ActiveX implementation in Imagine - first trial

This document briefly describes how we implemented the first ActiveX support in Imagine (build 245).

In Imagine there is a limited support to communicate with an ActiveX control (or better to say OLE automation server or OLE control).
The main restrictions are:

- The connected ActiveX has not visual representation. So the current support is not intended for real ActiveX controls, but rather for libraries of procedures/functions implemented as ActiveX (or just as OLE automation servers).
- Only procedures/functions/properties of single types are supported. I.e. functions, which parameters and/or results are arrays, or any kind of pointers, cannot be called from Imagine.
- The parameters sent to functions of the ActiveX are always converted to strings, so the ActiveX itself must implement the safecall convention to handle this situation.
- The events defined in the ActiveX control cannot be used from Imagine

Creating and using an ActiveX object in Imagine

There is a new primitive class named **oleobject**.
To create an Imagine object, which corresponds to an ActiveX object (the Imagine wrapper for the ActiveX object) you must create an instance of the **oleobject** class giving it the name of the ActiveX object under which it is known to the registry as the value of **comname** setting. The **comname** setting must be set in the input list of the **new** procedure and its value cannot be changed later during the lifetime of the wrapper object.
For example the LEGO RCX brick driver ActiveX control is registered under the name SPIRIT.SpiritCtrl.1 therefore if we want to create an instance of it and an Imagine wrapper for that instance, we can write:

```
new "oleobject [comname SPIRIT.SpiritCtrl.1]
```

This way an object named **ole1** is created. It corresponds to the ActiveX control in such a way that all properties of the ActiveX control are accessible as Imagine-like settings of the **ole1** object and all procedures/functions of the ActiveX control can be called from Imagine using Imagine syntax.
As for any other Imagine object we can directly set the values of any other settings in the call to **new** and we can also give a user-friendlier name to the newly created control. So in case of the RCX driver we can write:

```
new "oleobject [comname SPIRIT.SpiritCtrl.1 comportno 1 pbrick 1
linktype 0 name brick]
```

Note that the value of **comportno** must be set according to the actual port where the RCX's tower is actually connected.
To be able to use the newly created object meaningfully, we must have its documentation i.e. we must know what do its settings mean and which procedures/functions are defined for that object.

A longer example with RCX

Creting the object:

```
new "oleobject [comname SPIRIT.SpiritCtrl.1 comportno 1 pbrick 1
linktype 0 name brick]
```
To initialise it you must call:
```
pr brick'initcomm
```
or if you do not want to see the result:
```
ignore brick'initcomm
```
Note that many of the functions have a Boolean result, which can be ignored in many situations. In C you can always call a function without using its result (i.e. you call it as a procedure). But in Logo it would result in an error. Therefore you must either do something with the result or use the **ignore** primitive to consume the result.
After this we can call functions, which set outputs, for example:
```
ignore brick'on 1
ignore brick'setpower 1 2 3
ignore brick'off 1
```
Put a lamp on output B to see the effect.

To manipulate inputs is a bit more work because type and mode of the sensor must be set. For example if we connect an angle sensor to input 1 then we must infrom the brick like this:
```
pr brick'setsensortype 0 4
pr brick'setsensormode 0 7 0
```
Then we can read the values:
```
pr brick'poll 9 0
```
Or ve can create a textbox named **text1** and then write:
```
every 500 [text1'setvalue brick'poll 9 0]
```

When finishing the work it seems that a call to **closecomm** is useful to make, but it is not necessary:
```
ignore brick'closecomm
```

Known limitations and problems

- The current implementation brings all the definitions contained into the ActiveX library into an Imagine wrapper object. It means that the Imagine object, which corresponds to the ActiveX object holds all type information - it is in the same time a class and an instance, in a way. It means that you cannot redefine any of the ActiveX's procedures or functions because then you would lose its declaration in Imagine.
  In the future we plan to implement a way how to import the object types from an ActiveX library as Imagine classes and just then create objects of those classes. This approach would better serve for more advanced object-oriented uses in situations where the user will have more instances of the same ActiveX type. But in cases where we need only one instance and we will not want to redefine its procedures the current implementation gives a simple and efficient solution.
- During testing of the described ActiveX with RCX we sometimes experienced loss of connection (which can be detected as receiving **false** or an error message as the result of any call, which should set or get any information and also the result **false** of the **pbaliveornot** call). Each time such problems occurred we had to restart the computer to get rid of them. We are not sure if this is a physical problem or it has to do something with the implementation of the ActiveX support. Please report if you have similar problems and try to give more information when does it happen.