# Natural

## Extending Natural Studio with Plug-ins

Version 9.3.3

October 2025

ADABAS & NATURAL

This document applies to Natural Version 9.3.3 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.



**Document ID: NATWIN-PLUGIN-933-20251014**

# Table of Contents

# Preface

This documentation describes how to develop your own plug-ins. It is organized under the following headings:

| | |
|---|---|
| **What are Natural Studio Plug-ins?** | General information on the plug-ins that can be developed in order to extend Natural Studio functionality. |
| **Quick Start** | Prerequisites for developing plug-ins. How to create a minimal plug-in. How to install and activate the minimal plug-in. How to extend the generated code of the minimal plug-in with your own code. How to deactivate and uninstall the minimal plug-in. |
| **Plug-in Interfaces** | How Natural Studio interacts with a plug-in. |
| **Natural Studio Interfaces** | How a plug-in interacts with Natural Studio. |
| **Developing Plug-ins** | How to use remote debugging with a plug-in and how to deploy a plug-in to another machine. Some hints on how to develop plug-ins in languages that allow creating ActiveX components. |
| **Plug-in Example** | How to use the plug-in example. Information on the library which contains the source code of the plug-in example. |
| **Interface Reference** | Descriptions of all interfaces (plug-in interfaces and Natural Studio interfaces) in alphabetical order. |
| **DTDs** | Because of their length the DTDs used in several Natural Studio interfaces are provided separately and are listed in this part. |

# 1 About this Documentation

## Document Conventions

| Convention | Description |
|---|---|
| **Bold** | Identifies elements on a screen. |
| `Monospace font` | Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties. |
| *`Italic`* | Identifies:<br><br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br><br>Text you must type in.<br>Messages displayed by the system.<br>Program code. |
| { } | Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols. |
| \| | Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the \| symbol. |
| [ ] | Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols. |
| ... | Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...). |

## Online Information and Support

### Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

### Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

### Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://techcommunity.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

**Product Support**

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

## Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

# I

# 2 What are Natural Studio Plug-ins?

Natural Studio offers a set of functions, tools and utilities. You may need further functions or a tool that does not yet exist in Natural Studio, or find yourself repeatedly doing the same task. With Natural Studio plug-ins, you can create your own extensions of Natural Studio functionality. A variety of Natural Studio components has been developed using this technique, for example the plug-ins XRef Evaluation and Object Description.

A Natural Studio plug-in is an ActiveX component that provides specific interfaces. Using these interfaces, Natural Studio interacts with the plug-in. Vice versa, Natural Studio provides specific interfaces through which a plug-in can interact with Natural Studio. These interfaces are presented in the form of an object model that allows access to most areas of Natural Studio.

Because Natural Studio plug-ins are ActiveX components, you can develop plug-ins in any language that allows you to create ActiveX components. If you implement your plug-in in Natural, however, you can integrate it most closely into Natural Studio. In Natural, you implement a plug-in as a Natural class.

One type of plug-in is a wizard. A wizard is a step-by-step instructive program that leads users through a specific procedure. An example of this plug-in type is the Application Wizard. Another type of plug-in is an editor. An editor allows creating and modifying development objects in an MDI document window. An example of this plug-in type is Object Description.

See also *Plug-in Manager* in the documentation *Using Natural Studio*.

# 3 Quick Start

In order to understand the structure of a Natural Studio plug-in and its interaction with Natural Studio, it is instructive to create and explore a minimal, but fully operational plug-in. Later this plug-in will be extended. Perform the steps described in the topics below.

## Prerequisites

In order to develop a plug-in, you need the example library SYSEXPLG as a basis. This library contains the plug-in example and some central definitions and modules that are common to all plug-ins.

Plug-ins are always executed under the Natural parameter file NATPARM. While developing a plug-in, you need the same Natural environment during editing, cataloging, debugging and execution of your plug-in. Especially make sure that Natural Studio runs with the same system file settings as specified in the Natural parameter file NATPARM.

### To check the prerequisites

1 Make sure that the library SYSEXPLG is available.

2 Invoke the Configuration Utility and make sure that the libraries SYSEXPLG and SYSEXT are defined as steplibs in the Natural parameter file NATPARM.

3 This step applies only after a first-time installation of Natural.

In the Configuration Utility open **Natural Configuration Files > Global Configuration File > System Files**.

You will see an entry with the alias name "PLUGINS". This system file, which is part of the plug-in environment must be used as the FUSER of the Natural installation.

**Note:** This new "PLUGINS" entry is only available after a first-time installation of Natural. It is not available with an update installation or when Natural is installed in parallel to an older version of Natural.

4 This step applies only after an update installation of Natural or when a current version of Natural is installed in parallel to an older version of Natural.

After an update installation or when a current version of Natural is installed in parallel to an older version of Natural, the FUSER setting of the plug-in environment has to be added manually to the global configuration file:

- In the Configuration Utility open **Natural Parameter Files > NATPARM > System Files > User System File**.
- In the **FUSER** tab change the values for DBID and FNR to match those of the FUSER of your plug-in environment.

**Important:** Never use the FNAT of the plug-in environment as the FUSER of Natural since an update installation might delete the FNAT.

5 Save your configuration and exit the Configuration Utility.

6 Plug-ins are always executed in the plug-in environment (for example *C:\Program Files (x86)\Common Files\Software AG\Natural\V1*) under the Natural parameter file `NATPARM`.

While developing plug-ins, Natural for Windows must be started with the FUSER of the plug-in environment.

**Important:** Please be aware of the fact that the plug-in environment is independent from the Natural Studio environment. When a plug-in extends Natural Studio or retrieves data from Natural Studio, this should always be performed via the exposed interfaces.

## Creating a Minimal Plug-in

Plug-ins are created using the Plug-in Manager.

### To create a plug-in

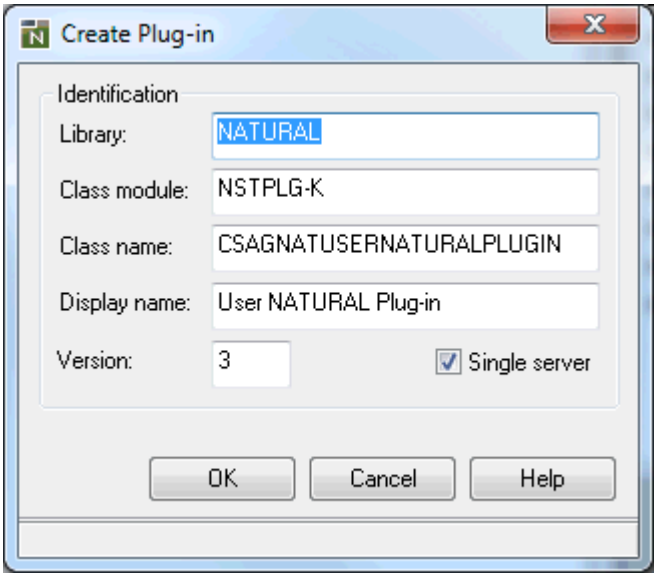
1 Make sure that the ***Prerequisites*** are met.

