“OBJECT REXX AND WINDOWS AUTOMATION INTERFACES”

Rony G. Flatscher

University of Augsburg, Germany

„The 2004 International REXX Symposium“, Böblingen, Germany,

May 3rd - May 6th, 2004.

ABSTRACT

In the Windows world the scripting respectively remote controlling of Windows and Windows applications is usually realized via the COM-based OLE/ActiveX-Automation Interfaces. Unfortunately, many times the documentation of the available interfaces is either not available or in a form which does not serve very well as reference for Object REXX programmers.

Usually, the available OLE/ActiveX-interfaces can be interrogated and therefore it becomes possible to analyze the interfaces (functions/methods with their signatures and return values as well as interceptable events) at runtime. This article describes the architecture of a set of Object REXX programs for interrogating and rendering the documented interfaces into HTML employing CSS and embedding Object REXX code. For non-Object REXX programmers a Windows Script Component (WSC) is devised, which allows e.g. C++, Visual Basic (Script) programmers to employ this particular functionality, which is implemented in Object REXX.

Exploiting the scriptability of the Internet Explorer it is applied as a GUI for some of the presented interrogation applications.

Keywords: Object REXX, Windows automation, OLE, ActiveX, WSC, WSH, HTML, DHTML, DOM

1 INTRODUCTION

IBM's Object REXX programming language version for the Windows platform got enhanced with specific Windows supporting modules. One Windows specific enhancement is the Object REXX class named „OLEOBJECT", which makes the important Windows infrastructure „Object Linking and Embedding" (OLE) available to Object REXX.

This article first introduces conceptually the building stones for OLE and ActiveX and explains the Object REXX supporting class OLEOBJECT by discussing the purpose of all of its methods. It then draws the attention to the IBM supplied OLEOBJECT examples which allow to remote control („script") Windows components by Object REXX and explains what they do. These examples are installed automatically on the Windows platform when Object REXX gets installed, but can be easily overseen by programmers.

In order to take full advantage of Object REXX on the Windows platform by remote controlling Windows OLE/ActiveX components, it is mandatory, that one learns about the published interfaces of such Windows components. For that purpose this paper introduces a set of Object REXX utilities which allow exploring Windows OLE/ActiveX components for their published interfaces, i.e. their methods, attributes, events and constants. The functionality of these programs is made available to any Windows programming language capable of instantiating and interacting with COM objects („Component Object Model", a building stone for OLE).

2 OBJECT LINKING AND EMBEDDING (OLE)

This section introduces the reader to the technological infrastructure (on a conceptual level) for scripting Windows applications. First the fundamental Windows component technology (COM) is presented on which the „Object Linking and Embedding“ (OLE) technology is built, which the Windows scripting technology uses („OLE automation“).

2.1 „Component Object Model“ (COM)

At the end of the 80’ies Microsoft introduced a remote procedure call (RPC) convention for Windows which it named „Component Object Model“ (COM). COM defined the protocol and a minimal set of functions („interfaces“) which needed to be implemented by all programmers adhering to the COM specification. Doing so allowed the creation of self contained and executable program packages („components“) which could be interacted with by the standardized COM interfaces. One important COM function, named „IUnknown“, allows the inspection of such components at runtime, e.g. enabling the detection of published[1)] functions (“methods“) and their signatures (arguments and argument types, if any, return value and return value’s type, if any).

Windows COM components can be registered with the Windows operating system by entering the appropriate information into the Windows registry, which Windows services then can use to locate, load, initialize („instantiate“), and run them. Each such registered COM component must possess a system wide unique value, such that two different COM components can be distinguished at all times. For that reason algorithms to create GUID[2)] („Globally Unique Identifier“) values are used, that are theoretically able to create unique values not duplicable on any other computer system in the world. An example for a GUID/UUID used for uniqely identifying COM components in Windows is: „{d51ba994-e701-4f1a-a7b4-4a18a2ef22b9}“.

---

1) There may be many more functions („methods“) implemented in a Windows COM component than the author of it publishes via IUnknown.

2) A synonym for GUID is UUID („Universally Unique Identifier“).

As COM components are called COM *classes*, the unique GUID to identify it is called „class identification“ (CLSID). Analyzing the Windows registry hive „HKEY_CLASSES_ROOT“ will yield many entries possessing a key named „CLSID“ carrying a unique UUID/GUID value. If Windows programs need to use a COM class, then it is necessary to identify the desired component by supplying its CLSID value, which allows Windows to look up its registry to find out which COM component it should load. Clearly, it is very cumbersome for humans to memorize such GUID/UUID values, therefore alternative naming schemes are available, allowing the addressing of COM components in a more programmer friendly way:

- Program identification strings (PROGID): these are human readable and easy to memorize strings like „InternetExplorer.Application“. Such PROGIDs must be contained in the Windows registry as well and point to the COM class by pointing to the respective CLSID.

- Moniker strings: these are human readable and easy to memorize or to build strings like „c:\MyExcelFile.xls“. The monikers allow Windows to find out what COM classes are able to work on them. In our example it is clear that an Excel spreadsheet is denoted and Windows will be able to infer the respective Excel COM class, because of the file type „.xls“ and the entries in the registry, determining the COM class for files of type „.xls“.

Clearly, the Windows registry plays an extremely important role addressing and identifying them in a standard way. However, it must be noted that COM classes need not be registered with the Windows registry. For that reason it is possible, that some COM methods return instances of (other) COM classes that cannot be inferred from the Windows registry.

The Microsoft Office (MSO) applications are created as a set of COM classes that together realize the end user applications. Therefore it is possible to programmatically instantiate MSO classes (components) and interact with them.

2.2 OLE

„Object Linking and Embedding“ (OLE) is a set of specifications building on COM, which unifies the interaction between different applications either by means of linking to external storage or by means of embedding external application data in one owns storage. The functionality is defined with a set of interfaces which need to be implemented using the COM specifications. Every Windows program adhering to the OLE specifications can interact with any other Windows application that adheres to OLE as well. So OLE defines a common set of interfaces to communicate for the following purposes:

- Exchange of information irrespectible of the storage: applications can link to each other and exchange data whenever data changes in the „source“ application. This communication can be kicked off by the „target“ application only, in which case a „cold link“ is realized.

  An alternative way of communicating the change of data is establishing a „warm link“ or a „hot link“. The „target“ application informs the „source“ application not only about the data it should make available to the target, but agrees with the source to get informed whenever the source data changes („warm link“). If the source application should automatically send the changed data without informing the target beforehand, then this link is called „hot“.

- Retrieving data from a source application which will be stored in the target application storage (e.g. a file). The target application is able through the OLE communication means to start the source application for editing or printing purposes. If editing mode is desried, then the source application takes over the target Window and allows editing „in place“. Hence, source applications are also named „editors“.

