
[MLWSHcj]	N.A.: Verify Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfverifymethod.htm
[MLWSHck]	N.A.: Script Components Overview ms-help://MS.VSCC/MS.MSDNVS/script56/html/letovervw.htm
[MLWSHcl]	N.A.: Introducing Windows Script Components ms-help://MS.VSCC/MS.MSDNVS/script56/html/letintro.htm
[MLWSHcm]	N.A.: How Script Components Work ms-help://MS.VSCC/MS.MSDNVS/script56/html/lethow.htm
[MLWSHcn]	N.A.: Script Component File Contents

	<a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letfilecont.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letfilecont.htm</a>
[MLWSHco]	N.A.: <?component?> <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleqscriptlet.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleqscriptlet.htm</a>
[MLWSHcp]	N.A.: Creating Registration Information <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letcreatereg.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letcreatereg.htm</a>
[MLWSHcq]	N.A.: Registering a Script Component <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letregS.htm#script_componentsmiscunregistering">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letregS.htm#script_componentsmiscunregistering</a>
[MLWSHcr]	N.A.: Creating a Script Component Type Library <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letcreatetypelib.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letcreatetypelib.htm</a>
[MLWSHcs]	N.A.: Exposing Methods <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexpmth.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexpmth.htm</a>
[MLWSHct]	N.A.: Exposing Properties <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexppro.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexppro.htm</a>
[MLWSHcu]	N.A.: Exposing Events <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexpevt.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letexpevt.htm</a>
[MLWSHcv]	N.A.: <property> Element <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleproperty.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleproperty.htm</a>
[MLWSHcw]	N.A.: <method> Element <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleMethod.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleMethod.htm</a>
[MLWSHcx]	N.A.: <event> Element <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleEvent.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/leteleEvent.htm</a>
[MLWSHcy]	N.A.: Using the Script Component Wizard <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letusingwiz.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letusingwiz.htm</a>
[MLWSHcz]	N.A.: ConnectObject Method <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthConnectObject.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthConnectObject.htm</a>
[MLWSHd]	N.A.: Creating Scripts that Can Be Used by WSH <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconCreatingScripts.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconCreatingScripts.htm</a>
[MLWSHda]	N.A.: Start Event <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfstartevent.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfstartevent.htm</a>
[MLWSHdb]	N.A.: Using a Script Component in an Application <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letusingapp.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letusingapp.htm</a>
[MLWSHdc]	N.A.: Registering a Script Component <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/letregs.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/letregs.htm</a>
[MLWSHe]	Cluts, Nancy: Server-Side Scripting <a href="ms-help://MS.VSCC/MS.MSDNVS/dnsrpt/html/allabout.htm#allabout_topic6">ms-help://MS.VSCC/MS.MSDNVS/dnsrpt/html/allabout.htm#allabout_topic6</a>
[MLWSHf]	N.A.: Hosting Environments and Script Engines <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconHostingEnvironments.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconHostingEnvironments.htm</a>
[MLWSHg]	N.A.: Microsoft Windows Script Interfaces-Introduction <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/scripting.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/scripting.htm</a>
[MLWSHh]	N.A.: Types of Script Files <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconScriptFiles.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconScriptFiles.htm</a>
[MLWSHi]	N.A.: Dividing Scripts into Reusable Parts <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconSplittingYourScriptsIntoReusablePieces.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconSplittingYourScriptsIntoReusablePieces.htm</a>
[MLWSHj]	N.A.: Overview of Windows Script Host in Windows 2000 <a href="ms-help://MS.VSCC/MS.MSDNVS/kbwinnt/Source/ntrelease/q232211.htm">ms-help://MS.VSCC/MS.MSDNVS/kbwinnt/Source/ntrelease/q232211.htm</a>
[MLWSHk]	N.A.: Setting and Customizing Script Properties (.wsh) <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsCreateWSH.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsCreateWSH.htm</a>
[MLWSHl]	N.A.: Running Scripts from the Command Prompt <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsRunCscript.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsRunCscript.htm</a>
[MLWSHm]	N.A.: Running Scripts from Windows <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsRunWscript.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsRunWscript.htm</a>
[MLWSHn]	N.A.: Windows Script Host Object Model <a href="ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWindowsScriptHostObjectModel.htm">ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsconWindowsScriptHostObjectModel.htm</a>
[MLWSHo]	N.A.: WshArguments Object

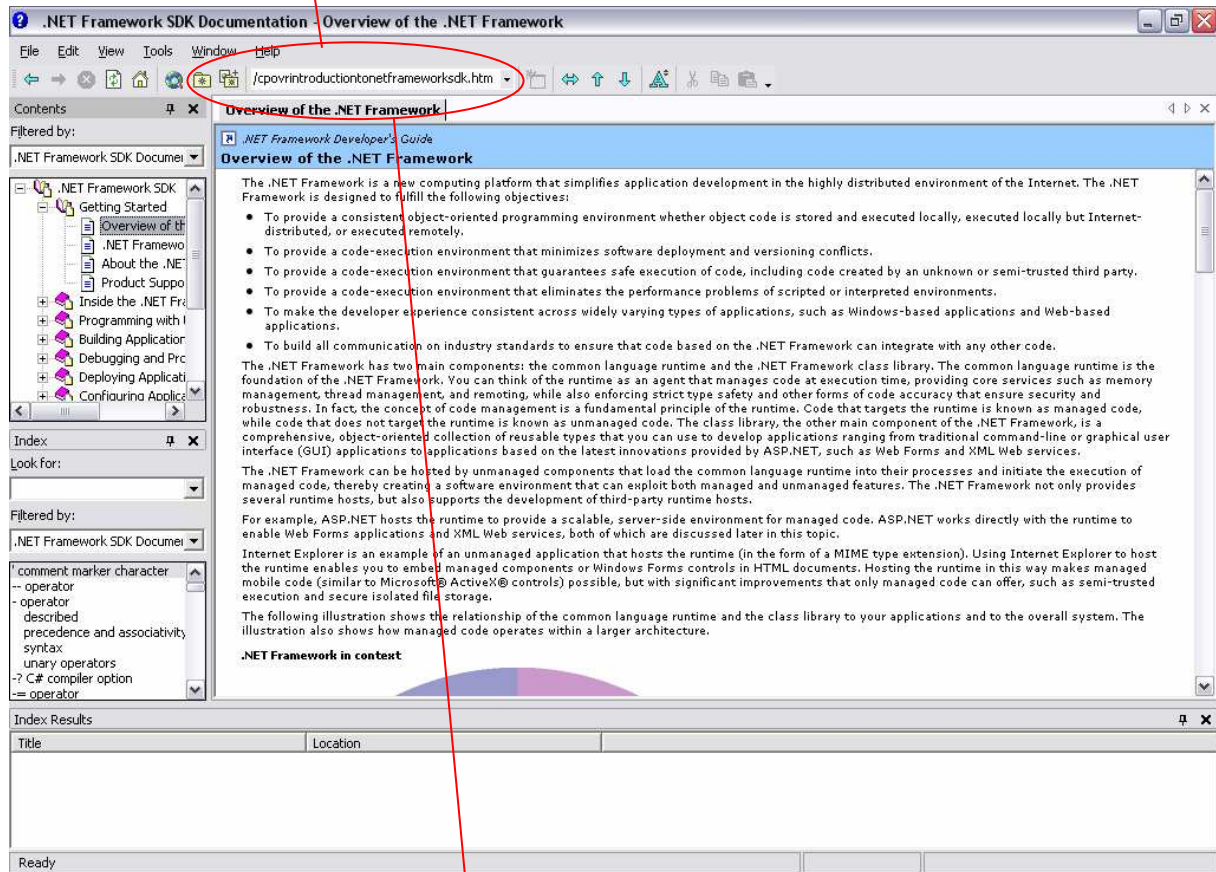
	ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsobjWshArguments.htm
[MLWSHp]	N.A.: Sleep Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsMthSleep.htm
[MLWSHq]	N.A.: length Property (WshArguments object) ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsprolength.htm
[MLWSHr]	N.A.: Count Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfcountmethod.htm
[MLWSHs]	N.A.: Exists Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfexistsmethod.htm
[MLWSHt]	N.A.: WshController Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfControllerObject.htm
[MLWSHu]	N.A.: CreateScript Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfCreateScriptMethod.htm
[MLWSHv]	N.A.: Status Property (WshRemote) ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfstatuspropertyremote.htm
[MLWSHw]	N.A.: Execute Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wslrfexecutemethod.htm
[MLWSHx]	N.A.: WshNetwork Object ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsObjWshNetwork.htm
[MLWSHy]	N.A.: EnumNetworkDrives Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthenumnetworkdrives.htm
[MLWSHz]	N.A.: MapNetworkDrive Method ms-help://MS.VSCC/MS.MSDNVS/script56/html/wsmthmapnetworkdrive.htm
[MLWSIa]	N.A.: Predefined Keys ms-help://MS.VSCC/MS.MSDNVS/sysinfo/regapi_1inn.htm