2 Make sure that plug-in activation has been enabled. See *Workspace Options* in the documentation *Using Natural Studio*.

3 Invoke the Plug-in Manager as described in *Invoking the Plug-in Manager* in the documentation *Using Natural Studio*.

4 Invoke the context menu and choose **New**.

The following dialog box appears.




The entries that are proposed in the different text boxes contain your user ID.

5 Specify all the following information:

**Library**

Enter the Natural library into which the plug-in shall be generated. You should ideally use a new library for each plug-in project. If the library is not empty, you will receive a warning. If you generate the plug-in anyway, existing modules will be replaced without further warnings.

**Note:** Do not use a library name starting with `SYS` since this plug-in library will then be created into the FNAT of the plug-in environment which is not permitted.

**Class module**

The plug-in consists basically of a Natural class. Choose an eight character name for the class module and enter it here.

**Class name**

This name will be used as class name in the `DEFINE CLASS` statement. Choose a 32 character class name and enter it here. This class name combined with the version number will be used as ProgID in the system registry when the plug-in is installed. Therefore you must use a name that is unique among all ActiveX components that are installed on the machine. It is good and common practice to prefix the name with an abbreviation of your company. For instance the class names of the plug-ins delivered with Natural Studio all start with "CSAGNAT".

**Display name**

This name will be used to display the plug-in in the Plug-in Manager.

**Version**

The version number specified here is combined with the class name specified above to form the ProgID of the plug-in, for example "CSAGNATUSERNATURALPLUGIN.3".

Different plug-ins with the same class name and different version numbers can coexist in one installation.

**Single server**
If this check box is selected, the new plug-in will run in an own Natural server process, distinct from all other plug-ins. This is required only if the plug-in uses **generic document windows**.

If this check box is not selected, the plug-in will run in the same server process as the Plug-in Manager. This saves an extra Natural server process during execution of the plug-in. However, it does not allow the usage of generic document windows.

6 Choose the **OK** button to generate the plug-in into the specified library. This is a minimal plug-in which you can extend with your own code (this is explained later in this section).

If an error occurs during the generation process, check the generation log. A common reason for errors is that the example library `SYSEXPLG` is not available, is not set as a steplib or was manually modified. In such a case, you have to reinstall the example library and check the steplib assignment.

7 In order to register and activate the plug-in, proceed as described in *Installing and Activating the Minimal Plug-in* below.

## Transferring a Plug-in From Natural 6 to Natural 8

This section describes how to transfer a custom plug-in running in a Natural Version 6 environment to a Natural Version 8 or higher environment.

**Note:** Following this approach, the custom plug-in will still run in the Natural Version 6 environment.

### To transfer a plug-in

1 Start Natural Version 6.x and unload the custom plug-in library to the file system using the *Unload Wizard* of the *Object Handler*.

2 Stop Natural Version 6.x.

3 Start the Natural Version 8.x Configuration Utility.

4 Adjust the steplibs as described in *Prerequisites*.

5 Add the FUSER setting of the plug-in environment manually to the global configuration file:

- In the Configuration Utility open **Natural Parameter Files > NATPARM > System Files > User System File**.

- In the **FUSER** tab change the values for DBID and FNR to match those of the FUSER of your plug-in environment.

  **Important:** Never use the FNAT of the plug-in environment as the FUSER of Natural since an update installation might delete the FNAT.

6 Save your configuration and exit the Configuration Utility.

7 Start Natural Version 8.x.

8 Load the custom plug-in library to the file system using the *Load Wizard* of the *Object Handler*.

**Note:** If the custom plug-in ported to Natural Version 8.x is located in an FNAT library (library name starting with `SYS`), the library must be renamed to an FUSER library.

9 Delete incompatible objects:

- Delete the `INSTAL-N` object of the ported custom plug-in library.
- Delete the *Resources* folder containing a *.reg* and a *.log* file.

10 Use a new GUID and a new version number:

- Generate a new GUID by creating a temporary plug-in library and copy the GUID into the clipboard.
- Open the `INSTALL` program located in the custom plug-in library.
- Init the `#CLSID` variable with the newly generated GUID value.
- Increment the `#VERSION` variable value.
- SAVE and STOW the `INSTALL` program.

11 Open the custom plug-in class in the custom plug-in library.

- Init the `#CLSID` variable with the newly generated GUID value.
- Increment the `#VERSION` variable value.
- SAVE and STOW the custom plug-in class.

12 In order to register and activate the plug-in, proceed as described in ***Installing and Activating the Minimal Plug-in*** below.

## Installing and Activating the Minimal Plug-in

When the minimal plug-in has been created as described above, it can be installed. When it has been installed, it can be activated.

The advantage of an activated plug-in is that you can immediately test whether your own code that you add to the plug-in works as intended.

### To install a plug-in

1 Execute the program `INSTALL` that was created in the library specified during the creation of the plug-in.

2 Restart Natural Studio to make the new plug-in visible in the Plug-in Manager.

**Note:** The next time you execute the program `INSTALL`, the plug-in is uninstalled.

### To activate a plug-in

1 Invoke the Plug-in Manager.

2 Activate the new plug-in as described in *Activating and Deactivating a Plug-in* in the documentation *Using Natural Studio*.

**Note:** When you define automatic activation mode for this plug-in, the plug-in will be activated each time you start Natural Studio. See *Defining Automatic or Manual Activation Mode for a Plug-in* in the documentation *Using Natural Studio*.

## Exploring the Minimal Plug-in

Log on to the library into which the plug-in was generated and open the generated class in the Class Builder. You will notice that the plug-in is just a Natural class that implements two specific interfaces, namely `INaturalStudioPlugIn` and `INaturalStudioPlugInTree`. These interfaces are specified in the interface modules (copycodes) `NSTPLG-I` and `NSTPLT-I`, which are contained in the example library `SYSEXPLG` and are shared by all plug-ins.

The minimal plug-in leaves most of the methods of these interfaces empty. In fact it really implements only two methods: `OnActivate` and `OnDeactivate` of the interface `INaturalStudioPlugIn`. These methods are of specific interest: Natural Studio calls the method `OnActivate`, when the user chooses the command **Activate** in the Plug-in Manager. `OnDeactivate` is called when the user chooses the command **Deactivate** in the Plug-in Manager.

If you open the method bodies of `OnActivate` and `OnDeactivate` in the Class Builder, you will notice that the minimal plug-in does nothing other than indicating its activation and deactivation by opening a message box. A real plug-in will of course use these methods to prepare itself for operation and to initialize and uninitialize its state. In the following section, we will see what this can mean.