One particular interesting OLE enhancement is the definition of a standard interface for scripting OLE applications. If there is an implementation of the OLE automation available to a programming language, then it becomes possible to script (remote control) OLE programs from that programming language. As Object REXX on Windows is capable of interfacing with OLE one can use Object REXX to script (remote control) OLE applications.

2.3 ActiveX Automation

At the end of the 90'ies Microsoft augmented the OLE automation functionality by introducing „ActiveX Automation", a set of COM interfaces which allow the creation of so called „Windows Script Hosts" (WSH).[3] WSH applications are Windows COM applications that are able to register scriptable COM objects with ActiveX scripting engines upfront, i.e. before any script gets invoked. By doing so a WSH is able to setup a scriptable environment alleviating a script programmer of incorporating those COM objects which he wishes to interact with.

Important WSH applications are MS Internet Explorer, MS Internet Information Server (IIS) and „Shell" (the Windows shell, a scriptable interface to Windows)[4].

The WSH infrastructure was introduced into the Windows operating systems with „Windows Millenium Edition" (WME, 16-Bit Windows) and „Windows 2000" (W2K, 32-Bit Windows). It also was made available via download for „Windows 95" (W95) and „Windows NT 4.0" (WNT) users.

2.4 The Object REXX „OLEObject" Class

The Windows version of Object REXX comes with an Object REXX class named „OLEObject". As the name implies, this class serves as a „proxy" class to allow Object REXX to interact with Windows COM objects as if they were Object REXX objects.

Instances of the Object REXX „OLEObject" class are Object REXX objects, which serve as „proxies" for instances of Windows COM classes. This means among other things, that sending Object REXX messages to OLEObject instances will cause the invocation of those methods on the Windows COM objects they represent. If Windows COM objects return other COM objects, the Object REXX OLEObject class

---

[3] Cf. [Flat02a], [Flat02b].

[4] The documentation to WSH 5.6 explicitly lists the name „Shell", yet, later Microsoft documentation has started to name the „Shell" Windows Script Host application „WSH"! This has caused the creation of a homonym and has a result introduced a lot of confusion.

will turn them into proxy objects, allowing Object REXX programmers to send them Object REXX messages as well.

It is interesting to explore how this mechanism is implemented in the OLEObject class, in order to understand why it is possible to interact with Windows COM objects and to learn about possible pitfalls and how to come by them.

Studying the (astonishingly few) methods OLEObject possesses one particular method plays an extremely important role: „UNKNOWN“.

2.4.1 The „UNKNOWN“ Method

The „UNKNOWN“ method in the Object REXX language has a particular meaning to the Object REXX runtime system: whenever a message is not understood by an object, the runtime system invokes that object’s UNKNOWN method supplying the name of the unknown message and the arguments sent with it. If there is no UNKNOWN method defined in the class or one of the superclasses of that object, then the Object REXX runtime system will abort the execution of the entire program with the error message „object does not understand message“.

Because of this behaviour the UNKNOWN method can serve as a particular runtime error handler, in that it gets activated whenever unknown messages are sent to an object. In the case of OLEObject proxy objects it is the case for most messages meant for the Windows COM objects they represent, which usually means that no methods by the name of such messages exist on the Object REXX side. Therefore such proxy objects are not understanding the message sent to them and the OLEObject UNKNOWN method gets invoked by the runtime system.

The OLEObject UNKNOWN method learns about the name of the unknown method and the arguments sent with the message (if any at all) in form of the runtime supplied arguments and will now interrogate the Windows COM object using the COM interfaces for learning about the available functions (methods) and their arguments (if any at all). With this knowledge and with the ability to convert the (Object) REXX arguments to the published Windows datatypes it becomes possible to set up a Windows method invocation call and have it carried out on the Windows side. The return value (if any at all) of the invoked COM method will get converted to

an appropriate Object REXX value (a string or an OLEObject proxy object, if a COM object was returned).

This forwarding mechanism using the Object REXX UNKNOWN mechanism works fine for most messages meant for the Windows COM objects. However, sometimes Windows COM objects need to get messages sent to which get consumed on the Object REXX side. This may happen, if the Windows COM objects possess methods with the same name as in the Object REXX root class „Object“ or the „OLEObject“ class itself.

Examples are the „CLASS“ or the „COPY“ methods, which are available in the Object REXX root class „Object“. Therefore e.g. a „COPY“ message sent to the Object REXX proxy will get resolved on the Object REXX side (the Object REXX root class „Object“ COPY method will get carried out), therefore the UNKNOWN runtime mechanism does not get triggered and the Windows COM object would never get a „COPY“ message sent to it (e.g. for copying a selection to the Windows clipboard). In such a case it becomes necessary to directly invoke the OLEObject UNKNOWN method, supplying the name of the message as its first argument, and either „.nil“ (= no arguments) or an Object REXX array object containing all arguments to be forwarded to the Windows side, e.g.

```
myExcelProxyObject~unknown("COPY", .nil) -- send the COPY message to Windows
```

2.4.2 The „INIT“ Method

Whenever an instance of an Object REXX class is created by sending the class object the „NEW“ message, that newly created object gets the „INIT“ message sent to it accompanied by the arguments which the programmer supplied to the „NEW“ message in the same order. Therefore the „INIT“ method of the OLEObject class documents the initialisation arguments one is able to send to the newly created object.

In the case of the OLEObject class one must supply a PROGID or a CLSID string as the first argument, identifying the Windows COM class which should get instantiated and for which the newly created Object REXX object serves as a proxy. The second argument is optional and determines whether Windows COM events should get forwarded to the Object REXX side. If the forwarding of Windows COM events is

desired, then the Object REXX programmer needs to create a subclass of „OLEObject“ and define methods by the name of the Windows COM events which should get intercepted on the Object REXX side. In such a case one needs to create an instance of the created subclass, supplying the second argument with a value of „WITHEVENTS“.

If using the „OLEObject“ class for creating Windows COM proxy objects, then by default the value „NOEVENTS“ is assumed for the second argument.

2.4.3 The „GetObject“ Class Method

There is an alternative way to create an Object REXX proxy object for a Windows COM object: the „GETOBJECT“ class method of the „OLEObject“ class. Sending this message to the „OLEObject“ class object allows defining an application dependent „moniker“ string allowing the Windows OLE support to figure out the OLE host application and querying it for a Windows COM object of it. Such monikers are specially formatted strings, as simple as a fully qualified file name possessing a „file type“ (e.g. a MS Excel or a MS Word file name) or as complicated as a WMI[5)] query.

If an instance of the appropriate (with the indicated moniker) Windows application is running already, the returned proxy object will refer to it. This way it becomes possible to acquire a proxy object to address running Windows programs.