[MLNETxx]

### Microsoft Library .NET Framework Sources

The **folder specifications** refer to the .NET Documentation<sup>514</sup>. In this case this documentation was installed on the machine as part of the Visual Studio .NET. It is possible to download the documentation from the Microsoft Homepage<sup>515</sup>.

Thereby the Microsoft Document Explorer was installed on the machine and the folder specification is inserted in the folder line.



**Figure 80: Microsoft Document Explorer**

<sup>514</sup> Start-> All Programs->.NET Framework SDK Documentation

<sup>515</sup> <http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/976/msdncompositedoc.xml&frame=true>

The short reference is inserted in brackets. The first two letters are ML for MSDN Library. The next three or four letters describe the chapter. In the phrase [MLNETa] this is .NET. After that there follow one or two small letters for the further distinction. The description of this short reference contains if available the name of the author, if not there occurs "N.A." for "No Author". Then the headline of the referenced text is written and at least the location is printed.

[MLNETa]	N.A.: Overview of the .NET Framework ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpovrintroductiontonetframeworksdk.htm
[MLNETaa]	N.A.: Creating Active Directory Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporilntroductionToActiveDirectoryObjects.htm
[MLNETab]	N.A.: Active Directory Technology Backgrounder ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconWhatYouNeedToKnowAboutActiveDirectoryADSI.htm
[MLNETac]	N.A.: Generating and Compiling Source Code Dynamically in Multiple Languages ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcongeneratingcompilingsourcecodedynamicallyinmultiplelanguages.htm
[MLNETad]	N.A.: Using the CodeDOM ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconusingcodedom.htm
[MLNETae]	N.A.: Generating Source Code and Compiling a Program from a CodeDOM Graph ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcongeneratingsourcecodecompilingprogramfromcodedomgraph.htm
[MLNETaf]	N.A.: Developing Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcomponentprogrammingessentials.htm
[MLNETag]	N.A.: Class vs. Component vs. Control ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconclassvscomponentvscontrol.htm
[MLNETah]	N.A.: Design-Time Attributes for Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcondesign-timeattributesforcomponents.htm
[MLNETai]	N.A.: Developing World-Ready Applications ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcondesigningglobalapplications.htm
[MLNETaj]	N.A.: Developing World-Ready Applications Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcondevelopingworld-readyapplicationsoverview.htm
[MLNETak]	N.A.: Localizability ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconlocalizability.htm
[MLNETal]	N.A.: Including Asynchronous Calls ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconasynchronousprogramming.htm

[MLNETam]	N.A.: Asynchronous Programming Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpovrasynchronousprogrammingoverview.htm
[MLNETan]	N.A.: Creating Messaging Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiIntroductionToMessageBasedFrameworkFeatures.htm
[MLNETao]	N.A.: Introduction to Messaging ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconIntroductionToMessagingInVisualStudio.htm
[MLNETap]	N.A.: Message Queues and Messaging Technology Backgrounder ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconWhatYouNeedToKnowAboutMessageQueues.htm
[MLNETaq]	N.A.: Managing Applications Using WMI ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconmanagingapplicationsusingwmi.htm
[MLNETar]	N.A.: Using WMI with the .NET Framework ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconusingwmiwiththenetframework.htm
[MLNETas]	N.A.: Processing Transactions ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconprocessingtransactions.htm
[MLNETat]	N.A.: Transaction Processing Fundamentals ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcontransactionprocessingfundamentals.htm
[MLNETau]	N.A.: Transaction Models ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcontransactionmodels.htm
[MLNETav]	N.A.: Manual Transactions ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconmanualtransactions.htm
[MLNETaw]	N.A.: Automatic Transactions ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconautomatictransactions.htm
[MLNETax]	N.A.: Securing Applications ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsecuringyourapplication.htm
[MLNETay]	N.A.: Key Security Concepts ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconkeyconceptsinscurity.htm
[MLNETaz]	N.A.: Permissions ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconpermissions.htm
[MLNETb]	N.A.: Common Language Runtime Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcommonlanguageruntimeoverview.htm
[MLNETba]	N.A.: Type Safety and Security ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpcontypesafetysecurity.htm
[MLNETbb]	N.A.: Security Policy ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsecuritypolicy.htm
[MLNETbc]	N.A.: Principal ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconprincipal.htm