## Extending the Minimal Plug-in

The following topics are covered below:



### Adding a Command

In order to interact with the user, the plug-in must define commands and present them to the user in menus or toolbars. Usually this will be done in the method `OnActivate`. Natural Studio passes a handle to the Natural Studio interface `INatAutoStudio` to the plug-in. The plug-in will store this handle and use it to access Natural Studio during further method calls.

**To add a command**

- As an example, add the code which is indicated in bold to the method `OnActivate`:

```
define data
parameter using nstact-a
object using nsttmp-o
local
1 #controlbars handle of object
1 #commands handle of object
1 #command handle of object
1 #toolbars handle of object
1 #toolbar handle of object
end-define
*
* Keep the Natural Studio Automation interface in mind.
#studio := nstact-a.iNatAutoStudio
* Show that we are coming up.
send "MessageBox" to #studio
with "Activating plug-in!" "Natural Studio Plug-in"
*
* Add a command.
#controlbars := #studio.ControlBars
#commands := #controlbars.Commands
send "Add" to #commands
```

```
with 100 "My Command" 1
return #command
*
* Select a toolbar.
#toolbars := #controlbars.Toolbars
send "Item" to #toolbars
with "Tools"
return #toolbar
*
* Insert the command.
send "InsertCommand" to #toolbar
with #command
*
end
```

This code sequence creates a command with the internal identifier "100" and inserts it into the Tools toolbar. Whenever the user chooses the new toolbar button, Natural Studio sends the command identifier "100" to the method `OnCommand` of the interface `INaturalStudioPlugIn`.

### Enabling the Command

Initially, Natural Studio shows the new command disabled. In order to make the command available to the user, the plug-in must implement a command status handler. In the command status handler, the plug-in can check any condition necessary to enable the command. In particular, it has access to the interface `INatAutoStudio` to perform operations in Natural Studio. In the simplest case, the plug-in enables the command without any condition.

#### To enable the command

- As an example, add the code which is indicated in bold to the method `OnCommandStatus` of your plug-in:

```
define data
parameter using nstcst-a
object using nsttmp-o
end-define
*
decide on first nstcst-a.Command
  value 100
    nstcst-a.Enabled := True
  none
    ignore
end-decide
*
end
```

### Handling the Command

In order to react to the command, the plug-in must implement a command handler. In the command handler, the plug-in can do anything necessary to implement the command. In particular, it has access to the interface `INatAutoStudio` to perform operations in Natural Studio.

#### To handle the command

- As an example, add the code which is indicated in bold to the method `OnCommand` of your plug-in:

```
define data
parameter using nstcmd-a
object using nsttmp-o
local
1 #objects handle of object
1 #progs handle of object

end-define
*
decide on first nstcmd-a.Command
  value 100
      #objects := #studio.Objects
      #progs := #objects.Programs
      send "Add" to #progs with 1009
  none
      ignore
end-decide
*
end
```

  Now when the user chooses the new toolbar button, the plug-in opens the program editor with an untitled program.

## Deactivating and Uninstalling the Minimal Plug-in

If you do not want to work with your minimal plug-in any longer, you can deactivate it. If you want to remove your minimal plug-in from the Plug-in Manager, you have to uninstall it.

#### To deactivate a plug-in

1. Invoke the Plug-in Manager.
2. Deactivate your minimal plug-in as described in *Activating and Deactivating a Plug-in* in the documentation *Using Natural Studio*.

**Note:** When automatic activation mode has been defined for this plug-in, the plug-in will be activated again the next time you start Natural Studio. See *Defining Automatic or Manual Activation Mode for a Plug-in* in the documentation *Using Natural Studio*.

### To uninstall a plug-in

1 Execute the program `INSTALL` that was created in the library specified during the creation of the plug-in.

2 Restart Natural Studio to remove the plug-in from the Plug-in Manager.

**Note:** The next time you execute the program `INSTALL`, the plug-in is installed again.

# 4 Plug-in Interfaces

Natural Studio accesses its plug-ins through an Automation interface.

The following individual interfaces form this Automation interface:

- `INaturalStudioPlugIn`
- `INaturalStudioPlugInTree`

A plug-in does not need to implement any of the methods in this interface. In fact the simplest working (but of course useless) plug-in just provides the interfaces, but leaves all method implementations empty.

# 5 Natural Studio Interfaces

## Root Interface

Plug-ins access Natural Studio functionality through an Automation interface. All individual interfaces that form the Natural Studio Automation interface can be reached from the root interface, `INatAutoStudio`. A handle to this interface is passed to each plug-in during activation and deactivation.

## Interface Structure

The following tree diagram shows the hierarchical structure of the Automation interface:

```
INatAutoStudio
 |_INatAutoObjects
           |_INatAutoDataAreas
               |_INatAutoDataArea
           |_INatAutoDialogs
               |_INatAutoDialog
           |_INatAutoPrograms
               |_INatAutoProgram
           |_INatAutoGenericDocuments
               |_INatAutoGenericDocument
           |_INatAutoGenericTexts
               |_INatAutoGenericText
           |_INatAutoObjectLists
               |_INatAutoObjectList
           |_INatAutoObjectTrees
               |_INatAutoObjectTree
                       |_INatAutoObjectTreeNode
           |_INatAutoSelectedObjects
               |_INatAutoSelectedObject
           |_INatAutoRefreshObject
 |_INatAutoControlBars
           |_INatAutoImages
           |_INatAutoCommands
               |_INatAutoCommand
           |_INatAutoToolBars
               |_INatAutoToolBar
           |_INatAutoFrameMenus
               |_INatAutoFrameMenu
                       |_INatAutoPopupMenu
           |_INatAutoContextMenus
               |_INatAutoContextMenu
                       |_INatAutoPopupMenu
 |_INatAutoTypes
           |_INatAutoNodeTypes
                       |_INatAutoNodeType
```

```
          |_INatAutoNodeImages
|_INatAutoPlugIns
          |_INatAutoPlugIn
|_INatAutoResultViews
          |_INatAutoResultView
|_INatAutoSystem
          |_INatAutoEnvironments
                    |_INatAutoEnvironment
                              |_INatAutoNatparm
                              |_INatAutoNatsvar
          |_INatAutoApplications
                    |_INatAutoApplication
                              |_INatAutoLinkedApplications
                              |_INatAutoEnvironment
          |_INatAutoSysmain
|_INatAutoProgressIndicator
```

## Working with Control Bars

Natural Studio users access plug-in functionality by using commands. A plug-in identifies each command by a number. The number can be freely chosen, but must of course be unique per plug-in. A command can have a caption and an image assigned. Caption and image represent the command in menus and toolbars.