This method allows the creation of proxy objects for already instantiated Windows COM classes which are already running, like a particular instance of MS Word or MS Excel processing a specific file. If there is a need to intercept events from such Windows COM objects then the „GETOBJECT“ class method allows for supplying an appropriate „OLEObject“ class as its second argument (a direct or indirect specialization of the „OLEObject“ class, containing methods by the same name as the Windows COM events which should activate them, supplying the arguments of the event, if any). If the second argument is omitted, then the „OLEObject“ class is

[5)] „WMI“ is the acronym for „Windows Management and Instrumentation“, a Microsoft implementation for managing information technology resources like computers, printers etc. via OLE. It is part of the MS Win2K or WinXP 32-Bit operating systems and can be downlaoded and installed to earlier versions of Windows. There are small („nutshell“) Object REXX examples distributed with Object REXX itself taking advantage of the WMI interfaces.

used to create the Object REXX proxy objets and no Windows COM events are intercepted.

2.4.4 The „GetConstant“ Method

The OLE specification allows for defining constants, which are symbolic names which serve as a synonym to some predefined value. The constant’s name carries the semantics for the human programmer, whereas the represented value itself is usually some binary value. An example for such a constant is „OLECmdID_undo“ which gets defined in th MS Internet Explorer application and which stands for the value „15“ which must be supplied to the message instead.

Many OLE applications publish the constants which are defined for them and the online help of such applications usually uses the symbolic names instead of the values they represent. For that reason it is important to be able to query the value of a constant of an OLE application (at runtime) to be used as the argument for the message directed at the Windows COM object. The „GetConstant“ method accepts the symbolic name as its sole argument and returns the value that constant represents, or .nil, if the constant is not published.

It is also possible to use this method to retrieve a list of all published constants of an OLE application by omitting the argument altogether in which case a stem array of all constants and the values they represent is returned.

2.4.5 The "GetKnownEvents" and "GetKnownMethods" Methods

Generally, the COM (and OLE) specifications define an interface which allows programmers to retrieve a list of all published methods (and events) in a standardized way. These two methods allow Object REXX programmers to retrieve stem arrays containing the exact definitions of the published methods and events made available by the Windows COM (OLE) class.

2.4.6 Type Conversion and Argument Considerations

Object REXX is a programming language which is based on REXX and as such has as its "basic type" a string value. Object REXX adds to this the Object REXX classes which serve as a additional types. None of the Object REXX types does represent or

Variant's Name	Comment	Object REXX Value
VT_EMPTY, VT_NULL, VT_VOID	Indicate no value.	.nil
VT_I1, VT_i2, VT_I4, VT_I8	Signed integer of size 1, 2, 4 or 8 bytes.	Respective signed whole number (a string)
VT_UI1, VT_UI2, VT_UI4, VT_UI8	Unsigned integer of size 1, 2, 4 or 8 bytes.	Respective unsigned (positive) whole number (a string).
VT_R4, VT_R8	Real number of size 4 or 8 bytes.	Respective real number (a string).
VT_CY	Fixed point decimal number to represent currency values (15 digits before the decimal point, 4 digits after the decimal point.	Respective decimal number (a string).
VT_DATE	Represents a date.	A string.
VT_BSTR	A string of bytes.	A string.
VT_DISPATCH	A COM (OLE) object.	An instance of "OLEObject".
VT_BOOL	A truth value. False is represented with the numeric value 0, true with -1 (sometimes 1 and sometimes any value but 0).	.true (string '1') or .false (string '0').
VT_SAFEARRAY	An array with fixed lower and upper bounds.	An array.
VT_VARIANT, VT_PTR	Wraps one of the above types.	Respective representation.

Table 11 Windows Datatypes Used in the Context of COM (OLE).

directly relate to the Windows data types used in COM (OLE) as depicted in table 1. For that reason the "OLEObject" class must convert Object REXX values to the appropriate Windows data types and vice versa. Because it is possible to reflect the signatures (name, return type and argument types) of Windows functions/methods at runtime, conversion from/to Windows data types can be carried out at runtime without the need for Object REXX programmers to intervene. From the Windows data types in table 1 it may be noteworthy, that the"VT_DISPATCH" type represents a Windows COM object for which an Object REXX proxy object will be created.

The COM specifications were created by Microsoft to define a remote procedure call (RPC) standard for Windows. Such calls allowed for various (mostly performance) reasons, that arguments passed in a RPC call could be changed by the receiving end and hence may serve as a means of communication back to the calling end. Arguments which can be changed by the receiving side are dubbed "OUT" (only meant for returning a value) or "INOUT" (meant for submitting and returning a value) arguments. Object REXX does not allow arguments to be changed in place, so

therefore a specific mechanism is supplied by the "OLEObject" class in the form of the method "GetOutParameters". This method returns an Object REXX array containing the values of any "INOUT" and "OUT" arguments in ascending order ("IN"-only arguments are omitted from this array).[6)]

2.5 Further Resources Coming with Object REXX

The Object REXX for Windows installation comes with a very rich set of very small ("nuthshell") examples, demonstrating the usage of the "OLEObject" class, i.e. the automation of Windows OLE applications. Because of the excellent OLE examples the reader's attentions is therefore pointed at them, as tehy demonstrate how Object REXX is able to unleash the power of OLE/ActiveX automation:[7)] e.g. remote controlling MS Internet Explorer, MS Excel, ADS ("advanced directory services"), WMI ("Windows management and instrumentation") and the like.

[6)] Clearly, allowing the OUT and INOUT semantics in Object REXX for Windows would be desirable, because it would ease coding and transcribing of e.g. Visual BASIC programs to Object REXX straight forwardly.

[7)] The "SAMPLES" subdirectory contains subdirectories which also demonstrate the "native" Object REXX low level interface to Windows GUI components. In addition there are interesting examples demonstrating how one can use Object REXX to implement COM classes that could be instantiated by any Windows programming language able to interact with COM like C++, Visual BASIC and the like ("WSC", "Windows Script COM"ponents).

3 THE "RGF_OLEINFO" PACKAGE

Object REXX for Windows adds specific support for the Windows platform, which allows Object REXX to be used as a fully fledged programming (and scripting) language for that platform. Especially the support for COM (OLE) allows for unleashing the power of Object REXX.

However, analyzing the Windows applications that can be remote controlled ("automated") via OLE (ActiveX) it becomes clear, that many such applications lack of a stringent, easy to understand documentation. Rather, documentation may be scattered around many documents and help files such, that one is hardly able to gain an overview and get at a compact listing of available interfaces. An example may be the MS Office suite of applications.

Sometimes the documentation on how an application can be driven via OLE is missing altogether, because it does not get installed with the application itself by default (like the Lotus set of applications). At other times OLE/ActiveX objects may be returned by driving documented OLE/ActiveX interfaces, which themselves are not documented online at all.