[MLNETbd]	N.A.: Authentication ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconauthentication.htm
[MLNETbe]	N.A.: Authorization ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconauthorization.htm
[MLNETbf]	N.A.: Code Access Security ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcodeaccesssecurity.htm
[MLNETbg]	N.A.: Role-Based Security ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconrole-basedsecurity.htm
[MLNETbga]	N.A.: Security Policy Management ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsecuritypolicymanagement.htm
[MLNETbh]	N.A.: Introduction to Role-Based Security ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconintroductiontorole-basedsecurity.htm
[MLNETbha]	N.A.: Security Policy Model ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsecuritypolicymodel.htm
[MLNETbi]	N.A.: Cryptography Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcryptographyoverview.htm
[MLNETbia]	N.A.: Security Tools ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsecuritytools.htm
[MLNETbj]	N.A.: .NET Framework Cryptography Model ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconnetframeworkcryptographymodel.htm
[MLNETbja]	N.A.: Creating System Monitoring Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiIntroductionToMonitoringFrameworkFeatures.htm
[MLNETbk]	N.A.: Monitoring Performance Thresholds ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiInstrumentingPerformanceThresholdsOnServer.htm
[MLNETbl]	N.A.: Logging Application, Server, and Security Events ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiLoggingNTApplicationServerSecurityEvents.htm
[MLNETbm]	N.A.: Monitoring Windows Services ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiMonitoringWindowsServices.htm
[MLNETbn]	N.A.: Monitoring and Managing Windows Processes ms-help://MS.NETFrameworkSDK/cpguidenf/html/cporiMonitoringManagingProcesses.htm
[MLNETbo]	N.A.: Creating ASP.NET Web Applications ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcreatingaspwebapplications.htm
[MLNETbp]	N.A.: Introduction to ASP.NET ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconintroductiontoasp.htm
[MLNETbq]	N.A.: Introduction to ASP.NET ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconintroductiontoasp.htm
[MLNETbr]	N.A.: Introduction to Windows Service Applications ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconIntroductionToNTServiceApplicatio

	ns.htm
[MLNETbs]	N.A.: Introduction to Windows Forms ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconIntroductionToWFCForms.htm
[MLNETbt]	N.A.: Enhancing Design-Time Support ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconenhancingdesign-timesupport.htm
[MLNETbu]	N.A.: Design-Time Architecture ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconnetframeworkdesign-timearchitecture.htm
[MLNETbv]	N.A.: Exposing .NET Framework Components to COM ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconexposingnetframeworkcomponentstocom.htm
[MLNETbw]	N.A.: Qualifying .NET Types for Interoperation ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconqualifyingnettypesforinteroperation.htm
[MLNETbx]	N.A.: Packaging an Assembly for COM ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconpackagingassemblyforcom.htm
[MLNETby]	N.A.: Assembly Registration Tool (Regasm.exe) ms-help://MS.NETFrameworkSDK/cptools/html/cpgrfassemblyregistrationtoolregasmexe.htm
[MLNETbz]	N.A.: Registering Assemblies with COM ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconregisteringassemblieswithcom.htm
[MLNETc]	N.A.: Assemblies ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconassemblies.htm
[MLNETca]	N.A.: Object Class ms-help://MS.NETFrameworkSDK/cpref/html/frlrfsystemobjectclasstopic.htm
[MLNETcb]	N.A.: Random Members ms-help://MS.NETFrameworkSDK/cpref/html/frlrfsystemrandommemberstopic.htm
[MLNETcc]	N.A.: Advanced COM Interop ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconadvancedcominterop.htm
[MLNETcd]	N.A.: COM Wrappers ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcomwrappers.htm
[MLNETce]	N.A.: COM Callable Wrapper ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcomcallablewrapper.htm
[MLNETcf]	N.A.: Simulating COM Interfaces ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsimulatingcominterfaces.htm
[MLNETcg]	N.A.: Customizing Standard Wrappers ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcustomizingstandardwrappers.htm
[MLNETch]	N.A.: Building from the Command Line ms-help://MS.NETFrameworkSDK/cscomp/html/vcgrfBuildingFromCommandLine.htm
[MLNETci]	N.A.: /target:library (Create a Code Library) ms-help://MS.NETFrameworkSDK/cscomp/html/vcrefdllcreatedllfile.htm
[MLNETcj]	N.A.: Choosing a Compiler ms-help://MS.VSCC/MS.MSDNVS/cpguide/html/cpconsourcecodelanguage.htm



[MLNETck]	N.A.: Compiling to MSIL ms-help://MS.VSCC/MS.MSDNVS/cpguide/html/cpconmicrosoftintermediatelanguagesil.htm
[MLNETcl]	N.A.: portable executable (PE) file ms-help://MS.VSCC/MS.MSDNVS/Netstart/html/cpglop.htm
[MLNETcm]	N.A.: Assembly Versions ms-help://MS.VSCC/MS.MSDNVS/sbscs/sidebyside_57n7.htm
[MLNETcn]	N.A.: strong name. ms-help://MS.VSCC/MS.MSDNVS/Netstart/html/cpglos.htm
[MLNETco]	N.A.: Licensing Components and Controls ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconlicensingcomponentscontrols.htm
[MLNETcp]	N.A.: The Global.asax File ms-help://MS.VSCC/MS.MSDNVS/cpguide/html/cpcontheglobalasaxfile.htm
[MLNETcq]	N.A.: Visual Basic Language Tour ms-help://MS.NETFrameworkSDK/vblr7net/html/vaconVisualBasicLanguageTour.htm
[MLNETcr]	N.A.: Programming Element Support Changes Summary ms-help://MS.NETFrameworkSDK/vblr7net/html/vaconProgrammingElementsChangesInVB7.htm
[MLNETd]	N.A.: Assemblies Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconassembliesoverview.htm
[MLNETe]	N.A.: Side-by-Side Execution ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconsidebysideexecution.htm
[MLNETf]	N.A.: Assembly Benefits ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconwhyuseassemblies.htm
[MLNETg]	N.A.: Assembly Contents ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcontentsofassembly.htm
[MLNETh]	N.A.: Assembly Manifest ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconassemblymanifest.htm
[MLNETi]	N.A.: Metadata and Self-Describing Components ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconmetadataselfdescribingcomponents.htm
[MLNETj]	N.A.: Metadata Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconmetadataoverview.htm
[MLNETk]	N.A.: Run-Time Use of Metadata ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconruntimeuseofmetadata.htm
[MLNETl]	N.A.: Metadata and the PE File Structure ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconmetadatapefilestructure.htm
[MLNETm]	N.A.: Cross-Language Interoperability ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcommonlanguagespecification.htm
[MLNETn]	N.A.: Language Interoperability Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconcross-languageinteroperability.htm
[MLNETo]	N.A.: What is the Common Language Specification? ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconwhatiscommonlanguagespecificati

	on.htm
[MLNETp]	N.A.: Writing CLS-Compliant Code ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconwritingcls-compliantcode.htm
[MLNETq]	N.A.: Introduction to the .NET Framework Class Library ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconthenetframeworkclasslibrary.htm
[MLNETr]	N.A.: Accessing Data with ADO.NET ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconaccessingdatawithadonet.htm
[MLNETs]	N.A. Overview of ADO.NET: ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconoverviewofadonet.htm
[MLNETt]	N.A. Design Goals for ADO.NET: ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconwhyadonet.htm
[MLNETu]	N.A.: ADO.NET Architecture ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconadonetarchitecture.htm
[MLNETv]	N.A.: Choosing Communication Options in .NET ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconchoosingcommunicationoptionsinn et.htm
[MLNETw]	N.A.: .NET Remoting Overview ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconnetremotingoverview.htm
[MLNETx]	N.A.: Introducing Pluggable Protocols ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconintroducingpluggableprotocols.htm
[MLNETy]	N.A.: Accessing the Internet ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconaccessingInternet.htm
[MLNETz]	N.A.: Introduction to Active Directory Objects ms-help://MS.NETFrameworkSDK/cpguidenf/html/cpconIntroductionToADSIObjectsInVisua lStudio.htm

