# 1

# Getting Started

The following step by step tutorials will help you create your first EiffelCOM projects. An EiffelCOM project can consist either in accessing an existing COM component from Eiffel or in creating a new COM component.

## 1.1 CREATING A NEW COM COMPONENT

This first tutorial describes the steps involved in creating a COM component in Eiffel. These components can be either DLLs (in-process) or EXEs (out-of-process). This tutorial will focus on creating an in-process component. This component, called *StringManipulator*, will expose one interface *IString* that include the functions *ReplaceSubstring* and *PruneAll* corresponding respectively to the features *replace_substring* and *prune_all* of the EiffelBase *STRING* class. *IString* also includes the property *String* that represents the string being manipulated. This property can be set or read. Other languages will then be able to access these features using this component.

### Step by step instructions

- Launch the EiffelCOM Wizard (from the *Start* menu, open *Programs* then *EiffelXX* and click *EiffelCOM Wizard*)
- Choose *Create a new project*
- Click *Next*
- Choose *Server code for a new component*
- Click *Next*
- Click the top *Browse* button and open the file *$EIFFEL4\examples\com\wizard\string_manipulator\string_manipulator.idl* where *$EIFFEL4* represents the path to your EIFFEL45 directory.
- Click the bottom *Browse* button and select the directory where you want to create the project (later referenced as *destination folder*). Choose *$EIFFEL4\examples\com\wizard\string_manipulator\generated* where *$EIFFEL4* represents the path to your EIFFEL45 directory.
- Click *Next*
- Choose *Virtual Table, Standard Marshalling*

- Click *Next*
- Click *Finish*
- Wait until the wizard is done.

## First look at the generated code

The first interesting file you might want to look at is the file *generated.txt* in the destination folder. It includes a list of all the files that the wizard generated. When done, the wizard automatically launches EiffelBench with the generated project, you can use EiffelBench to browse the system and get an idea of the class hierarchy. The most interesting classes are the ones in relation with the generated coclass *STRING_MANIPULATOR_COCLASS*. This class inherits from *ISTRING_INTERFACE* corresponding to the IString interface. The coclass is deferred since we are creating a project for a server. It has one descendant *STRING_MANIPULATOR_COCLASS_IMP* that implements the deferred features of the interface. The default implementation returns the error *E_notimpl* to the client. This error means that the function is not implemented on that server.

## implementing the component

We want to do something more interesting than just returning an error to our client an are going to edit the implementation of the features in *STRING_MANIPULATOR_COCLASS_IMP. This class will not be overwritten by future executions of the wizard*. There is an implementation of the coclass in *$EIFFEL4\examples\com\wizard\string_manipulator\server*. Just copy this file over to *$EIFFEL4\examples\com\wizard\string_manipulator\generated\server\component* and quick melt your project. Voila!, you have your first EiffelCOM component up and running.

## Tips

- If you are to develop an EiffelCOM component, you will most probably need to set the registry entry *MELT_PATH*. This value needs to be setup in the case the system is not launched from the directory containing the byte code (by default *EIFGEN\W_code*). The EiffelCOM component client will launch the server from its current location and will cause the server to crash if the *MELT_PATH* key is not setup properly. To put this value in your registry, create a key under *HKEY_CURRENT_USER\SOFTWARE\ISE\Eiffel45* with the same name as the name of your project (e.g. *string_manipulator*) and add the string value named *MELT_PATH* containing the path to the byte code (*.melted* file). You will find an example of such entry in the file *melted_path.reg* in *$EIFFEL4\wizard\config*. You can edit this file as needed (replace *string_manipulator* with the name of your system and change the path to point on the correct location using '\\' as directory separator) and double click it to enter the information in your registry.

- To test your component, you need first to register it. If the component is in-process then it is registered via the *regsvr32* utility using the following syntax: *regsvr32 system_name.dll* where *system_name* is the name of the dll (e.g. *string_manipulator*). If the component is out-of-process then it is register using the following syntax: *system_name.exe -Regserver* where *system_name* is the name of the component executable file.

For the purpose of this tutorial you will need to register the component with the following command (run from a dos console): *regsvr32 string_manipulator.dll* and to enter the information in *$EIFFEL4\wizard\config\melted_path.reg*. You will have to change the content of this file if you did not install *Eiffel* under *C:*.

## 1.2 ACCESSING A COMPONENT

Now that we have build a component, we are going to reuse it from another Eiffel project. The same exact steps should be followed for components written in any language.

### Step by step instructions

- Launch the wizard
- Choose *Create a new project* and click *Next*
- Click *Next* again
- Click the top *Browse* button and open the file *$EIFFEL4\examples\com\wizard\string_manipulator\string_manipulator.idl* where *$EIFFEL4* represents the path to your EIFFEL45 directory. You could also choose to open the type library (*.tlb*) that was generated by the wizard when CREATING A NEW COM COMPONENT in the destination folder of that first project.
- Click the bottom *Browse* button and select the directory where you want to create the project (later referenced as *destination folder*). Choose *$EIFFEL4\examples\com\wizard\string_manipulator\generated* where *$EIFFEL4* represents the path to your EIFFEL45 directory.
- Click *Next*
- Click *Finish*
- Wait until the wizard is done.

### First look at the generated code

In the case of a client the wizard will precompile the generated code and open EiffelBench on the precompilation project. *Note: a precompilation project is read-only, you will need to start another EiffelBench to reuse the generated classes.* The interesting classes are all related to the coclass proxy *STRING_MANIPULATOR_PROXY*. The proxy is the Eiffel class that gives access to the component. Each feature on the proxy calls the corresponding interface function

on the component. You can use the EiffelBench opened by the wizard to browse through the generated classes and get an idea of the class hierarchy.

## Implementing a client

We are now going to implement a client of the *StringManipulator* component. Open a new EiffelBench that will be used to create the client project. Create the project in *$EIFFEL4\examples\com\wizard\string_manipulator\client* using the ace file found in that directory. Freeze and run the project. You are now accessing the previously built component and calling functions on its interfaces!. The interesting class is *MY_STRING_MANIPULATOR* which inherits from the generated *STRING_MANIPULATOR_PROXY* and redefine the feature *replace_substring_user_precondition*. The generated interfaces will include contracts for each exposed function. You can redefine the *user_precondition* features to implement your own preconditions.

## Contracts

Contracts can be broken directly on the proxy in which case you will get a standard contract violation or in the server. If contracts are broken on the server then the exception will be forwarded by the EiffelCOM runtime to the client. The feature *replace_substring_user_precondition* in *MY_STRING_MANIPULATOR* includes the following commented line:

```
-- Result := True
```

Un-comment this line by removing the preceding '--' and comment out the rest of the feature. Now the contract of the *replace_substring* feature is wrong and erroneous calls can be made. Quick melt the changes and run the client. Enter some invalid numbers in the fields used to call this feature. After you click *Replace* you will see an error message box warning you that a precondition was violated on the server side. This is how you can use contracts 'over the wire'. The preconditions was violated in the server, this exception was caught by the EiffelCOM runtime and sent back to the client.

## Summary

You now have the basic knowledge needed to run the EiffelCOM wizard and produce or access COM components. The benefits of using the wizard are numerous including a fast and easy generation of "plumbing" code as well as enhanced debugging capabilities through the use of contracts. The wizard is also very generic, it can generate or wrap any kind of components. We hope you enjoy using it as much as we enjoyed developing it,

The EiffelCOM team.