To provide a command to the user, a plug-in first creates an `INatAutoCommand` interface in the `INatAutoCommands` collection:

```
send "Add" to #commands
with 4711 "MyCommand" #myImage
return #myCommand
```

While creating the command, in the above example the plug-in refers to an image `#myImage`. This image is used to represent the command visually in menus and toolbars. The plug-in may have loaded the image before, using the method `INatAutoImages::LoadImage`:

```
send "LoadImage" to #images
with "e:\images\myimage.bmp"
return #myImage
```

This results in an `IPictureDisp` interface that can be passed to the method `INatAutoCommands::Add`. The `IPictureDisp` interface is a predefined interface in Windows. An `IPictureDisp` interface can be created in Natural using the method `INatAutoImages::LoadImage`.

Alternatively, the plug-in can pass the image file name directly to the method `INatAutoCommands::Add`:

```
send "Add" to #commands
with 4711 "MyCommand" "e:\images\myimage.bmp"
return #myCommand
```

When the user later chooses the command in a menu or toolbar, the plug-in is notified by using the method `INaturalStudioPlugIn::OnCommand`.

But in order to make the command accessible to users in the first place, the plug-in must insert it into a menu or toolbar. We show this with a toolbar as example. Here the plug-in first locates the Tools toolbar. Then it inserts the previously created command into the toolbar.

```
send "Item" to #toolbars
with "Tools"
return #toolsToolbar
send "InsertCommand" to #toolsToolbar
with #myCommand
```

The plug-in needs to create the command only once and can then assign it to different toolbars or menus.

The plug-in might as well create its own toolbar and add the command to this toolbar:

```
send "Add" to #toolbars
with "MyToolbar"
return #myToolbar
send "InsertCommand" to #myToolbar
with #myCommand
```

We saw the plug-in use the interfaces `INatAutoCommands`, `INatAutoImages` and other interfaces. But how does the plug-in get access to these interfaces in the first place? The plug-in accesses them by querying properties of the root interface, `INatAutoStudio`. A handle to this interface is passed to each plug-in during activation and deactivation. From this interface the plug-in can navigate to any other section of the Natural Studio Automation interface.

To work with control bars, a plug-in uses the interfaces described in the following sections:

`INatAutoControlBars`
`INatAutoImages`
`INatAutoCommands`
`INatAutoCommand`
`INatAutoToolBars`
`INatAutoToolBar`
`INatAutoContextMenus`
`INatAutoContextMenu`
`INatAutoFrameMenus`
`INatAutoFrameMenu`
`INatAutoPopupMenu`

# Working with Node Types

Natural Studio frequently uses tree views and list views to display development objects and to navigate through them. Each node in a tree view or list view is characterized by a node type. The node type defines how nodes of a given type are represented in the user interface.

Each node type is identified by an integer number. A development object belonging to a given node type is identified by the number of that node type and by an alphanumeric key. The format of the key varies from node type to node type.

Plug-ins that wish to create their own tree views and list views in Natural Studio can refer to the predefined node types. In addition, plug-ins can define their own node types and can then refer to these user-defined node types.

The following topics are covered below:



## Predefined Node Types

The built-in Natural Studio development objects such as program, dialog, class, library or application have a predefined node type and key format. Many interfaces and methods in the Natural Studio Automation interface refer to the predefined node types. The full list of available predefined node types and the format of their keys is defined in the following tables.

### Predefined Node Types

| Node Type Number | Node Type Name | Key Format |
|---|---|---|
| 1001 | Parameter data area | NATID |
| 1002 | Copycode | NATID |
| 1003 | DDM | NATID |
| 1004 | Global data area | NATID |
| 1005 | Helproutine | NATID |
| 1006 | Local data area | NATID |
| 1007 | Map | NATID |
| 1008 | Subprogram | NATID |
| 1009 | Program | NATID |
| 1010 | Subroutine | NATID |
| 1011 | Text | NATID |
| 1012 | View | NATID |

| Node Type Number | Node Type Name | Key Format |
|---|---|---|
| 1013 | Dialog | NATID |
| 1014 | Class | NATID |
| 1015 | Command processor | NATID |
| 1017 | Mainframe DDM | DDMID |
| 1018 | Function | NATID |
| 1019 | Shared resource | RESID |
| 1020 | Error message file | NATID |
| 1021 | Adapter | NATID |
| 1051 | Parameter data area (in application) | NATID |
| 1052 | Copycode (in application) | NATID |
| 1053 | DDM (in application) | NATID |
| 1054 | Global data area (in application) | NATID |
| 1055 | Helproutine (in application) | NATID |
| 1056 | Local data area (in application) | NATID |
| 1057 | Map (in application) | NATID |
| 1058 | Subprogram (in application) | NATID |
| 1059 | Program (in application) | NATID |
| 1060 | Subroutine (in application) | NATID |
| 1061 | Text (in application) | NATID |
| 1062 | View (in application) | NATID |
| 1063 | Dialog (in application) | NATID |
| 1064 | Class (in application) | NATID |
| 1065 | Command processor (in application) | NATID |
| 1067 | Mainframe DDM (in application) | DDMID |
| 1068 | Function (in application) | NATID |
| 1069 | Shared resource (in application) | RESID |
| 1070 | Error message file (in application) | NATID |
| 1071 | Adapter (in application) | NATID |
| 1101 | System file | FILEID |
| 1102 | Natural system file | FILEID |
| 1103 | User system file | FILEID |
| 1104 | DDM system file | FILEID |
| 1111 | Library | LIBID |
| 1112 | Library (in application) | LIBID |
| 1121 | Environment | BSTR |
| 1131 | Base application | BSTR |

| Node Type Number | Node Type Name | Key Format |
|---|---|---|
| 1132 | Compound application | BSTR |
| 1141 | Application server | BSTR |

### Format NATID

| Syntax | Description |
|---|---|
| *name library fnr dbnr* | Identifies the Natural development object with the given name in the given library in the given system file. The individual parts of the identifier are separated by spaces. |
| *name library* | Identifies the Natural object with the given name in the given library. The system file is then determined from the library according to the usual Natural logic, depending on the library name. The individual parts of the identifier are separated by spaces. |
| *name* | Identifies the Natural object with the given name in the current logon library. |

### Format RESID

| Syntax | Description |
|---|---|
| *name*/*library*/*fnr*/*dbnr* | Identifies the shared resource with the given name in the given library in the given system file. The individual parts of the identifier are separated by slashes. |
| *name*/*library* | Identifies the shared resource with the given name in the given library. The system file is then determined from the library according to the usual Natural logic, depending on the library name. The individual parts of the identifier are separated by slashes. |
| *name* | Identifies the shared resource with the given name in the current logon library. |

### Format DDMID

| Syntax | Description |
|---|---|
| *name fnr dbnr* | Identifies the mainframe DDM with the given name in the given `FDIC` system file. The individual parts of the identifier are separated by spaces. |
| *name* | Identifies the mainframe DDM with the given name in the current `FDIC` system file. |

### Format LIBID

| Syntax | Description |
|---|---|
| *name fnr dbnr* | Identifies the Natural library with the given name in the given system file. The individual parts of the identifier are separated by spaces. |
| *name* | Identifies the Natural library with the given name. The system file is then determined from the library according to the usual Natural logic, depending on the library name. |

### Format FILEID

| Syntax | Description |
|---|---|
| *fnr dbnr* | Identifies the Natural system file with the given numbers. The individual parts of the identifier are separated by spaces. |

### Example

Assume that in a certain development environment, we have a system file with the database number "101" and the file number "99", containing a library `MYLIB` with a program `MYPROG`.