### Other Sources:

[AP02]	AP: Mindelheimer Zeitung. Revolutionen lassen noch auf sich warten, p.19. 02-12-31
[Ar03]	Archmatic: ARCHmatic-Glossar E. <a href="http://www.glossar.de/glossar/amglos_e.htm">http://www.glossar.de/glossar/amglos_e.htm</a> , retrieval 03-03-07
[BI02]	Blakely, Beth: Microsoft gegen Sun: Krieg in der Web-Services-Arena <a href="http://techupdate.zdnet.de/story/0,,s2109366,00.html">http://techupdate.zdnet.de/story/0,,s2109366,00.html</a> , 2002, retrieval 03-03-17
[Br02]	Brüssau, Kai: Komponenten bei der Softwareerstellung. In: WISU. Das Wirtschaftsstudium 10/02 (2002), p.1216.
[BuHe03]	Bucher, Martin; Herlicska, Michael et. Al.: Tourenplanung. Seminarpaper, University of Augsburg, 2003. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/G-08-Tourenplanung.zip">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/G-08-Tourenplanung.zip</a> , retrieval 03-03-01
[CI01]	Clinick, Andrew: Providing a Secure eXperience <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting10082001.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting10082001.asp</a> , 2001, retrieval 03-03-13
[DI03]	Newsgroup Develpoersindex: WSHController. <a href="http://www.developersdex.com/asp/message.asp?p=593&amp;ID=%3COuLkrGm2CHA%2E2264%40TK2MSFTNGP11%2Ephx%2Egbl%3E">http://www.developersdex.com/asp/message.asp?p=593&amp;ID=%3COuLkrGm2CHA%2E2264%40TK2MSFTNGP11%2Ephx%2Egbl%3E</a> , 2003, retrieval 03-02-28
[DI03a]	Newsgroup Develpoersindex: WSHController. <a href="http://www.developersdex.com/asp/message.asp?p=593&amp;ID=%3CeGDE5r03CHA%2E1640%40TK2MSFTNGP10%2Ephx%2Egbl%3E">http://www.developersdex.com/asp/message.asp?p=593&amp;ID=%3CeGDE5r03CHA%2E1640%40TK2MSFTNGP10%2Ephx%2Egbl%3E</a> , 2003, retrieval 03-03-12
[DI03b]	Newsgroup Develporsindex: Accessing .NET Framework classes as COM objects with or without "dm.net Moniker" <a href="http://www.developersdex.com/vb/message.asp?p=2920&amp;ID=%3COePO45NwCHA%2E2124%40TK2MSFTNGP11%3E">http://www.developersdex.com/vb/message.asp?p=2920&amp;ID=%3COePO45NwCHA%2E2124%40TK2MSFTNGP11%3E</a> , 2003, retrieval 03-03-14
[DM00]	Developerator: The dm.net COM Moniker <a href="http://staff.develop.com/jasonw/clr/readme.htm">http://staff.develop.com/jasonw/clr/readme.htm</a> , 2000, retrieval 03-03-14
[Doe02]	E-Mail from Stefan Dörsam. 02-12-11. Content: DCOM and COM+
[Doe02a]	E-Mail from Stefan Dörsam. 02-12-02. Content: .RxMessageBox hidden
[Doe03]	E-Mail from Stefan Dörsam. 03-02-28. Content: Problems with events of OLE objects.
[Doe03a]	E-Mail from Stefan Dörsam. 03-02-17. Content: MS Agent Technology and Events
[Doe03b]	E-Mail from Stefan Dörsam. 03-01-16. Content: MS.NET und Object Rexx
[Doe03c]	E-Mail from Stefan Dörsam. 03-03-10. Content: MS Speech SDK 5.1 - Events
[Doe03d]	E-Mail from Stefan Dörsam. 03-03-26. Content: Speech Recognition
[DPA03]	DPA EB: "Web Services" hauchen Internet-Wirtschaft neuen Atem ein <a href="http://www.pcwelt.de/news/internet/29697/index.html">http://www.pcwelt.de/news/internet/29697/index.html</a> , 2003, retrieval 03-03-21
[DPA03a]	DPA EB:Anwendungen für den Endverbraucher? <a href="http://www.pcwelt.de/news/internet/29697/2.html">http://www.pcwelt.de/news/internet/29697/2.html</a> , 2003, retrieval 03-03-21
[Dr03]	Driver, Mark: This Isn't Your Father's Visual Studio. <a href="http://www.fawcette.com/dotnetmag/2002_03/magazine/columns/strategy/default_pf.asp">http://www.fawcette.com/dotnetmag/2002_03/magazine/columns/strategy/default_pf.asp</a> , retrieval 03-03-21
[En01]	Engelhausen, Jan: Rexx on Windows. The Power of OLE/ActiveX Automation. IBM Corp. <a href="http://www.share.org/proceedings/sh96/data/S8312.PDF">http://www.share.org/proceedings/sh96/data/S8312.PDF</a> , 2001, retrieval 03-03-19
[En02]	E-Mail from Jan Engelhausen. 02-01-28. Content: Embedding of Object Rexx in HTML.
[En03]	Engelhausen, Jan: Über mich: <a href="http://home.arcor.de/jan.engelhausen/refer.html">http://home.arcor.de/jan.engelhausen/refer.html</a> , retrieval 03-03-21
[Es02]	Espostio, Dino: Windows Script Host 5.6 Boasts Windows XP Integration, Security, New Object Model <a href="http://msdn.microsoft.com/msdnmag/issues/02/05/wsh/default.aspx">http://msdn.microsoft.com/msdnmag/issues/02/05/wsh/default.aspx</a> , 2002, retrieval 03-03-12
[Fla02]	Rony G. Flatscher: Automatisierung von Windows Anwendungen. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_01.pdf">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_01.pdf</a> , retrieval 02-11-12
[Fla02a]	Rony G. Flatscher: Automatisierung von Windows Anwendungen. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_07.pdf">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_07.pdf</a> , retrieval 02-11-12
[Fla02b]	Rony G. Flatscher: Automatisierung von Windows Anwendungen. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_08.pdf">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_08.pdf</a> , retrieval 02-11-12
[Fla02c]	Rony G. Flatscher: "Overview of the Document Object Model (DOM) a.k.a.DHTML Under Windows"

