MysterX: Legacy Support for Windows COM

Version 6.1.1

Paul Steckler

November 4, 2014

MysterX allows scripting of most COM components from Racket. A COM component can be scripted in MysterX if it supports OLE Automation via the `IDispatch` interface, and if it publishes type information using the `ITypeInfo` interface.

> **NOTE:** This library is deprecated; use `ffi/com` or `ffi/unsafe/com`, instead. MysterX formerly provided ActiveX support; we no longer support that ActiveX functionality, but see §5.12.3 “ActiveX Controls”.

```
(require mysterx)
```
package: `mysterx`

Contents

Index **11**

1 COM Methods and Properties

MysterX allows scripting of most COM components from Racket. A COM component can be scripted in MysterX if it supports OLE Automation via the `IDispatch` interface, and if it publishes type information using the `ITypeInfo` interface.

```
(com-all-coclasses) → (listof string?)
```

Returns a list of strings for all COM classes registered on a system.

```
(com-all-controls) → (listof string?)
```

Returns a list of strings for all COM classes registered on a system that have the `"Control"` subkey.

```
(cocreate-instance-from-coclass coclass
                                [where]) → com-object?
  coclass : string?
  where : (or/c (one-of/c 'local 'remote) string?) = 'local
(cci/coclass coclass [where]) → com-object?
  coclass : string?
  where : (or/c (one-of/c 'local 'remote) string?) = 'local
```

Returns an instance of *coclass*. This is useful for COM classes without a visual representation, or when a visual representation is not needed.

The optional argument *where* indicates a for running the instance, and may be `'local`, `'remote`, or a string indicating a machine name. See §5.12.1.7 “Remote COM servers (DCOM)” for more information.

```
(cocreate-instance-from-progid progid
                               [where]) → com-object?
  progid : string?
  where : (or/c (one-of/c 'local 'remote) string?) = 'local
(cci/progid progid [where]) → com-object?
  progid : string?
  where : (or/c (one-of/c 'local 'remote) string?) = 'local
```

Like `cocreate-instance-from-coclass`, but using a ProgID.

```
(com-get-active-object-from-coclass coclass) → com-object?
  coclass : string?
(gao/coclass coclass) → com-object?
  coclass : string?
```

Like cocreate-instance-from-coclass, but gets an existing active object (always local) instead of creating a new one.

```
(coclass obj) → string?
  obj : com-object?
```

Returns a string that is the name of the COM class instantiated by *obj*, or raises an error if the COM class is not known.

```
(progid obj) → string?
  obj : com-object?
```

Returns a string that is the name of the ProgID instantiated by *obj*, or raises an error if the COM class is not known.

```
(set-coclass! obj coclass) → void?
  obj : com-object?
  coclass : string?
```

Sets the COM class for *obj* to *coclass*. This is useful when MysterX COM event-handling procedures can obtain only ambiguous information about the object's COM class.

```
(set-coclass-from-progid! obj progid) → void?
  obj : com-object?
  progid : string?
```

Like set-coclass!, but using a ProgID.

```
(com-methods obj/type) → (listof string?)
  obj/type : (or/c com-object? com-type?)
```

Returns a list of strings indicating the names of methods on *obj/type*.

```
(com-method-type obj/type method-name) → (listof symbol?)
  obj/type : (or/c com-object? com-type?)
  method-name : string?
```

Returns a list of symbols indicating the type of the specified method in *obj/type*. See §2 "COM Types" for information on the symbols.

```
(com-invoke obj method-name v ...) → any/c
  obj : com-object?
  method-name : string?
  v : any/c
```

Invokes *method-name* on *obj* with *v*s as the arguments. The special value com-omit may be used for optional arguments, which useful when values are supplied for arguments after the omitted argument(s).

```
(com-get-properties obj/type) → (listof string?)
  obj/type : (or/c com-object? com-type?)
```

Returns a list of strings indicating the names of readable properties in *obj/type*.

```
(com-get-property-type obj/type
                       property-name) → (listof symbol?)
  obj/type : (or/c com-object? com-type?)
  property-name : string?
```

Returns a list of symbols indicating the type of the specified property in *obj/type*. See §2 "COM Types" for information on the symbols.

```
(com-get-property obj property ...+) → any/c
  obj : com-object?
  property : (or/c string?
                   (cons/c string? list?))
```

Returns the value of the final property by following the indicated path of *property*s, where each intermediate property must be a COM object.