In the given environment

- the program is identified by the node type 1009 and the key "MYPROG MYLIB 99 101",
- the library is identified by the node type 1111 and the key "MYLIB 99 101",
- the system file is identified by the node type 1103 and the key "99 101".

## User-defined Node Types

Plug-ins can define their own node types. This is useful if a plug-in wants to display tree views or list views of development objects not belonging to the predefined set of Natural Studio objects. An example is the Object Description plug-in. It is also useful for plug-ins that want to display certain aspects of Natural Studio objects not covered by built-in Natural Studio functionality. An example is the XRef Evaluation plug-in.

When defining its own node type, a plug-in is free to choose an arbitrary positive integer value starting with "20000". Values below "20000" are reserved for predefined node types. It does not matter if different plug-ins chose the same integer value for a node type. Internally, Natural Studio distinguishes the node types by their numbers and by the plug-in that defined the node type.

The plug-in is free to define the key format for each user-defined node type. Natural Studio does not interpret the keys of user-defined node types, but treats them as opaque strings.

To define new node types and their visual representations, a plug-in uses the interfaces described in the following sections:

`INatAutoTypes`
`INatAutoNodeImages`
`INatAutoNodeTypes`
`INatAutoNodeType`

## Working with Selections

Through the interfaces described in this section, plug-ins can access the set of objects the user has currently selected in Natural Studio. A plug-in might need this information to decide if a specific menu or toolbar command is applicable to the current selection and must hence be enabled in the user interface. If the user then executes the command, the plug-in again needs to know the set of selected objects in order to apply the command to each of them. A plug-in has access to the current selection through the interfaces described in this section.

In order to work with the current selection, a plug-in starts with the root interface `INatAutoStudio`, retrieves the `INatAutoObjects` interface and then the `INatAutoSelectedObjects` interface. We assume here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`.

```
#objs := #studio.Objects
#selobjs := #objs.SelectedObjects
```

The returned `INatAutoSelectedObjects` interface gives access to the set of objects the user has currently selected. Using this interface, the plug-in can, for instance, iterate across the selected objects and inspect them. The method `Item` returns an `INatAutoSelectedObject` interface to a specific selected object.

```
#iCount := #selobjs.Count
for #i := 1 to #iCount
  send "Item" to #selobjs
  with #i return #selobj
  #iType := #selobj.Type
  #aKey := #selobj.Key
end-for
```

The property `FocusObject` returns the index of the specific selected object that currently has the focus. This index can be used to retrieve the `INatAutoSelectedObject` interface of the focus object.

```
#iFocus := #selobjs.FocusObject
send "Item" to #selobjs
with #iFocus return #focus
```

The method `ContainsObjectType` can be used for a quick check if the current selected set contains objects of a specific type. This might be sometimes sufficient to decide if a specific command shall be enabled or not.

```
send "ContainsObjectType" to #selobjs
with 1009 return #bContainsPrograms
```

For specific checks the plug-in can also retrieve and process the current selection as an XML document.

```
#aSelectedObjectsXML := #selobjs.SelectedObjects
```

To work with selections, a plug-in uses the interfaces described in the following sections:

INatAutoObjects
INatAutoSelectedObjects
INatAutoSelectedObject

## Working with Natural Development Objects

Through the interfaces described in this section, plug-ins can create and edit Natural development objects. Being able to create new development objects, load existing objects into an editor, manipulate their contents and to save and stow them, enables plug-ins to provide generation functions and thus to help automating the development process. An example is the Program Generation plug-in.

To open a program in the program editor, for instance, a plug-in starts with the root interface INatAutoStudio, retrieves the INatAutoObjects interface and then the INatAutoPrograms interface. Now it uses the method INatAutoPrograms::Open to load the program into the editor.

We assume here that the plug-in has kept a handle to the interface INatAutoStudio in a variable named #studio. This handle was passed to the plug-in during activation, in the method INaturalStudioPlugIn::OnActivate.

```
#objects := #studio.Objects
#programs := #objects.Programs
send "Open" to #programs
with 1009 "MYPGM" "MYLIB" return #program
```

The resulting INatAutoProgram interface can now be used to operate on the program, for instance, to stow it and then to close the editor.

```
send "Stow" to #program
send "Close" to #program
```

A new program source is created by using the method `Add`.

```
send "Add" to #programs
with 1009 return #program
```

Source code is added to the program either as a whole by using the property `Source`:

```
#program.Source := "WRITE ""HELLO, WORLD!"" END
```

Or incrementally by using the method `InsertLines`.

```
send "InsertLines" to #program
with "WRITE ""HELLO, WORLD!" #return #next
send "InsertLines" to #program
with "END" #next return #next
```

The interface `INatAutoProgram` provides also search and replace methods and other methods to modify the source code.

```
send "Search" to #program
with "HELLO" return #found
send "ReplaceLines" to #program
with "WRITE ""Good morning" #found
```

Dialogs and data areas are accessed in a similar way by using the interfaces `INatAutoDialog` and `INatAutoDataArea`. But there is one particularity with these objects: Even though there is a graphical or structured editor in the user interface for these objects, they are edited textually through the Automation interface. Applied to data areas this means: If a plug-in wants to generate a data area, it actually has to generate a `DEFINE DATA` statement.

```
#objects := #studio.Objects
#dataareas := #objects.DataAreas
send "Add" to #dataareas
with 1006 return #lda
*
send "StartEdit" to #lda
send "InsertLines" to #lda
with "DEFINE DATA LOCAL" return #next
send "InsertLines" to #lda
with "1 MYSTRING(A10)" #next return #next
send "InsertLines" to #lda
with "1 MYNUMBER(I4)" #next return #next
send "InsertLines" to #lda
with "END-DEFINE" #next return #next
send "EndEdit" to #lda
*
```

```
send "Stow" to #lda
send "Close" to #lda
```

The calls to the methods `INatAutoDataArea::StartEdit` and `INatAutoDataArea::EndEdit` are used to mark the beginning and end of a series of editing operations.