	<a href="http://wi.wu-wien.ac.at/rgf/rexx/orx13/2002_DOM.pdf">http://wi.wu-wien.ac.at/rgf/rexx/orx13/2002_DOM.pdf</a> , 2002, retrieval November 2002
[Fla02d]	Rony G. Flatscher: "Applying the Object REXXWindows Scripting Engine with Windows Scripting Host " <a href="http://wi.wu-wien.ac.at/rgf/rexx/orx13/2002_WSH.pdf">http://wi.wu-wien.ac.at/rgf/rexx/orx13/2002_WSH.pdf</a> , 2002, retrieval 02-11-19
[Fla02e]	E-Mail from Prof. Rony G. Flatscher. 02-12-2. Content: .rxs Format
[Fla03]	Conversation with Prof.Dr.Rony G. Flatscher. 03-03-05. Content: Discussion of master thesis.
[Fla03a]	E-Mail from Prof. Rony G. Flatscher. 03-02-10. Content: MS Agent, MS.NET, MS Speech, OREXXOLE.CLS
[Fla03b]	E-Mail from Prof. Rony G. Flatscher. 03-01-16. Content: New OREXXOLE.DLL
[Fla03c]	Rony G. Flatscher: Read.me. Part of <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/things/agent_upd_ole.zip">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/things/agent_upd_ole.zip</a> , 2003, retrieval 03-01-20
[Fla03d]	E-Mail from Prof. Rony G. Flatscher. 03-03-28. Content: .local und WSC
[Fla03e]	E-Mail from Prof. Rony G. Flatscher. 03-03-29. Content: Correcting of master thesis
[Fla03f]	E-Mail from Prof. Rony G. Flatscher. 03-03-29. Content: Embedding
[Ge00]	Gellersen, Hans-W.: Ubiquitous Computing. <a href="http://www.teco.edu/lehre/ubiqws0001/skript/04.pdf">http://www.teco.edu/lehre/ubiqws0001/skript/04.pdf</a> , 2000, retrieval 03-03-15
[Ge03]	Geocity: Windows Scripting Host is more powerful than AppleScript <a href="http://www.geocities.com/siliconvalley/sector/9295/pc-advantage/advantage-scripting.html">http://www.geocities.com/siliconvalley/sector/9295/pc-advantage/advantage-scripting.html</a> , retrieval 03-03-29
[He02]	Helmecke, Florian: Remote Control for MS Word and MS Excel with Object Rexx. Seminarpaper, University of Augsburg, 2002.
[HeLu03]	Herzog, Thomas; Lutz, Niko et. Al.: Döner Dome Restaurants. Seminarpaper, University of Augsburg, 2003. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/Autowin_Abschlussarbeit_Team3.zip">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/Autowin_Abschlussarbeit_Team3.zip</a> , retrieval 03-03-01
[IBM01]	IBM: Object Rexx Reference. 1. Ed.. IBM Corp. 2001 (Location: ...\\Windows\\Version 2.1\\books\\RexxRef.PDF on the Object Rexx CD)
[IBM01a]	IBM: REGISTRY.REX. IBM Corp. 2001 (Location: ...\\ObjREXX\\SAMPLES)
[IBM01b]	IBM: Object Rexx Programming Guide. 1. Ed.. IBM Corp. 2001 (Location: ...\\Windows\\Version 2.1\\books\\RexxPG.PDF on the Object Rexx CD)
[IBM01b]	IBM: SAMPLE12.rex. IBM Corp. 2001 (Location: ...\\ObjREXX\\SAMPLES\\OLE\\APPS\\
[IBM01c]	IBM: USEWMGR.REX. IBM Corp. 2001 (Location: ...\\ObjREXX\\SAMPLES)
[IBM01c]	IBM: Object Rexx Programming Guide 1. Ed.. IBM Corp. 2001 (Location: ...\\Windows\\Version 2.1\\books\\RexxPg.PDF on the Object Rexx CD)
[IBM02]	IBM: Writing a COM object in Object REXX. <a href="http://www-1.ibm.com/support/docview.wss?rs=22&amp;context=SS8PLL&amp;uid=swg21008846">http://www-1.ibm.com/support/docview.wss?rs=22&amp;context=SS8PLL&amp;uid=swg21008846</a> , 04.08.2002, retrieval 03-02-24
[IBM02a]	IBM: From another script language to Object REXX. (activex_convert.pdf). <a href="http://www-1.ibm.com/support/retmgr.wss?rs=22&amp;rt=0&amp;org=SW&amp;doc=1044475">http://www-1.ibm.com/support/retmgr.wss?rs=22&amp;rt=0&amp;org=SW&amp;doc=1044475</a> , retrieval 02-03-10
[IBM02b]	IBM: OLEINFO Help File. Part of Object Rexx installation. ...\\ObjREXX\\SAMPLES\\OLE\\OLEINFO\\HELP.TXT, 2002
[IBM02c]	IBM: Terminating a process <a href="http://www-1.ibm.com/support/docview.wss?uid=swg21039851">http://www-1.ibm.com/support/docview.wss?uid=swg21039851</a> , 2002, retrieval 03-01-13
[IBM03]	IBM: Where do we stand today, traditional or classic rexx. <a href="http://www-3.ibm.com/software/ad/obj-rexx/section2.html">http://www-3.ibm.com/software/ad/obj-rexx/section2.html</a> , retrieval 03-02-28
[IBM03]	IBM: IBM WebSphere SDK for Web Services (WSDK) <a href="http://www-106.ibm.com/developerworks/webservices/wsdk/wsdkfaqs.html">http://www-106.ibm.com/developerworks/webservices/wsdk/wsdkfaqs.html</a> , retrieval 03-03-17
[Ku02]	Kurzweil, Andreas: Windows Scripting Host - Definieren von COM-Interfaces (Object Rexx) <a href="http://www.wu-wien.ac.at/usr/h96a/h9651692/wi_sem.pdf">http://www.wu-wien.ac.at/usr/h96a/h9651692/wi_sem.pdf</a> , 2002, retrieval 03-03-13
[MeSc03]	Mengele, Markus; Schuhwerket, Christoph. Al.: High Value Customers consultancy.