Analyzing a modern installation of Windows like XP it is interesting to learn that there are well over 2,000 (!) COM classes that are installed and that get recorded in the Windows registry, many of which can be driven via OLE/ActiveX automation. Most users of Windows XP systems are not even aware of the wealth of functionality being presented in an automatable form!

Yet another interesting capability to note when comparing Object REXX with MS Visual BASIC in the context of OLE/ActiveX scripting is the ability of the Visual BASIC programming environment to allow using the symbolic names of constants instead of the values they represent, making coding rather easy in this respect.[8)]

[8)] The Visual BASIC development environment allows identifying those OLE/ActiveX libraries from which one wishes to use the name of the constants, by choosing the appropriate entries in the "Tools" main menu. In addition the signatures of methods and events are made available in a context sensitive manner, which provides some nice help while coding.

This section therefore introduces programs and utlities written entirely in Object REXX which attempt to forgo all of these shortcomings, using the Windows specific support supplied with IBM's Object REXX for the Windows platform. There will be four utilities introduced and discussed:

1) "rgf_oleconstants.rex", a tool for making it easy to use published OLE constants in Object REXX,

2) "rgf_oleinfo.rex", an Object REXX program creating a HTML rendering containing the published interfaces to a OLE application,

3) "rgf_oleinfo.wsc", a "Windows Script COMponent" allowing any programming language to take advantage of the anaylzing and documentation capabilities of "rgf_oleinfo.hta",

4) "rgf_oleinfo.hta", a "HTML application" making it easy to analyze the COM classes available on a Windows system.

3.1 "rgf_oleconstants.rex"

OLE/ActiveX applications usually define constants which allow human programmers to easily memorize and understand the purpose of the value a constant represents. Object REXX programmers can query the value of any published OLE constant using the "OLEObject" method "getConstant". Sometimes, especially if driving the same OLE applications repeatedly for different application needs, it would be helpful to allow for an easier way at getting at the values the constants represent.

The Object REXX application "rgf_oleconstants.rex" is a little utility program, which interrogates all of the published constants of a given OLE application, sorts them by constant name and saves them one by one in an Object REXX directory object. This is carried out by creating actually an Object REXX program in which the directory object is set with all the entries mapping the name of the constant to its value. The "constant" directory object is stored in the local environment under the name "ole.const" and created, if it does not exist yet.

```
/* [rgf_oleconstants.rex] run on: [20040430] [21:55:16] */

-- OLE/ActiveX-application/CLSID: [InternetExplorer.Application]
-- there is/are [84] constants

-- create directory 'ole.const', if necessary; maybe shared with
-- OLE constant definitions of other programs

   -- create directory 'ole.const' in .local
if .local~hasentry('ole.const')=.false then .local~ole.const=.directory~new

.ole.const~CSC_NAVIGATEBACK                = 2
.ole.const~CSC_NAVIGATEFORWARD             = 1
.ole.const~CSC_UPDATECOMMANDS              = -1
.ole.const~OLECMDEXECOPT_DODEFAULT         = 0
.ole.const~OLECMDEXECOPT_DONTPROMPTUSER    = 2
.ole.const~OLECMDEXECOPT_PROMPTUSER        = 1
.ole.const~OLECMDEXECOPT_SHOWHELP          = 3
.ole.const~OLECMDF_DEFHIDEONCTXTMENU       = 32
.ole.const~OLECMDF_ENABLED                 = 2
.ole.const~OLECMDF_INVISIBLE               = 16
.ole.const~OLECMDF_LATCHED                 = 4
.ole.const~OLECMDF_NINCHED                 = 8
.ole.const~OLECMDF_SUPPORTED               = 1
.ole.const~OLECMDID_ALLOWUILESSSAVEAS      = 46
.ole.const~OLECMDID_CLEARSELECTION         = 18
.ole.const~OLECMDID_CLOSE                  = 45
.ole.const~OLECMDID_COPY                   = 12
... cut ...
```

Figure 1: Content of "iex.rex" (an excerpt).

The name of this utility is "rgf_oleconstants.rex". It nees to be executed from a command line window and expects two blank delimited arguments: the PROGID or CLSID of the COM/OLE class to be queried and optionally the name of an output file, which will contain an Object REXX program setting the constant directory to the respective values. Example creating all the constants published by MS Internet Explorer:

```
rgf_oleconstants InternetExplorer.Application iec.rex
```

This will create an Object REXX program named "iec.rex" containing all necessary statements to store all published MS Internet explorer constants in the Object REXX directory object named ".ole.const". An excerpt of the generated file can be seen in figure 1. Any Object REXX program needing access to the Internet Explorer constants then would either issue a "CALL iec.rex" or define the directive "::REQUIRES iec.rex" in its program. After that it is easy to retrieve the value of any constant by merely sending the constant's name as a message to that directory object (".ole.const"):

```
.ole.const~csc_navigateBack -- will return the value '2'
```

No matter how many different Object REXX programs containing the constant definitions of different Windows OLE classes one creates with the utility "rgf_oleconstants.rex", they will re-use the same directory object with the environment name ".ole.const". This utility therefore creates a unique source serving the values for the different OLE constants different OLE classes define.

3.2 "rgf_oleinfo.rex"

The set of "rgf_oleinfo.rex" utility programs hase been created to ease a Windows user in exploring and understanding the COM/OLE classes installed on his computer.

One utility ("rgf_olinfo.rex") analyzes a given OLE class with the help of all of its published interfaces and creates a HTML rendering of it. Another utility ("ole_info.wsc") makes the functionality of "ole_infor.rex" available to non-Object REXX programs like Visual Basic (Script), C++, etc. Lastly, a utility, that ("rgf_oleinfo.hta") uses the MS Internet Explorer to create a GUI frontend got created which allows analyzing the Windows registry for entries pointing to COM/OLE classes and making it easy for users to point to those OLE classes they wish to get their published interfaces rendered as HTML by merely clicking on them.

3.2.1 "rgf_oleinfo.cls"

The "rgf_oleinfo.cls" program defines the public class "rgf.oleinfo" which is used to query the published interfaces of OLE classes, stores the results and makes them readily available via its attributes in form of directory objects for later use by other Object REXX programs. In addition, the most important Windows registry entries for that OLE class are analyzed and maintained for later referral by client Object REXX programs.

3.2.2 "rgf_oleinfo2html.rex"

The "rgf_oleinfo2thml.rex" program uses "rgf_oleinfo.cls" to get at the published interfaces of a given OLE class and renders the results in HTML using CSS (Cascading Style Sheets) in two forms: a "normal" and a "compact" form.

"rgf_oleinfo2html.rex" inserts Object REXX code into the generated HTML taking advantage of Microsoft's DHTML ("dynamic HTML") to hide or show HTML sections. In addition it formats registry information pertaining to OLE classes which allow documenting which CLSID, PROGID and VersionIndependentPROGID (if any) an OLE class carries, as well as when it got registered in the Windows registry (if registered at all) and which EXE or DLL implements the functionality.