To work with Natural development objects, a plug-in uses the interfaces described in the following sections:

`INatAutoObjects`
`INatAutoPrograms`
`INatAutoProgram`
`INatAutoDialogs`
`INatAutoDialog`
`INatAutoDataAreas`
`INatAutoDataArea`

## Working with Generic Text Documents

Through the interfaces described in this section, plug-ins can use the Natural Studio program editor as editor for arbitrary text objects. A plug-in can open a program editor session, pass a buffer with text data to it, let the user edit the data and then retrieve the modified data back. The plug-in itself is responsible for providing and storing the data to be edited. The program editor provides the usual editing functions, as far as they are appropriate for generic text objects.

To let the user edit a given text in the program editor, a plug-in starts with the root interface `INatAutoStudio`, retrieves the `INatAutoObjects` interface and then the `INatAutoGenericTexts` interface. Now it uses the method `INatAutoGenericTexts::Open` to load the text buffer into the editor.

We assume here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`. Also we assume that the text to be edited is contained in the alphanumeric variable `#buffer`.

```
#objects := #studio.Objects
#texts := #objects.GenericTexts
send "Open" to #texts
with "Curriculum Vitae" "Dana Scully" #buffer
return #text
```

The editor is then opened and the user can edit the text interactively.

The plug-in can use the resulting `INatAutoGenericText` interface to operate on the text, for instance, to insert lines:

```
send "InsertLines" to #text
with "Taught for two years at Quantico Medical School"
```

If the user chooses the **Save** button in the editor, the plug-in receives the notification `PLUGIN-NOTIFY-SAVE`. In response to this notification, it will usually retrieve the edited text from the editor and save it.

```
#buffer := #text.Source
* Now save the text in a plug-in specific way.
```

To work with generic text documents, a plug-in uses the interfaces described in the following sections:

`INatAutoGenericTexts`
`INatAutoGenericText`
`INatAutoObjects`

## Working with Generic Documents

A plug-in that maintains own development objects might want to provide its own editors for each of its development object types. Editors in Natural Studio typically maintain development objects in MDI (Multiple Document Interface) windows. In the following, we call them document windows. Natural Studio has a number of built-in editors, for instance, the program editor and the dialog editor. A plug-in can implement its own editor with a so-called generic document window.

Implementing such an editor as a generic document window makes the editor behave like the built-in editors in Natural Studio. Essentially this means: Several editor windows on different objects can be opened in parallel and the user can switch between them.

In order to implement a generic document window, you first create a Natural dialog of type "Plug-in MDI window". In your plug-in code, you can then open this dialog with the `OPEN DIALOG` statement and let Natural Studio display the dialog as a document window. Normally you will do this in the command handler of your plug-in, that is in the method `INaturalStudioPlugIn::OnCommand`:

```
open dialog "mydlg" null-handle giving #dialogid
#objects := #studio.Objects
#genericdocs := #objects.GenericDocuments
send "Add" to #genericdocs with #dialogid return #doc
```

The resulting `INatAutoGenericDocument` interface can now be used to operate on the document window.

The plug-in has several other means to communicate with the Natural dialog contained in the generic document window:

- The plug-in can send events to the dialog with the `SEND EVENT` statement and using the dialog ID.
- The dialog can send method calls to the plug-in. To achieve this, the plug-in should pass its own `*THIS-OBJECT` handle to the dialog in the `OPEN DIALOG` statement.
- The dialog can call the Natural Studio Automation interface. To achieve this, the plug-in should pass the `INatAutoStudio` interface pointer to the dialog in the `OPEN DIALOG` statement.

Whenever the user activates a document window, Natural Studio automatically switches the frame menu to a menu that contains the commands applicable to the active document. In the case of built-in document windows, these frame menus are predefined. In the case of a generic document window, the plug-in itself can provide an appropriate frame menu.

The plug-in can create a frame menu of its own by cloning an existing frame menu using the method `INatAutoFrameMenus::Clone` and adding new commands to the clone as necessary using the method `INatAutoFrameMenu::InsertCommand`.

Afterwards it passes the resulting `INatAutoFrameMenu` interface to Natural Studio when calling the method `INatAutoGenericDocuments::Add`.

To work with generic documents, a plug-in uses the interfaces described in the following sections:

`INatAutoGenericDocuments`
`INatAutoGenericDocument`
`INatAutoObjects`

## Working with Tree Views and List Views

Through the interfaces described in this section, a plug-in can display its own tree views and list views in Natural Studio. Tree views and list views are frequently used in Natural Studio to display development objects and to navigate through them.

In order to display objects in tree views and list views, the plug-in must first register the types of the tree or list view nodes that it is going to display. This procedure is described in ***Working with Node Types***.

A plug-in that displays objects in tree views and list views must also implement the methods of the interface `INaturalStudioPlugInTree` appropriately. Natural Studio calls the methods of this interface when expanding or refreshing the tree.

In order to open a tree view, the plug-in starts with the root interface `INatAutoStudio`, retrieves the `INatAutoObjects` interface and then the `INatAutoObjectTrees` interface. We assume here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`.

```
#objs := #studio.Objects
#trees := #objs.ObjectTrees
```

The resulting `INatAutoObjectTrees` interface gives access to the currently open tree view document windows. Through this interface the plug-in can open a new tree view with a given root object.

```
send "Open" to #trees
with #type #key #caption
return #tree
```

The node type specified in the method `Open` must have been registered before, as described in ***Working with Node Types***. The `INatAutoObjectTree` interface returned from the method `Open` gives access to the tree view document window just opened. Through this interface the plug-in can, for instance, later close the document window.

```
send "Close" to #tree
```

When opening a tree view, the plug-in specifies at least the type and key of the root object and a caption to be displayed on the tree view document window. Natural Studio will retrieve additional information needed to expand the tree view by using the interface `INaturalStudioPlugInTree` that must be implemented by the plug-in.

The nodes of a tree view can be accessed through the interface `INatAutoObjectTreeNode`. The root node of a tree view is retrieved with the method `INatAutoObjectTree::GetRootNode`, which returns an interface `INatAutoObjectTreeNode`. This interface can then be used, for instance, to expand the node and to access the child nodes. In the same way, the currently selected node of a tree view can be retrieved.

```
send "GetRootNode" to #tree
return #rootnode
send "Expand" to #rootnode
send "GetChild" to #rootnode
return #firstchildnode
send "GetNext" to #firstchildnode
return #nextchildnode
send "Expand" to #nextchildnode
send "GetSelectedNode" to #tree
return #selectednode
send "Expand" to #selectednode
```

The interface `INatAutoObjectTreeNode` controls only the visual appearance of an individual tree view, not the underlying object structure, which is possibly represented differently in several views at a time. The object structure itself is under the control of the plug-in that defines and provides it through its `INaturalStudioPlugInTree` interface.

List view document windows are created in a similar way as tree view document windows, except that the interface `INatAutoObjectLists` is used instead of `INatAutoObjectTrees`.