	Seminarpaper, University of Augsburg, 2003. <a href="http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/AutoWin%20Team9.zip">http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/aufgaben/Projekte/AutoWin%20Team9.zip</a> , retrieval 03-03-01
[MM03]	McMullan, Andy:.NET Framework Frequently Asked Questions <a href="http://www.eponymous.eclipse.co.uk/dotnetfaq.htm">http://www.eponymous.eclipse.co.uk/dotnetfaq.htm</a> , 2003, retrieval 03-03-14
[Mo98]	Moss, Julian: Understanding the Windows Script Host. <a href="http://www.itp-journals.com/sasample/T1205.pdf">http://www.itp-journals.com/sasample/T1205.pdf</a> , 1998, retrieval 03-03-14
[MS01]	Microsoft: Constant names <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deconConstantNames.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deconConstantNames.asp</a> , 2001, retrieval 03-03-08
[MS01a]	Microsoft: Microsoft Agent Downloads <a href="http://microsoft.com/products/msagent/downloads.htm">http://microsoft.com/products/msagent/downloads.htm</a> , 2001, retrieval 03-02-14
[MS01b]	N.A.: Debugging Scripts <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdbug/Html/sdbug_2.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdbug/Html/sdbug_2.asp</a> , 2001, retrieval 03-03-12
[MS01c]	Microsoft: XML Web Services Basics <a href="http://msdn.microsoft.com/library/?url=/library/en-us/dnwebsrv/html/webservbasics.asp?frame=true">http://msdn.microsoft.com/library/?url=/library/en-us/dnwebsrv/html/webservbasics.asp?frame=true</a> , 2001, retrieval 03-03-14
[MS02]	Microsoft: What Is Speech Technology? <a href="http://www.microsoft.com/speech/evaluation/techover/">http://www.microsoft.com/speech/evaluation/techover/</a> , 2002, retrieval 03-02-04
[MS02a]	Microsoft:What .NET Means for Users <a href="http://www.microsoft.com/net/basics/faq.asp">http://www.microsoft.com/net/basics/faq.asp</a> , 2002, retrieval 03-03-14
[MS02b]	N.A.:What Is Microsoft .NET? Microsoft: <a href="http://msdn.microsoft.com/netframework/productinfo/overview/default.asp">http://msdn.microsoft.com/netframework/productinfo/overview/default.asp</a> , 2002, retrieval 03-03-14
[MS02c]	Microsoft: .NET Passport <a href="http://www.microsoft.com/netservices/passport/overview.asp">http://www.microsoft.com/netservices/passport/overview.asp</a> , 2002, retrieval 02-12-31
[MS02d]	Microsoft: :NET Passport Review Guide <a href="http://www.microsoft.com/net/downloads/passport_reviewguide.doc">http://www.microsoft.com/net/downloads/passport_reviewguide.doc</a> , 2002, retrieval 02-12-31
[MS02e]	Microsoft:.NET Compact Framework Overview. <a href="http://msdn.microsoft.com/vstudio/device/compactfx.asp">http://msdn.microsoft.com/vstudio/device/compactfx.asp</a> , 2002, retrieval 03-03-17
[MS03]	Microsoft: OLE Background: Containers and Servers <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_ole_background.3a_.containers_and_servers.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_ole_background.3a_.containers_and_servers.asp</a> , retrieval 2003-01-06
[MS03a]	Microsoft: Visible Property of MS Word <a href="http://msdn.microsoft.com/library/en-us/vbawd10/html/woproVisible.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbawd10/html/woproVisible.asp?frame=true</a> , 2003, 03-03-08
[MS03b]	Microsoft: Visible Property of MS Internet Explorer <a href="http://msdn.microsoft.com/library/default.asp?url=/workshop/browser/webbrowser/reference/properties/visible.asp">http://msdn.microsoft.com/library/default.asp?url=/workshop/browser/webbrowser/reference/properties/visible.asp</a> , 2003, retrieval 03-03-08
[MS03c]	Microsoft: WorkSheet object <a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlobjWorksheet.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlobjWorksheet.asp?frame=true</a> , 2003, retrieval 03-03-08
[MS03d]	Microsoft: Visible Property of MS Excel <a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlproVisible.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlproVisible.asp?frame=true</a> , 2003, 03-03-09
[MS03e]	Microsoft: EntireColumn of MS Excel <a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlproEntireColumn.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlproEntireColumn.asp?frame=true</a> , 2003, retrieval 03-03-09
[MS03f]	Microsoft: Range Collection of Excel <a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlobjRange.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlobjRange.asp?frame=true</a> , 2003, retrieval 03-03-09
[MS03g]	Microsoft: Autofit of MS Excel <a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlmthAutoFit.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlmthAutoFit.asp?frame=true</a> , 2003, retrieval 03-03-09
[MS03h]	Microsoft: Select of MS Excel