Each *property* is either a property-name string or a list that starts with a property-name string and continues with arguments for a parameterized property.

```
(com-set-properties obj/type) → (listof string?)
  obj/type : (or/c com-object? com-type?)
```

Returns a list of strings indicating the names of writeable properties in *obj/type*.

```
(com-set-property-type obj/type
                       property-name) → (listof symbol?)
  obj/type : (or/c com-object? com-type?)
  property-name : string?
```

Returns a list of symbols indicating the type of the specified property in *obj/type*. See §2 "COM Types" for information on the symbols.

```
(com-set-property! obj property ...+ v) → void?
  obj : com-object?
  property : (or/c string?
                   (cons/c string? list?))
  v : any/c
```

Sets the value of the final property in *obj* to *v* by following the *property*s, where the value of each intermediate property is a COM object. A *property* can be a list instead of a string to represent a parameterized property and its arguments.

```
(com-help obj/type [topic]) → void?
  obj/type : (or/c com-object? com-type?)
  topic : string? = ""
```

Starts the Window Help system with help about the COM object or COM type. The optional *topic* is typically a method or property name.

```
(com-object-eq? obj1 obj2) → boolean?
  obj1 : com-object?
  obj2 : com-object?
```

Returns `#t` if the two COM objects are the same, `#f` otherwise.

```
(com-object? obj) → boolean?
  obj : com-object?
```

Returns `#t` if the argument is a COM object, `#f` otherwise.

2 COM Types

In the result of a function like com-method-type, a type 'mx-any standards for a character, real number, string, boolean, COM currency (as in com-currency?), COM date (as in com-date?), COM scode value (as in com-scode?), COM IUnknown value (as in com-iunknown?, or COM object (as in com-object?).}

```
(com-object-type obj) → com-type?
  obj : com-object?
```

Returns a type for a COM object.

```
(com-is-a? obj type) → boolean?
  obj : com-object?
  type : com-type?
```

Return #t if *obj* is of the type *type*.

```
(com-currency? v) → boolean?
  v : any/c
```

Returns #t if *v* is a COM currency value, #f otherwise.

```
(com-currency->number curr) → real?
  curr : com-currency?
```

Returns a number for *curr*.

```
(number->com-currency n) → com-currency?
  n : real?
```

Converts a number to a COM currency value. A currency value is repsented with a 64-bit two's-complement integer, though *n* may contain decimal digits. If *n* is too large, an exception is raised.

```
(com-date? v) → boolean?
  v : any/c
```

Returns #t if *v* is a COM date value, #f otherwise.

```
(com-date->date d) → date?
  d : com-date?
```

Converts a COM date to an instance of the date structure type. In the result, the dst? field is always #f, and the time-zone-offset field is 0.

```
(date->com-date d) → com-date?
  d : date?
```

Converts a `date` instance to a COM date value.

```
(com-scode? v) → boolean?
  v : any/c
```

Returns `#t` if `v` is a COM scode value, `#f` otherwise.

```
(com-scode->number sc) → integer?
  sc : com-scode?
```

Converts a COM scode value to an integer.

```
(number->com-scode n) → com-scode?
  n : integer?
```

Converts a number to a COM scode value. The number must be representable as a 32-bit two's-complement number, otherwise an exception is raised.

```
(com-iunknown? v) → boolean?
  v : any/c
```

Returns `#t` if `v` is a COM IUnknown value, `#f` otherwise.

```
com-omit : any/c
```

Used with `com-invoke` to represent an argument that is not provided.

3 COM Events

COM events are generated by COM objects. Unlike HTML events, there is no fixed set of COM events, though there are "stock" events that many COM objects support. MysterX allows the programmer to write handlers for both stock and custom events.

```
(com-events obj/type) → (listof string?)
  obj/type : (or/c com-object? com-type?)
```

Returns a list of strings naming the events supported by *obj/type*.

If calling this procedure results in an error indicating that the COM object's coclass is ambiguous, try using either `set-coclass!` or `set-coclass-from-progid!`, then retry `com-events`.

```
(com-event-type obj/type ev) → (listof string?)
  obj/type : (or/c com-object? com-type?)
  ev : string?
```

Returns the type of an event handler for the event *ev* generated by the particular COM object/type *obj/type*. The return type of all COM event handlers is void.

See also `com-events` for dealing with a COM object that has an ambiguous class.

```
(com-register-event-handler obj ev f) → void?
  obj : com-object?
  ev : string?
  f : (any/c . -> . any)
```

Registers *f* as event handler for the event *ev* when generated by *obj*. The type of argument supplied to *f* depends on the event; the result of *f* is always discarded.

See also `com-events` for dealing with a COM object that has an ambiguous class.

```
(com-unregister-event-handler obj ev) → void?
  obj : com-object?
  ev : string?
```

Unregisters any event handler for the event *ev* that is generated by the COM object *obj*.

4 Version

```
(mx-version) → string?
```

Returns a string indicating the version of MysterX.

Index