The HTML rendering employs CSS ("cascading style sheets"), which determine the physical formatting of the HTML embedded data. The CSS allows users to change the formatting to their particular needs, e.g. using larger fonts, different colors and the like. CSS allows a very detailed control over formatting and it is even possible to define distinct formatting rules for printing the HTML data to a printer.

There are two different modes of presenting the data with this utility:

- "Normal mode": in this mode separate sections are created for listing all published methods, read-only attributes ("properties"), write-only attributes ("properties"), read-/write attributes ("properties"), events and constants.
- "Compact mode": in this mode all information is given as compact as possible, e.g. all attribute sections are folded into one. No constant section will be created.
  This mode is meant for creating "reference card" like listings, which are formatted (optimized) for color printouts.

3.2.3 "rgf_oleinfo.rex"

The utility program "rgf_oleinfo.rex" serves as the frontend to "rgf_oleinfo2html.rex" and can be used as a command via the commandline or as a function from any REXX program. The following arguments can be supplied:

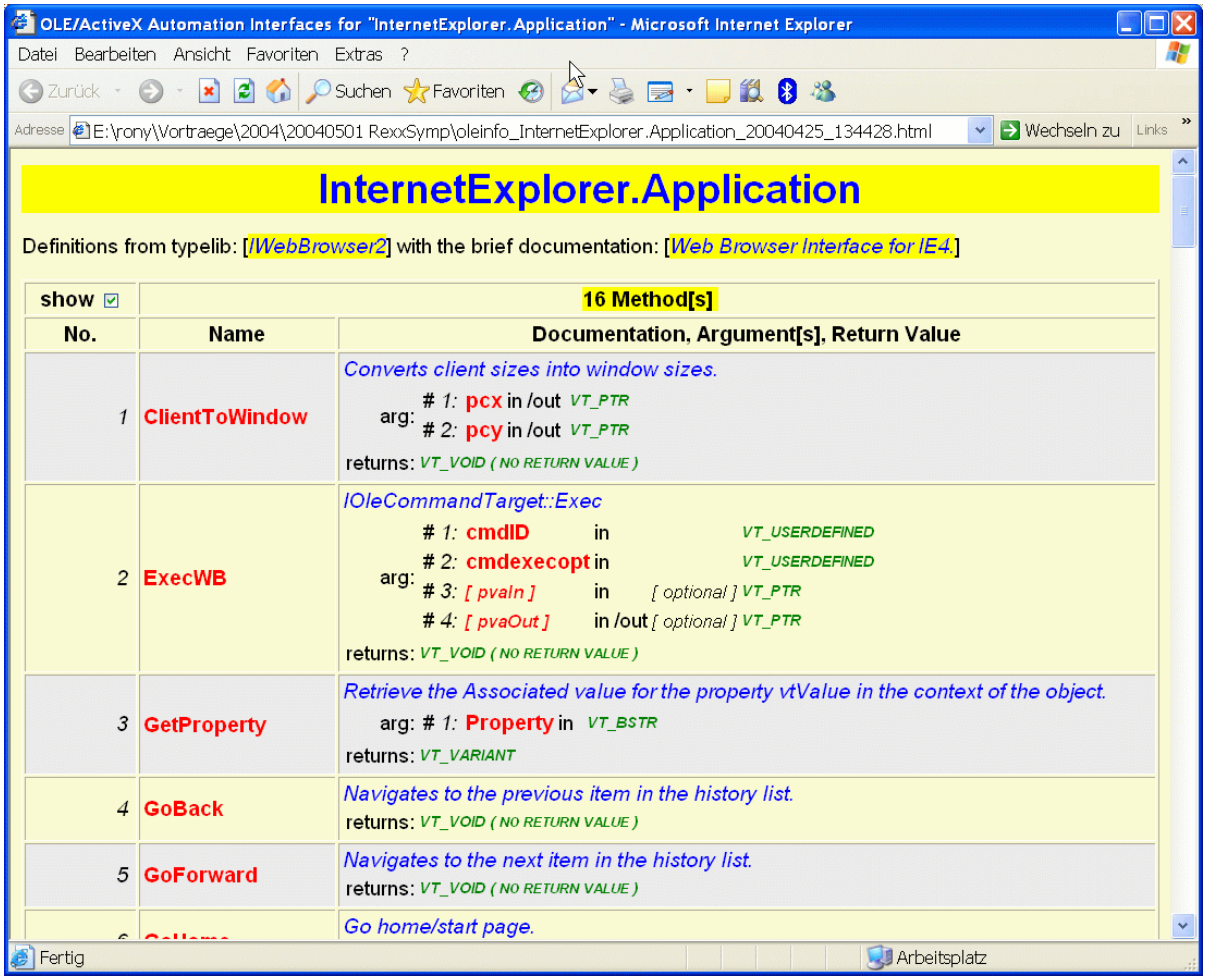

Figure 2: Result of running "`REXX rgf_oleinfo InternetExplorer.Application 0 1`".

- PROGID | CLSID: either string, identifying the OLE component to analyze OLEOBJ ... a proxy object, if invoking the program as a function,
- "header": the HTML header to be used in the HTML file,
- "mode": the mode (normal=0, compact=1), default: normal (command), compact (function),
- "display": should the resulting HTML file be displayed with Interent Explorer (no display=0, display=1), default: display.

The command line syntax uses blank delimited arguments and is defined as follows:[9)]

[9)] Optional arguments are enclosed in square brackets ([]), alternatives are delimited with a vertical