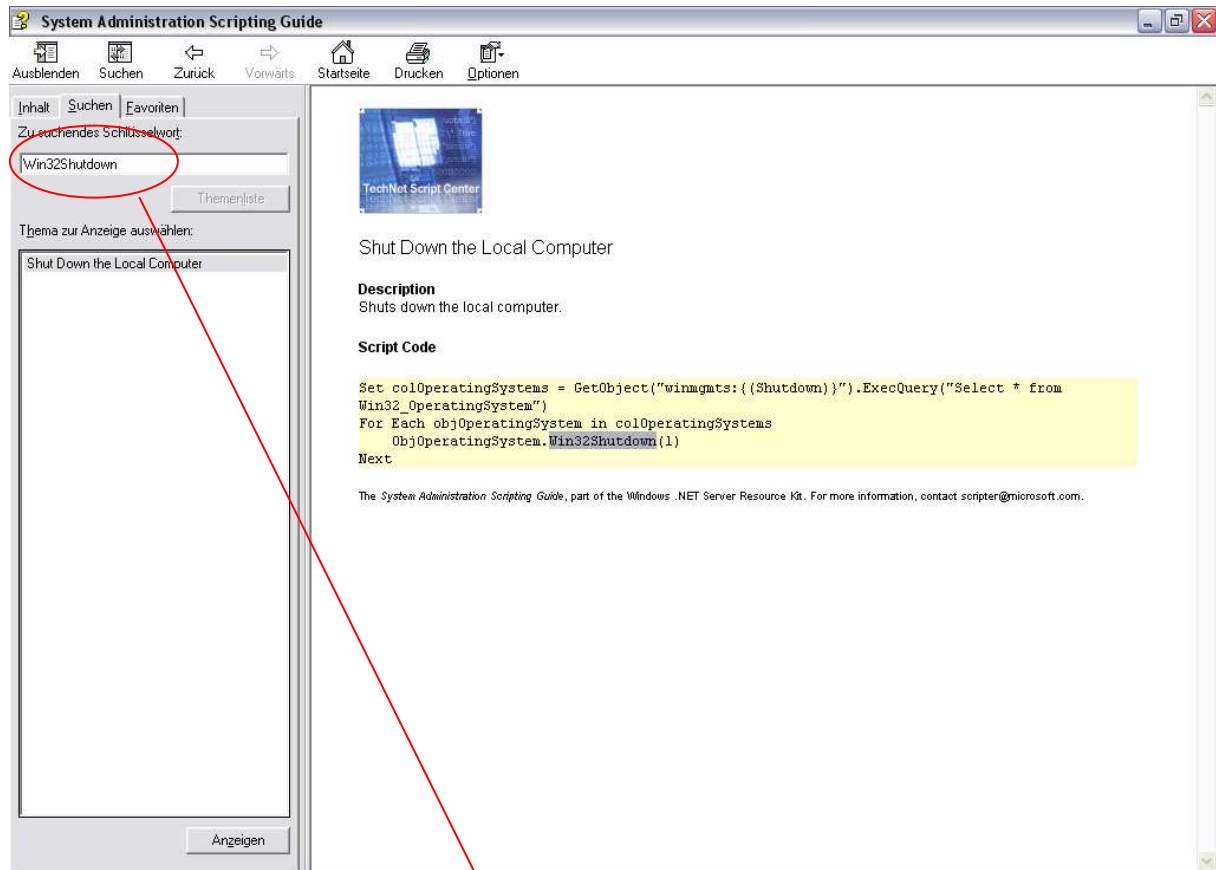
	<a href="http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlmthSelect.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbaxl10/html/xlmthSelect.asp?frame=true</a> , 2003, retrieval 03-03-09
[MS03i]	Microsoft: Checkspelling of MS Word <a href="http://msdn.microsoft.com/library/en-us/vbawd10/html/womthCheckSpelling.asp?frame=true">http://msdn.microsoft.com/library/en-us/vbawd10/html/womthCheckSpelling.asp?frame=true</a> , 2003, retrieval 03-01-11
[MS03j]	Microsoft: MkCEVoice <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcesapi/htm/cesamMkCEVoice.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcesapi/htm/cesamMkCEVoice.asp</a> , 2003, retrieval 03-02-04
[MS03k]	Microsoft: Setting up Remote WSH <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/wstsksetupofremotewsh.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/wstsksetupofremotewsh.asp</a> , 2003, retrieval 03-03-12
[MS03l]	Microsoft: Running Scripts on Remote Computers <a href="http://www.microsoft.com/technet/treeview/default.asp?url=/technet/scriptcenter/scrguide/sas_wsh_wwgn.asp">http://www.microsoft.com/technet/treeview/default.asp?url=/technet/scriptcenter/scrguide/sas_wsh_wwgn.asp</a> , 2003, retrieval 03-03-12
[MS03m]	Microsoft: Defining the Basic Elements of .NET <a href="http://www.microsoft.com/net/basics/whatis.asp">http://www.microsoft.com/net/basics/whatis.asp</a> , 2003, retrieval 03-03-14
[MS03n]	Microsoft: .NET Passport <a href="http://www.microsoft.com/net/services/passport/">http://www.microsoft.com/net/services/passport/</a> , 2003, retrieval 03-03-14
[MS96]	Redmond III, Frank: Client/Server ActiveX: Activate the Internet with ODBC <a href="http://www.microsoft.com/mind/0796/ODBC/ODBC.asp">http://www.microsoft.com/mind/0796/ODBC/ODBC.asp</a> , Sept. 1996, retrieval 2003-3-1
[MS98]	Microsoft: The OLE/COM Object Viewer <a href="http://www.microsoft.com/com/resources/oleview.asp">http://www.microsoft.com/com/resources/oleview.asp</a> , 6.11.1998, retrieval 03-02-25
[Mü01]	Münz, Stefan: SelfHTML. HTML als Auszeichnungssprache. <a href="http://selfhtml.teamone.de/intro/technologien/html.htm#auszeichnungssprache">http://selfhtml.teamone.de/intro/technologien/html.htm#auszeichnungssprache</a> , 2001, retrieval 03-03-10
[Mü01a]	Münz, Stefan: SelfHTML. Kommentare und Credits. <a href="http://selfhtml.teamone.de/html/allgemein/kommentare.htm#nicht_angezeigt">http://selfhtml.teamone.de/html/allgemein/kommentare.htm#nicht_angezeigt</a> , 2001, retrieval 03-03-10
[Mü01b]	Münz, Stefan: SelfHTML. Das Document Object Model (DOM). <a href="http://selfhtml.teamone.de/dhtml/modelle/dom.htm#allgemeines">http://selfhtml.teamone.de/dhtml/modelle/dom.htm#allgemeines</a> , 2001, retrieval 03-03-10
[Mue02]	Mueller, Dietmar: Apple startet eigene Web Service-Initiative <a href="http://news.zdnet.de/story/0,,t101-s2119357,00.html">http://news.zdnet.de/story/0,,t101-s2119357,00.html</a> , 2002, retrieval 03-03-17
[NGS03]	Newsgroup microsoft.public.speech_tech: L&H TTS engine of MS Agent and SDK 5.1 <a href="http://communities.microsoft.com/newsgroups/previewFrame.asp?ICP=cddgall&amp;sLCID=US&amp;sgroupURL=microsoft.public.speech_tech&amp;sMessageID=%253C3e504db8%25241@news.microsoft.com%253E">http://communities.microsoft.com/newsgroups/previewFrame.asp?ICP=cddgall&amp;sLCID=US&amp;sgroupURL=microsoft.public.speech_tech&amp;sMessageID=%253C3e504db8%25241@news.microsoft.com%253E</a> , retrieval 03-02-18
[Onl02]	Online GmbH: Was sind MVPs? <a href="http://www.ms-mvp.de/">http://www.ms-mvp.de/</a> , 2002, retrieval 03-03-08
[Pe03]	E-Mail from Lee Peedin. 03-01-21. Content: Agent and the procedure CkStatus
[Ro01]	Rothous, Doug: ADO.NET for the ADO Programmer <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/adonetprogmsdn.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/adonetprogmsdn.asp</a> , 2001, retrieval 02-12-19
[Sp03]	Conversation with Tobi Specht. 2003-03-05. Content: Windows XP Pro and network.
[To99]	Tower, Tandy: Uncork the Power of Microsoft Agent 2.0 <a href="http://www.microsoft.com/mind/0499/agent/agent.asp">http://www.microsoft.com/mind/0499/agent/agent.asp</a> , 1999, retrieval 03-01-23
[Tu02]	Turowsky, Klaus; Krammer, Andreas: Beiblattsammlung zur Vorlesung Web Engineering, 2002
[WS03]	W3Schools: The SOAP Header Element <a href="http://www.w3schools.com/soap/soap_header.asp">http://www.w3schools.com/soap/soap_header.asp</a> , 2003, retrieval 03-03-15
[WSS02]	Windows Scripting Solutions: Rem: Getting the WshController Object to Work <a href="http://www.winscriptingsolutions.com/Articles/Index.cfm?ArticleID=23607">http://www.winscriptingsolutions.com/Articles/Index.cfm?ArticleID=23607</a> , 2002, retrieval 03-02-23

[SGX]

### System Administration Scripting Guide Sources

The System Administration Scripting Guide Version 1.1, August 2002 can be downloaded from the Microsoft Homepage:

<http://www.microsoft.com/downloads/release.asp?ReleaseID=38942>



**Figure 81: System Administration Scripting Guide**

To get a text referring to the short reference get the keyword from the reference description and insert it into the Search line.

The short reference is inserted in brackets. The first two letters are SG for Scripting Guide. After that there follow one small letter for the further distinction.