To work with tree views and list views, a plug-in uses the interfaces described in the following sections:

`INatAutoObjects`
`INatAutoObjectTrees`
`INatAutoObjectTree`
`INatAutoObjectTreeNode`
`INatAutoObjectLists`
`INatAutoObjectList`
`INatAutoRefreshObject`

## Working with Result Views

Through the interfaces described in this section, a plug-in can display the results of its work in the Natural Studio result view. Objects displayed in a result view can be target of commands and can be used as starting point for navigation. Examples of built-in functions that use result views are the **Cat All** command and the **Find** command.

In order to display objects in result views, the plug-in must first register the types of nodes that it is going to display. This procedure is described in the section *Working with Node Types*.

In order to work with result views, the plug-in starts with the root interface `INatAutoStudio` and retrieves the `INatAutoResultViews` interface. We assume here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`.

```
#resultviews := #studio.ResultViews
```

The resulting `INatAutoResultViews` interface gives access to the result view control bar and the currently open result views. The plug-in can use this interface, for instance, to show or hide the result view control bar.

```
send "Show" to #resultviews
```

Through the interface `INatAutoResultViews` the plug-in can open a new result view.

```
send "Open" to #resultviews
with #caption #image #headers
return #resultview
```

When opening a result view, the plug-in specifies a caption and an image to be displayed on the result view tab and (if needed) column headers for the result view.

The INatAutoResultView interface returned from the method Open gives access to the result view just opened. Through this interface the plug-in can activate the result view, insert rows into it and update the display. The method SetVisible scrolls a specific row into view.

```
#resultview.Active := true
send "InsertRows" to #resultview
with #rows return #last
send "Update" to #resultview
send "SetVisible" to #resultview
with #last
```

Finally the plug-in can close its result view.

```
send "Close" to #resultview
```

To work with result views, a plug-in uses the interfaces described in the following sections:

INatAutoResultViews
INatAutoResultView

## Working with Environments

Through the interfaces described in this section, plug-ins can inspect the available local and remote development environments, map environments, connect to and disconnect from a remote environment and activate an environment.

In order to work with environments, a plug-in starts with the root interface INatAutoStudio, retrieves the INatAutoSystem interface and then the INatAutoEnvironments interface. We assume here that the plug-in has kept a handle to the interface INatAutoStudio in a variable named #studio. This handle was passed to the plug-in during activation, in the method INaturalStudioPlugIn::OnActivate.

```
#system := #studio.System
#envs := #system.Environments
```

The returned INatAutoEnvironments interface gives access to the local environment and all remote environments that have once been connected during the current Natural Studio session.

Through this interface the plug-in can, for instance, map a new remote environment, specifying host name, port number, user ID, password and other arguments.

```
send "Add" to #envs
with "IBM2" "4712" "SCULLY" "secret" "STACK=(LOGON XFILES)"
return #env
```

The returned interface `INatAutoEnvironment` gives access to attributes of the environment.

```
#bIsActive := #env.Active
#bIsConnected := #env.Connected
```

The property `Parameters` gives access to the interface `INatAutoNatparm`. This interface contains properties that represent the Natural parameters under which the environment is running. Only a subset of the Natural parameters is available through this interface.

```
#natparm := #env.Parameters
#fuserDBnr := #natparm.FuserDBnr
#fuserFnr := #natparm.FuserFnr
```

The property `SystemVariables` gives access to the interface `INatAutoNatsvar`. This interface contains properties that represent the system variables currently set in the environment. Only a subset of the system variables is available through this interface.

```
#natsvar := #env.SystemVariables
#language := #natsvar.Language
```

The plug-in uses the method `Disconnect` to disconnect from the remote environment.

```
send "Disconnect" to #env
```

To work with environments, a plug-in uses the interfaces described in the following sections:

`INatAutoEnvironments`
`INatAutoEnvironment`
`INatAutoSystem`
`INatAutoNatparm`
`INatAutoNatsvar`

## Working with Applications

Through the interfaces described in this section, plug-ins can inspect the applications available on the application server, map applications into the Natural Studio session, connect to and disconnect from an application, activate an application and create and modify applications. An example of a plug-in that uses this section of the interface is the Application Wizard.

In order to work with applications, the plug-in starts with the root interface `INatAutoStudio`, retrieves the `INatAutoSystem` interface and then the `INatAutoApplications` interface. We assume

here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`.

```
#system := #studio.System
#apps := #system.Applications
```

The resulting `INatAutoApplications` interface gives access to the currently active application server and the applications it contains. Through this interface the plug-in can, for instance, ask for the currently active application.

```
#app := #apps.ActiveApplication
```

The plug-in can also create a new application and map it into the Natural Studio session.

```
send "Add" to #apps
with "MYAPPLICATION" return #app
```

The resulting `INatAutoApplication` interface gives access to attributes of the application.

```
#bIsActive := #app.Active
#bIsConnected := #app.Connected
```

For a compound application, the property `LinkedApplications` returns the interface `INatAutoLinkedApplications`. This interface allows accessing the base applications that are linked to the compound application.

```
#linkedapps:= #app.LinkedApplications
#iCount := #linkedapps.Count
```

For a base application, the property `LinkedObjects` returns an XML document containing the list of objects linked to the application.

```
#aObjects:= #app.LinkedObjects
```

The plug-in can also link and unlink objects to and from the application.

```
send "UnlinkObject" to #app
with 1009 "OLDPROG" "MYLIB"
send "LinkObject" to #app
with 1009 "NEWPROG" "MYLIB"
```

Finally the plug-in can disconnect and unmap the application.

```
send "Disconnect" to #app
send "Unmap" to #app
```

To work with applications, a plug-in uses the interfaces described in the following sections:

`INatAutoApplications`
`INatAutoApplication`
`INatAutoLinkedApplications`

## Working with Plug-ins

Through the interfaces described in this section, a plug-in can inspect the currently installed plug-ins, read their properties and activate or deactivate a plug-in. This includes the possibility that a plug-in deactivates itself. An example for a plug-in that uses this section of the interface is the Plug-in Manager.