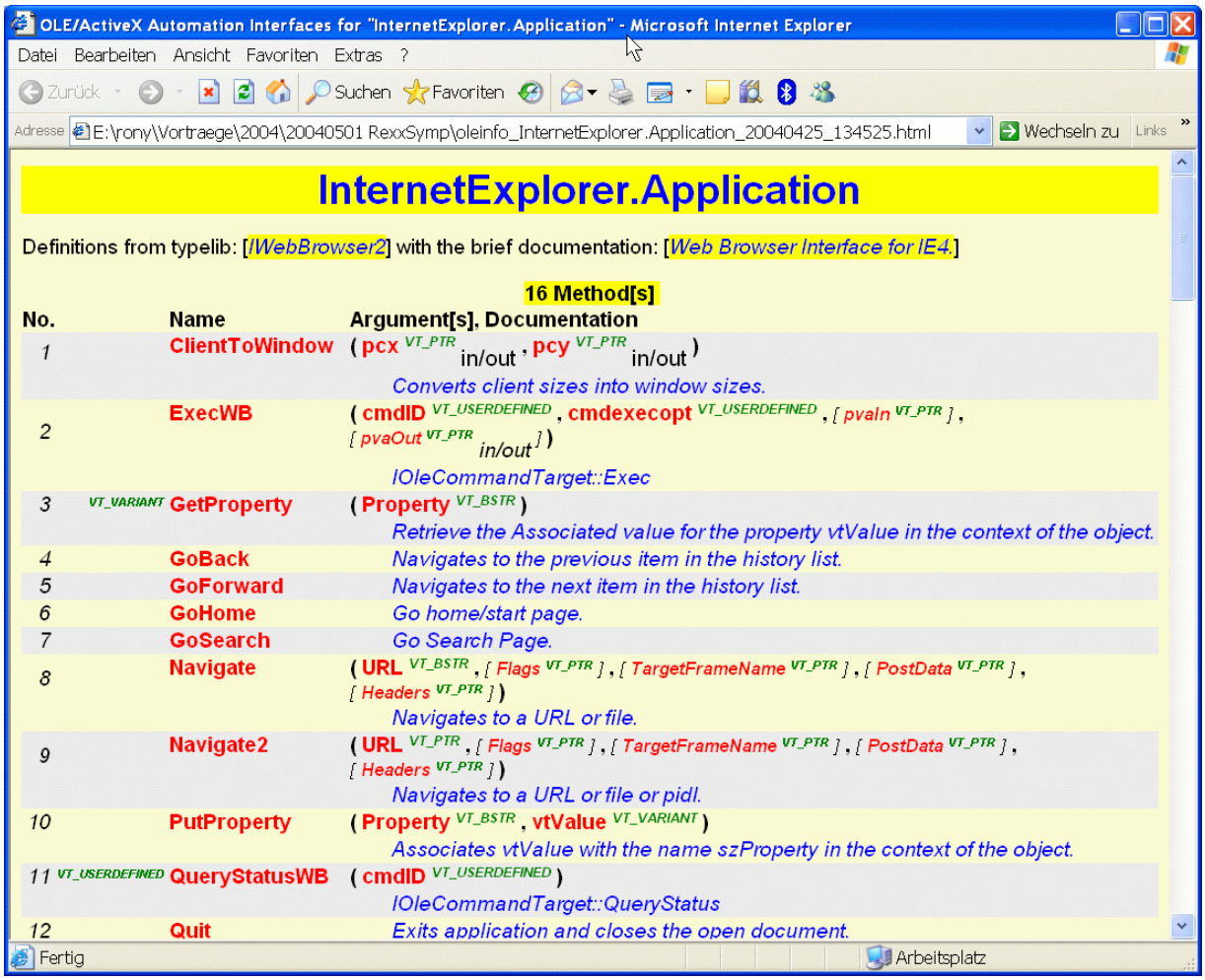

Figure 3: Result of: `res=rgf_oleinfo("InternetExplorer.Application",1,1).`

```
REXX rgf_oleinfo PROGID|CLSID [mode [display]]
```

The function syntax uses comma delimited arguments and is defined as follows:

```
res=rgf_oleinfo( PROGID|CLSID|{OLEOBJ [, header]} [, mode [, display]] )
```

The function will return .true if successful, .false if it was not possible to get an OLEObject proxy to analyze.

Figure 2 shows the result ("normal mode", displayed in Internet Explorer) of running the command:

```
REXX rgf_oleinfo InternetExplorer.Application 0 1
```

bar (|) and a mandatory argument is enclosed in curly brackets ({}) .

```
<?xml version="1.0"?>
<component>
<comment>
   author:  Rony G. Flatscher, WU Wien
   date:    2004-01-22
   purpose: wrapper for non-REXX programs wishing to analyze the documented interfaces
            of OLE/ActiveX objects

   hint:    if using this from Visual Basic (Script) you need to enclose the variable
            referring to the OLE/ActiveX object into parenthesis in order to pass the
            reference (and not the default string value of it)!

   needs:   the rgf_oleinfo-package for Object REXX
</comment>

<?component error="true" debug="true"?>

<registration
        description="Analyzes the OLE/ActiveX-Interface of the given PROGID or OLE/ActiveX object."
        PROGID="REXX.OLEinfo"
        version="1.00"
        classid="{d51ba994-e701-4f1a-a7b4-4a18a2ef22b9}"
>
</registration>

<public>
        <method name="analyze">
                <parameter name="PROGID_or_OLEobject"/>
                <parameter name="stringHTML_Header"/>
                <parameter name="boolCompactRendering"/>
                <parameter name="boolDisplayImmediatelyInMSIE"/>
        </method>
</public>

<script language="Object REXX">
<![CDATA[

       -- invoke program which does the gory work ...
    ::routine analyze public
      call rgf_oleinfo arg(1), choose(arg(2)="" | arg(2, "o"), "n/a" , arg(2)), arg(3), arg(4)

       -- if arg(1) yields .true return arg(2), else return arg(3)
   ::routine choose
     if arg(1)=.true then return arg(2)
                     else return arg(3)
]]>
</script>

</component>
```

Figure 4: "rgf_oleinfo.wsc".

Figure 3 shows the result ("compact mode", displayed in Internet Explorer) of running the function:

```
res=rgf_oleinfo("InternetExplorer.Application", 1, 1)
```

3.3 "rgf_oleinfo.wsc"

The utility program "rgf_oleinfo.wsc" is realized as a "Windows Script COM" program, implementing a [D]COM class in Object REXX and serves as the interface to any Windows program that is capable of using COM classes. This way it becomes possible, e.g. for Visual Basic or VBScript programmers to use the introduced Object REXX programs to analyze any COM object and render all the published interfaces

```
' VBScript
dim mxVar, myVar
Set mxVar = createObject("REXX.OLEinfo")
Set myVar = createObject("InternetExplorer.Application")

WScript.echo "about to use 'REXX.OLEinfo.analyze()...'"

' OLEobject *must* be enclosed in parenthesis, otherwise
' the default string value is retrieved!
mxVar.analyze ( myVar )

WScript.echo "done. (vbs)"
```

Figure 5: Using the Object REXX COM "REXX.OLEinfo" class from VBScript.

attractively in HTML[10]. Figure 4 depicts the code of "rgf_oleinfo.wsc".

The PROGID of the WSC class is "REXX.OLEinfo" and becomes available, once the WSC-file got registered[11] on Windows, the sole published method is named "analyze" and expects the following arguments:

- "PROGID_or_OLEobject": supply either the PROGID of the COM class to analyze (Object REXX needs to be able to successfully instantiate it), or supply an OLE object,
- "stringHTML_Header": supply the title for the HTML rendering of the published interfaces,

```
// JScript
var mxVar, myVar
mxVar = new ActiveXObject("REXX.OLEinfo")
myVar = new ActiveXObject("InternetExplorer.Application")

WScript.echo( "about to use 'REXX.OLEinfo.analyze()...'" )
mxVar.analyze(myVar, "invoked via JScript !")
WScript.echo( "done. (js)" )
```

Figure 6: Using the Object REXX COM "REXX.OLEinfo" class from JScript.

