Developing Plug-ins

This chapter covers the following topics:

- Creating a Plug-in
- Debugging a Plug-in
- Deploying a Plug-in
- Developing Plug-ins in Other Programming Languages

Creating a Plug-in

To create a new plug-in, proceed as described in the section *Quick Start*.

Debugging a Plug-in

Plug-ins written in Natural are running in server processes distinct from the process that runs Natural Studio. Therefore, in order to debug a plug-in, remote debugging must be used. See the *Debugger* documentation for information on how to set up and use remote debugging in general.

The following topics describe the specific activities required to debug a plug-in using the remote debugger.

- Single Server
- Shared Server

Single Server

A plug-in that was created with the option **Single server** runs in its own Natural server process, distinct from all other plug-ins. This Natural server process is started when the plug-in is activated in the Plug-in Manager. In order to debug such a plug-in, this server process must be configured to run under the remote debugger.

Plug-ins are running under the Natural parameter file `NATPARM`. Therefore, the following configuration must be applied to the Natural parameter file `NATPARM` before activating the plug-in in the Plug-in Manager:

- `RDACTIVE` must be set to "ON" to enable remote debugging.
- `RDNODE` must be set to the name of the machine where the Natural Remote Debugging Service is running. Normally this is the machine you are working on.
- `RDPORT` must be set to "2600" (default) or another port number, depending on which port you have installed the Natural Remote Debugging Service.

See also *Remote Debugging* in the *Configuration Utility* documentation.

Now, when you activate the plug-in in the Plug-in Manager, the Natural debugger is started and stops on the first statement in the plug-in's method `OnActivate`. At this point, you can set breakpoints as necessary.

Shared Server

A plug-in that was created without the option **Single server** runs in the same Natural server process as the Plug-in Manager. This Natural server process is started when the Plug-in Manager is activated. This happens during the start of the Natural Studio session. We call this mode "shared server". In order to debug such a plug-in, this common server process must be configured to run under the remote debugger.

Plug-ins are running under the Natural parameter file `NATPARM`. Therefore, the following configuration must be applied to the Natural parameter file `NATPARM` before starting Natural Studio:

- `RDACTIVE` must be set to "ON" to enable remote debugging.
- `RDNODE` must be set to the name of the machine where the Natural Remote Debugging Service is running. Normally this is the machine you are working on.
- `RDPORT` must be set to "2600" (default) or another port number, depending on which port you have installed the Natural Remote Debugging Service.

See also *Remote Debugging* in the *Configuration Utility* documentation.

Now, when you start Natural Studio, the Natural debugger is started and stops on the first statement in the Plug-in Manager's method `OnActivate`. At this point, you can load the source code of your own plug-in's method `OnActivate` and set breakpoints as necessary.

Deploying a Plug-in

To deploy a plug-in written in Natural to other machines, the Object Handler is used. Start the Object Handler, select all modules that belong to your plug-in and unload them into a sequential file. Do not forget to unload the modules `INSTALL` and `INSTAL-N` along that were generated during the creation of the plug-in. These modules are required to install the plug-in in the target environment.

In the target environment, load the sequential file again using the Object Handler. Execute the program `INSTALL` in the plug-in library and restart Natural to make the new plug-in visible in the Plug-in Manager.

Plug-ins written for a specific version of Natural Studio should only be installed under this version.

Developing Plug-ins in Other Programming Languages

Because Natural Studio plug-ins are ActiveX components, you can develop plug-ins in any language that allows creating ActiveX components. This section contain some hints on how to proceed. Please refer to the documentation of the respective development environment for details.

To develop a plug-in using Microsoft Visual Basic

1. Create a new project of type "ActiveX DLL".
2. Add references to the type libraries `NATURALSTUDIOPLUGIN.TLB` and `NATURALSTUDIOAUTO.TLB`. These type libraries describe the Plug-in Interface and the Natural Studio Interface respectively.
3. Add the following code to the class that implements your plug-in:

   ```
   Implements INaturalStudioPlugIn
   Implements INaturalStudioPlugInTree
   ```

4. Implement the interface methods. The method bodies may initially be left empty.
5. Build the project and register the resulting DLL using `regsvr32`.
6. In order to make the ActiveX component visible in the Natural Studio Plug-in Manager, add an additional registry entry as shown in the example below.

   ```
   [HKEY_LOCAL_MACHINE\SOFTWARE\Software AG\Natural\n.n\Plug-ins\{617D1BE3-D1D8-4EAC-9633-4FF2842D8B6C}]
   @="Visual Basic Minimal Plug-in"
   "CLSID"="{617D1BE3-D1D8-4EAC-9633-4FF2842D8B6C}"
   "ProgID"="MinimalPlugIn.PlugInClass"
   ```

   n.n in the first line of the above example stands for the current version number of Natural.

   - Replace both occurrences of the CLSID in the example by the CLSID of your ActiveX component.
   - Replace the ProgID in the example by the ProgID of your ActiveX component.

   The name in the line starting with `@=` will be displayed in the Plug-in Manager.

To develop a plug-in using Microsoft Visual C++ and the ATL

1. Create an ATL project using the ATL COM Wizard.
2. Create an ATL object in the ATL project.
3. Choose **Implement Interface**….
4. Select the type library `NATURALSTUDIOPLUGIN.TLB`. This type library describes the Plug-in Interface.
5. Select the interfaces `INaturalStudioPlugIn` and `INaturalStudioPlugInTree`.
6. Implement the interface methods. The method bodies may initially just return "S_OK".
7. Build the project and register the resulting DLL using `regsvr32`.
8. In order to make the ActiveX component visible in the Natural Studio Plug-in Manager, add an additional registry entry as shown in the example below.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Software AG\Natural\n.n\Plug-ins\{617D1BE3-D1D8-4EAC-9633-4FF2842D8B6C}]
@="C++ ATL Minimal Plug-in"
"CLSID"="{617D1BE3-D1D8-4EAC-9633-4FF2842D8B6C}"
"ProgID"="MinimalPlugIn.PlugInClass"
```

n.n in the first line of the above example stands for the current version number of Natural.

- Replace both occurrences of the CLSID in the example by the CLSID of your ATL component.
- Replace the ProgID in the example by the ProgID of your ATL component.

The name in the line starting with `@=` will be displayed in the Plug-in Manager.