MzCOM: Racket as a Windows COM Object

Version 9.4.0.3

Paul Steckler

October 27, 2026

`MzCOM.exe` is a Windows COM (i.e., Component Object Model) class wrapper for Racket.

During normal installation of MzCOM, the executable is registered as a COM object automatically. If that registration fails or if you move the Racket installation, re-register `MzCOM.exe` with

⟨*installation*⟩`\lib\MzCOM.exe /RegServer /v`

The `MzCOM.exe` executable will find DLLs and library collections relative to its own path.

1 Loading MzCOM

To load a COM object, COM hosts require a COM class name or a ProgID. MzCOM has the class name "MzObj Class" and the ProgID "MzCOM.MzObj.⟨*version*⟩", where ⟨*version*⟩ is 9.4.0.3.

In the Visual BASIC 6 environment, from the Project|References (VB6), check MzCOM 1.0 Type Library. In Visual BASIC .NET, choose Project|Add Reference, and from the COM tab, select MzCOM 1.0 Type Library. In your code, declare a variable, then assign to it:

```
DIM schemeObject AS MzObj
SET schemeObject = NEW MzObj
```

From Visual C++:

```
#include "mzcom.h"

CLSID clsid;
IMzObj *pIMzObj;

CoInitialize(NULL);
CLSIDFromProgID(L"MzCOM.MzObj.<version>",&clsid);
CoCreateInstance(clsid,NULL,CLSCTX_SERVER,IID_IMzObj, (void
**)&pIMzObj);
```

where <version> is the version number. You'll need the definition of IID_IMzObj (see §2 "GUIDs"). The header file "mzcom.h" is generated as "src\worksp\mzcom\" when building from the Racket source distribution. The above C/C++ code is for illustration; your actual code should check return values, of course.

Using mysterx to manipulate COM objects within Racket, you can load MzCOM with either

```
(cci/coclass "MzObj Class")
```

or

```
(cci/progid "MzCOM.MzObj.<version>")
```

Consult your documentation for loading MzCOM into other COM environments. MzCOM is compiled as a "dual-mode" class, meaning its methods may be called directly or by using OLE Automation.

2 GUIDs

When compiled from the Racket source distibrution, the directory `"src\worksp\mzcom\"` contains the file `"MzCOM_i.c"` that contains GUIDs for MzCOM. Those GUIDs are as follows:

```
const IID IID_IMzObj =
  {0xA604CBA8,0x2AB5,0x11D4,{0xB6,0xD3,0x00,0x60,0x08,0x90,0x02,0xFE}};

const IID LIBID_MZCOMLib =
  {0xA604CB9C,0x2AB5,0x11D4,{0xB6,0xD3,0x00,0x60,0x08,0x90,0x02,0xFE}};

const IID DIID__IMzObjEvents =
  {0xA604CBA9,0x2AB5,0x11D4,{0xB6,0xD3,0x00,0x60,0x08,0x90,0x02,0xFE}};

const CLSID CLSID_MzObj =
  {0xA3B0AF9E,0x2AB0,0x11D4,{0xB6,0xD2,0x00,0x60,0x08,0x90,0x02,0xFE}};
```

which represent the `IMzObj` interface, the MzCOM type library, the `IMzObjEvents` interface, and the `MzObj` class, respectively.

3 Methods

MzCOM support three COM methods:

- `void About()`

  Takes no arguments and displays an informational dialog.

- `BSTR Eval(BSTR input)`

  Takes and returns strings (specifically, `BSTR`s). The returned value is the result of evaluating the input expression, formatted as a string. The input string may contain several S-expressions. The embedded Racket updates its environment with each evaluation. Therefore, it is possible to define procedures in a call to `Eval`, and use the procedures in subsequent calls.

- `void Reset()`

  Resets the Racket environment to the initial environment. Also, the custodian for the primary Racket thread is invoked, shutting all its managed values.

4 Events

MzCOM supports a single event.

- `SchemeError()`

  Passed a string (specifically, a `BSTR`) that explains the error.

5 Errors

When an error occurs in MzCOM, it creates a COM error object. C and C++ clients can use `GetErrorInfo` to retrieve error information. Clients implemented in other languages typically have some equivalent means to obtain COM error information.

6 Evaluation thread

The Racket evaluator runs in a Win32 thread created when MzCOM is loaded. If an expression kills the primary Racket thread, as in

```
(kill-thread (current-thread))
```

then the evaluator Win32 thread is also killed. When that happens, subsequent calls to `Eval` will fail.

7 Acknowledgments

MzCOM was developed in response to a query by Andre Van Meulebrouck. Andre also did extensive testing with Visual BASIC.