[10] In Visual Basic or VBScript it is necessary to supply the OLE object to be analyzed enclosed in parenthesis. Otherwise the string value of the OLE object gets supplied.

[11] Registering is easiest via the Windows Explorer by moving the mouse over the file "rgf_oleinfo.wsc", right-click on it and choose "Register".

- "boolCompactRendering": supply "1" (.true) to render in compact mode (cf. figure 3), or "0" (.false) to render in verbose mode (cf. figure 2),
- "boolDisplayImmediatelyInMSIE": if "1" (.true) the resulting HTML file is displayed with the MS Internet Explorer, else ("0"=.false) the HTML file gets created and stored in the file system.

Figure 5 shows how one can use the Object REXX COM class from VBScript and figure 6 displays the code to achieve the same from JScript.

3.4 "rgf_oleinfo.hta"

The utility program "rgf_oleinfo.hta" is a so-called "HTML application" (cf. [Flat02b]) for Windows and serves as a graphical front end for entering PROGIDs or CLSIDs in a user friendly way for analyzing and rendering the published interfaces in HTML.

HTA-files are regarded like any other executable under Windows and therefore execute under the credentials of the logged-on user who invokes it. As a result script code embedded in the HTML applications is not executing in the "sandbox" environment of the MS Internet Explorer and has unrestricted access to the system it runs on.[12)]

"rgf_oleinfo.hta" starts out with a list of pre-defined applications ("initial apps") and is depicted in figure 7:

- Checkbox "Generate compact listings": if checked, a compact listing of the published interfaces will be generated (cf. figure 3), otherwise a little bit more verbose and structured listing with the published constant values is created (cf. figure 2).
- Clicking on any link in the list of "initial applications": the appropriate COM class is instantiated and its published interfaces queried and rendered.

[12)] "Unrestricted" in this context means with the credentials of the user's session in which the HTA got invoked. If that user happens to be the "Administrator" or a member of the Windows "Administrator group", the HTML application indeed has practically unrestricted access to the Windows machine!

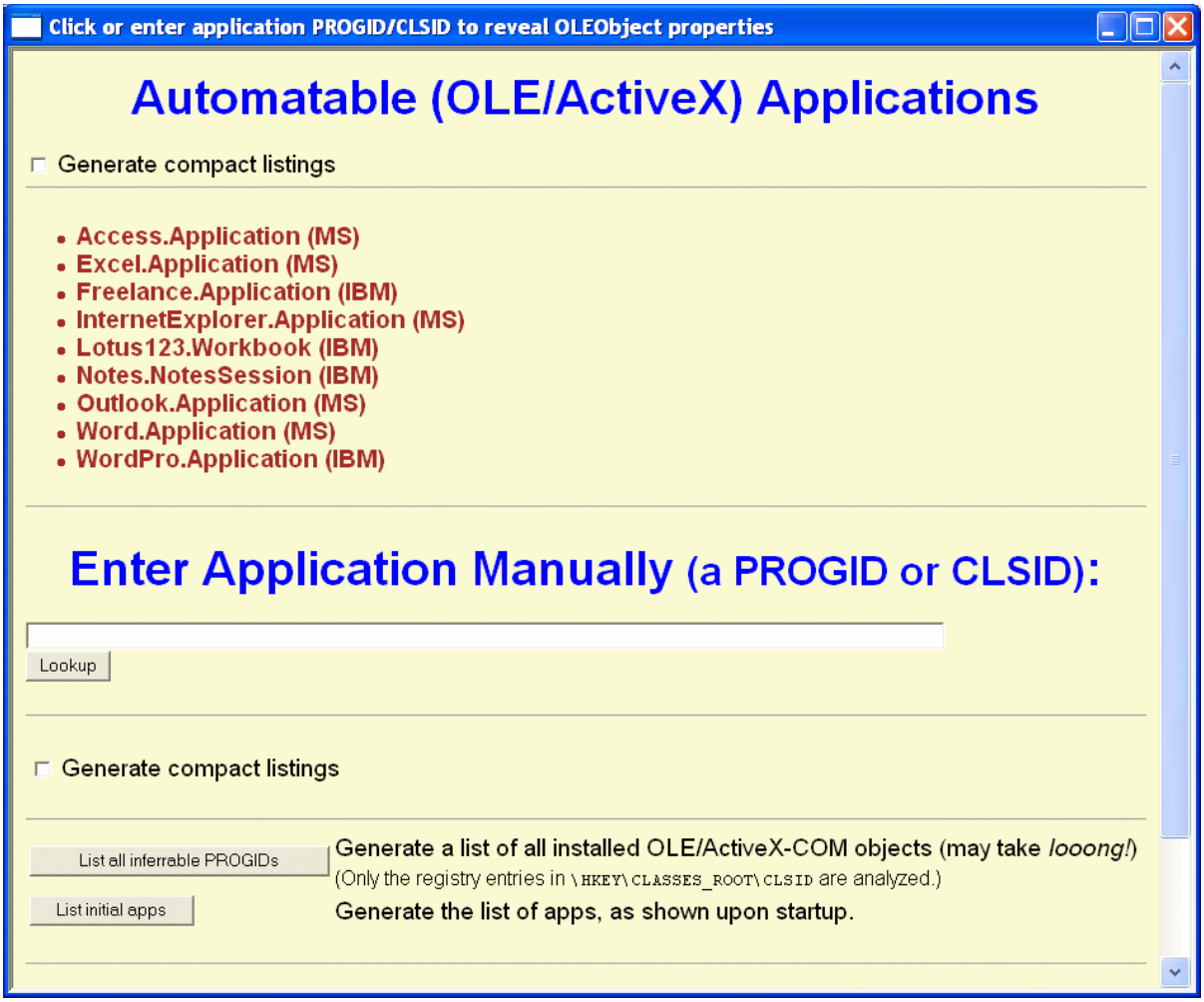

Figure 7: Using a HTA for a GUI Frontend, Initial Screen.

- Text-field underneath "Enter Application Manually (PROGID or CLSID)": allows entering a known PROGID or CLSID directly, pressing the push button "Lookup" will start the analyzing and rendering process.

- Push button "List all inferrable PROGIDs": this will cause the analysis of the Windows registry for PROGIDs in the "HKEY\CLASSES_ROOT\CLSID" branch. This process may take quite a while, depending on the power of the processor and on the amount of registry entries, which could be in the thousands. Figure 8 depicts such a run where the screen got scrolled to the very end, revealing that on that Windows machine there were 2,587 inferrable PROGIDs registered!

- Push button "List initial apps" will replace the list of links to COM classes with the one that is pre-defined and displayed at start-up.