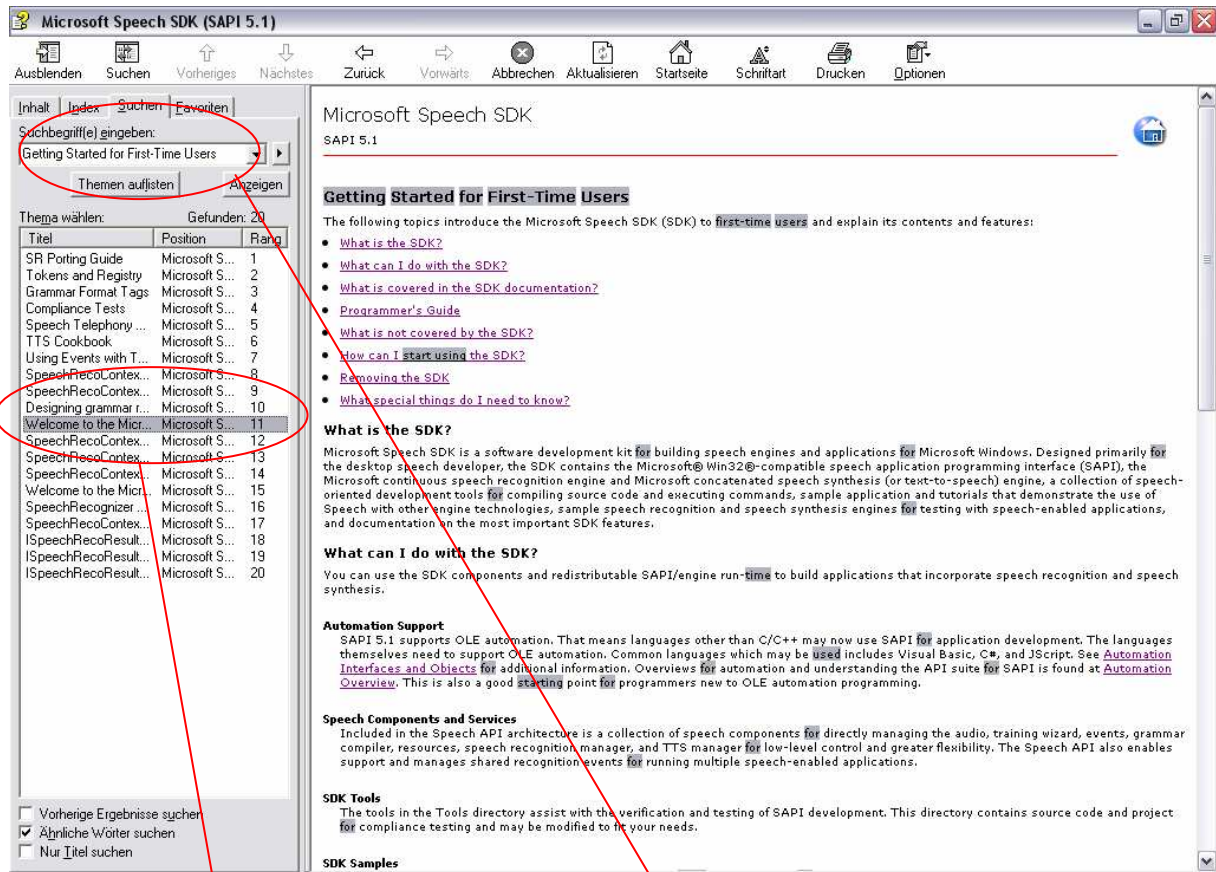
The description of this short reference contains the headline of the referenced text and the keyword, which is inserted into the Search field. If there are several results the resultnumber describes which result has to be selected.

[SGa]	Shut Down the Local Computer. Keyword: Win32Shutdown. Resultnumber: 1
-------	---

[SPXX]

### Microsoft Speech SDK 5.1 Documentation Sources

The Microsoft Speech SDK 5.1 Documentation is part of the MS Speech SDK and is located in the menu "Start->Programs->Microsoft Speech SDK 5.1. It can be also single downloaded from <http://www.microsoft.com/speech/download/sdk51/>.



**Figure 82: Microsoft Speech SDK 5.1 Documentation**

To get a text referring to the short reference get the keyword from the reference description and insert it into the Search line. The short reference is inserted in brackets. The first two letters are SP for Speech. After that there follow one or two small letters for the further distinction. The description of this short reference contains the headline of the referenced text and the keyword/sentence, which is inserted in the Search field. If there are several results the resultnumber describes which result has to be selected.

[SPa]	Getting Started for First-Time Users. Keyword: Getting Started for First-Time Users. Resultnumber: 11
[SPb]	Object Tokens and Registry Settings. Keyword: Object Tokens and Registry Settings. Resultnumber: 1
[SPc]	SpVoice. Keyword: SpVoice. Resultnumber: 7
[SPd]	GetVoices Method. Keyword: GetDescription. Resultnumber: 6
[SPE]	What is a Recognition Context?: Keyword: SpSharedRecoContext. Resultnumber: 3
[SPf]	CreateGrammar Method. Keyword: CreateGrammar. Resultnumber: 1



[SPg]	DictationSetState Method. Keyword: DictationSetState. Resultnumber: 1
[SPh]	Recognition Event. Keyword: Recognition Event. Resultnumber: 12
[SPi]	FalseRecognition Event. Keyword: FalseRecognition Event. Resultnumber: 3
[SPj]	StartStream Event. Keyword: StartStream Event. Resultnumber: 7
[SPk]	VB Application Sample: Dictation Recognition (Shared). Keyword: VB Application Sample: Dictation Recognition (Shared). Resultnumber: 2
[SPl]	CmdLoadFromFile Method. Keyword: CmdLoadFromFile. Resultnumber: 1
[SPm]	SpeechLoadOption Enum. Keyword: SpeechLoadOption Enum. Resultnumber: 1
[SPn]	CmdSetRuleIdState Method. Keyword: CmdSetRuleIdState. Resultnumber: 1
[SPo]	VB Application Sample: Command and Control Recognition. Keyword: VB Application Sample: Command and Control Recognition. Resultnumber: 3
[SPp]	CreateRecoContext Method. Keyword: CreateRecoContext. Resultnumber: 1
[SPq]	SpeechRuleState Enum. Keyword: SpeechRuleState. Resultnumber: 1
[SPr]	Add Method. Keyword: Add Method. Resultnumber: 17
[SPs]	SpeechRuleAttributes Enum. Keyword: SpeechRuleAttributes Enum. Resultnumber: 1
[SPt]	Clear Method. Keyword: Clear Method. Resultnumber: 8
[SPu]	AddWordTransition Method. Keyword: AddWordTransition Method. Resultnumber: 3
[SPv]	AddState Method. Keyword: AddState Method. Resultnumber: 1
[SPw]	Commit Method. Keyword: Commit Method. Resultnumber: 10
[SPx]	CmdSetRuleState Method. Keyword: CmdSetRuleState Method. Resultnumber: 1
[Spy]	SpeechRuleState Enum. Keyword: SpeechRuleState Enum. Resultnumber: 1