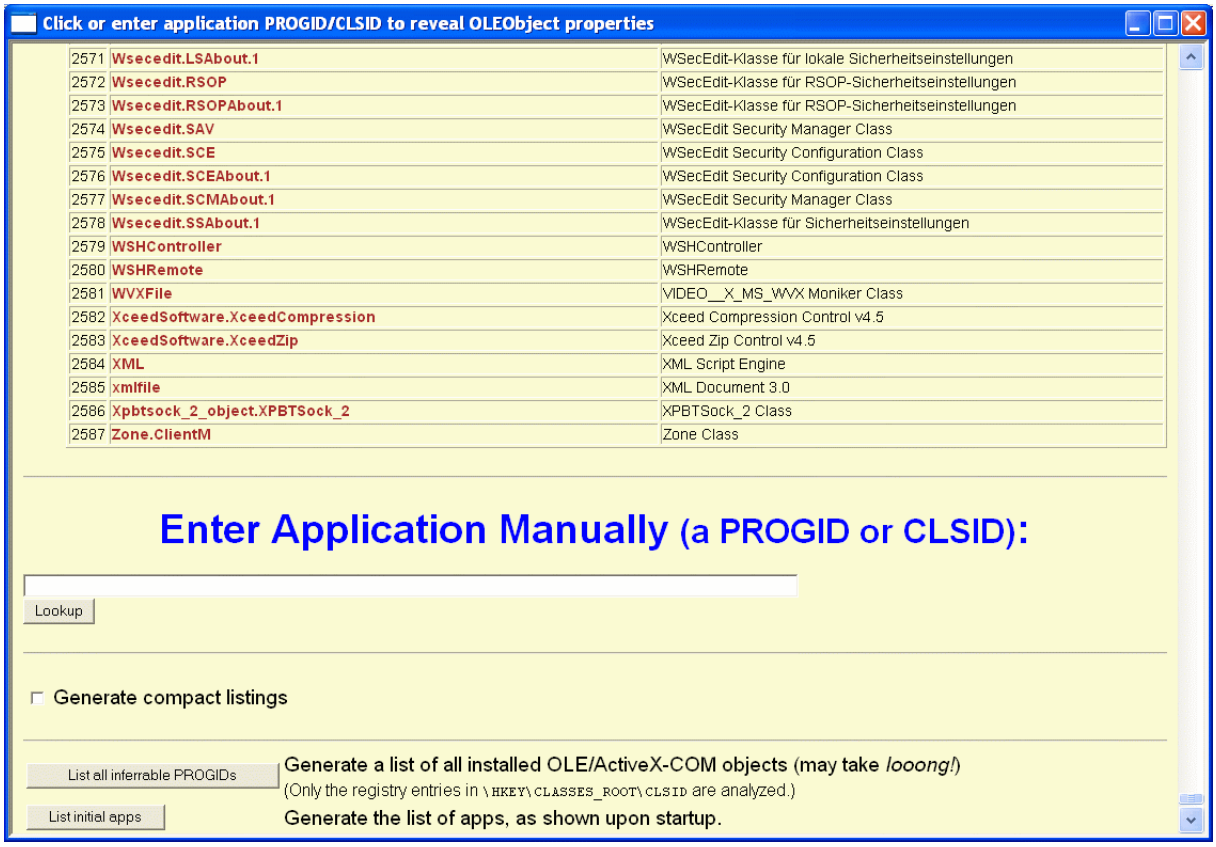

Figure 8: Using a HTA for a GUI Frontend, Full List of Inferrable PROGIDs.

4 SUMMARY AND OUTLOOK

This article introduced the reader to the Object REXX support for Object Linking and Embedding Automation (OLE automation) on the Windows platform, which is realized via the Object REXX class named "OLEObject". It discusses the methods of that class and explains the role of the Object REXX UNKNOWN mechanism, which gets exploited by "OLEObject".

The "rgf_oleinfo" suite of Object REXX programs allow for analyzing the published interfaces of Windows COM and DCOM classes via OLE respectively ActiveX, which is the latest incarnation of the Windows script host support. The article describes the purpose and the available interfaces of these programs and shows how non-Object REXX languages can take advantage of the analysis and rendering to HTML capablities by the means of a "Windows Script Component" (WSC) file.

Finally, a GUI interface realized via a "HTML application" (HTA) is introduced and discussed, which eases considerably the researching of the installed COM classes on any Windows computer, that can be driven via OLE/ActiveX automation.

The resulting HTML renderings allow for printing reference card like documentations of the published interfaces to COM classes, which otherwise are hardly available at all in such a concise form.

5 REFERENCES

[Ende97] Ender T.: "Object-Oriented Programming with REXX", John Wiley & Sons, New York et.al. 1997.

[Flat02a] Flatscher R.G.: "Applying the Object REXX Windows Scripting Engine with Windows Script Host", in: Proceedings of the „The 2002 International REXX Symposium“, April 28 - May 1, The REXX Language Association, Research Triangle Park, North Carolina 2002.

[Flat02b] Flatscher R.G.: "Overview of the Document Object Model (DOM) a.k.a. DHTML Under Windows", in: Proceedings of the „The 2002 International REXX Symposium“, April 28 - May 1, The REXX Language Association, Research Triangle Park, North Carolina 2002.

[Flat02c] Flatscher R.G.: "The Windows Script Host (WSH) - Architecture and Security Issues", in: Proceedings of the „E-World@Syria - From Technology to E-Business (ET2EB 2002)“, April 8 - April 9, Damascus, Syria 2002.

[VeTrUr96] Veneskey G., Trosky W., Urbaniak J.: "Object REXX by Example", Aviar, Pittsburgh 1996.

[W3LP] Lee Pedin's Object REXX for Windows scripts, URL:
`http://pragmaticlee.safedataisp.net/`

[W3MSORX] Microsoft Object REXX scripts, URL:
`http://www.microsoft.com/technet/scriptcenter/scripts/REXX/default.mspx`

[W3MSWSH] Microsoft Windows Script Host Homepage, URL:
`http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp`

[W3NetREXX] NetREXX homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2004-05-01):
`http://www2.hursley.ibm.com/netREXX/`

[W3ObjREXX] Object REXX homepage of IBM, URL (2004-05-01):
`http://www.ibm.com/software/ad/obj-REXX/`

[W3REXX] REXX homepage of the creator of the language, the IBM fellow Mike Cowlishaw, URL (2004-05-01): `http://www2.hursley.ibm.com/REXX/`

[W3REXXLA]REXX homepage of the “REXX Language Association”, URL (2004-05-01): http://www.REXXLA.org

Published in: Proceedings of the „2004 International REXX Symposium“, Böblingen, Germany, May 3nd - May 6th, 2004, The REXX Language Association, Raleigh N.C., 2004.

Presented at: „2004 International REXX Symposium“, Böblingen, Germany, May 4th, 2004.