In order to work with plug-ins, the plug-in starts with the root interface `INatAutoStudio` and retrieves the interface `INatAutoPlugIns`. We assume here that the plug-in has kept a handle to the interface `INatAutoStudio` in a variable named `#studio`. This handle was passed to the plug-in during activation, in the method `INaturalStudioPlugIn::OnActivate`.

```
#plugins := #studio.PlugIns
```

The resulting `INatAutoPlugIns` interface gives access to the currently installed plug-ins. Using this interface, the plug-in can, for instance, iterate across the installed plug-ins and inspect their attributes. The method `Item` returns an `INatAutoPlugIn` interface to a specific plug-in.

```
#iCount := #plugins.Count
for #i := 1 to #iCount
  send "Item" to #plugins
  with #i return #plugin
  #aName := #plugin.Name
  #aProgID := #plugin.ProgID
  #bIsActive := #plugin.Active
end-for
```

Through the interface `INatAutoPlugIn` the plug-in can also activate or deactivate a specific plug-in by modifying the property `Active`. The following sample toggles the activation state of a plug-in.

```
#bIsActive := #plugin.Active
if #bIsActive
  #plugin.Active := false
else
  #plugin.Active := true
end-if
```

The interface INatAutoPlugIn can be used to send a command to the plug-in. The following sample checks whether the plug-in command with the ID "200" is currently enabled and if so, lets the plug-in execute the command. Of course this requires that we know that the plug-in implements a command with the ID "200" and what this command does.

```
#bEnabled = false
#bChecked := false
send "OnCommandStatus" to #plugin
with 200 #bEnabled #bChecked
if #bEnabled
  send "OnCommand" to #plugin with 200
end-if
```

Through the interface INatAutoPlugIn, the plug-in can get access to arbitrary services that another plug-in provides with a so-called custom interface. The following sample retrieves the custom interface of a plug-in and calls one of its services. Of course this requires that the plug-in has documented the services it provides with its custom interface.

```
#icustom := null-handle
send "GetCustomInterface"
to #plugin return #icustom
if #icustom ne null-handle
  #result := 0
  send "GetMaritalStatus" to #icustom
  with "Anderson, Gillian" return #result
end-if
```

In order to provide a custom interface, a plug-in must implement an additional interface beside the two predefined interfaces INaturalStudioPlugIn and INaturalStudioPlugInTree and make this interface the default dispatch interface. For a plug-in implemented in Natural this means placing this interface at the first position in the DEFINE CLASS statement.

```
define class ...
  object using ...
  id "..."
  interface icustom
    id "..."
    method GetMaritalStatus id 1 is gstat-n
      parameter using gstat-a
    end-method
  end-interface
  interface using nstplg-i
```

```
  interface using nstplt-i
end-class
end
```

To work with plug-ins, a plug-in uses the interfaces described in the following sections:

`INatAutoPlugIns`
`INatAutoPlugIn`

## Working with Dialogs

If a plug-in wants to open dialog boxes in Natural Studio, some special considerations have to be taken. The support of dialog boxes in plug-ins depends mainly on the condition if the plug-in is running in as an in-process ActiveX component or if it is running in a separate process.

The following topics are covered below:

- Plug-ins Running in a Separate Process
- Plug-ins Running In-process

### Plug-ins Running in a Separate Process

In general, an ActiveX component running in a separate process cannot open a dialog box in the client process. This restriction of the Windows system itself is overcome in the special case that a plug-in is written in Natural.

If a plug-in is written in Natural, it can open modal dialog boxes in Natural Studio. Precisely this means: Natural dialogs that are defined in the dialog editor with the **Type** attribute set to "Standard window" and the **Style** attribute set to "Dialog box". Other styles of the dialog type "Standard window" cannot be used in plug-ins.

To open a dialog box, a plug-in uses the usual `OPEN DIALOG` statement.

### Plug-ins Running In-process

If a plug-in is implemented as an in-process ActiveX component (this means: as a DLL), it can open modal and non-modal dialogs in Natural Studio. To open a dialog, the plug-in uses the statements usual in the programming language it is written in. Plug-ins written in Natural always run in a separate process, so this applies only to plug-ins written in programming languages that support implementing in-process ActiveX components.

For details on how to implement plug-ins in programming languages other than Natural, see *Developing Plug-ins in Other Programming Languages*.

# 6 Developing Plug-ins

## 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

**Note:** Remote debugging is no longer supported. Instead, you will have to use the debugger of NaturalONE. Using the NaturalONE debugger, it is possible, for example, to debug Natural RPC applications or to debug workplace applications which have been created with Natural for Ajax.

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.

**Note:** Remote debugging is no longer supported. Instead, you will have to use the debugger of NaturalONE. Using the NaturalONE debugger, it is possible, for example, to debug

Natural RPC applications or to debug workplace applications which have been created with Natural for Ajax.

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

**Note:** Remote debugging is no longer supported. Instead, you will have to use the debugger of NaturalONE. Using the NaturalONE debugger, it is possible, for example, to debug Natural RPC applications or to debug workplace applications which have been created with Natural for Ajax.

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.

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.

# 7 Plug-in Example

The Natural Studio plug-in example demonstrates how the Natural Studio metastructure can be extended with plug-ins that define your own object types, assign commands to them and display objects as nodes in tree views and list views.

The plug-in example shows information about the Natural application programming interfaces contained in the library SYSEXT. It allows users to list their documentation and to execute a test program for each of the application programming interfaces.

The source code of the plug-in example is delivered in the library SYSEXPLG. It is intended to give an impression of how plug-ins can be implemented with Natural Studio.

## Activating the Plug-in Example

The plug-in example is installed automatically during Natural Studio installation. Initially, the activation of plug-ins is disabled. Therefore, in order to use the plug-in example, you must first enable plug-in activation and then activate the plug-in example.

### To activate the plug-in example

1. Make sure that plug-in activation has been enabled. See *Workspace Options* in the documentation *Using Natural Studio*.
2. Invoke the Plug-in Manager as described in *Invoking the Plug-in Manager* in the documentation *Using Natural Studio*.
3. In the **Plug-in Manager** window, select **Plug-in Example**.
4. Activate the plug-in example as described in *Activating and Deactivating a Plug-in* in the documentation *Using Natural Studio*.

## Using the Plug-in Example

When the plug-in example has been activated in the Plug-in Manager, the following additional elements are available in the Natural Studio window.

### Menu Commands

The **Tools** menu provides the cascading menu **Plug-in Example** with the following commands:

| Command | Description |
|---|---|
| **Open Tree View** | If an application programming interface (subprogram USR*nnnn*N), its description (text member USR*nnnn*T) or its test program (program USR*nnnn*P) is selected in library SYSEXT, this command displays information about this application programming interface in a tree view window. If none of the above is selected, this command displays information about all application programming interfaces in a tree view window. |
| **Open List View** | Displays the same information as above in a list view window. |

### Context Menus

The cascading menu **Plug-in Example** with the above commands is available in the context menus of the Natural object types program, subprogram and text.

### Toolbar

An additional toolbar is shown. The toolbar buttons represent the following menu commands:

 **Open Tree View**

 **Open List View**

# II Interface Reference

This part provides descriptions of all interfaces (plug-in interfaces and Natural Studio interfaces) in alphabetical order. The following interfaces are available:

# 8 INatAutoApplication

## Purpose

An application available on the current application server. Applications and the application server are only available with Natural Single Point of Development. See also *Remote Development Using SPoD*.

## Properties

The following properties are available:

- Parent
- Studio
- BaseApplication
- MainframeApplication
- Mapped
- Connected
- Active
- Name
- Description
- Host
- Port
- Profile
- ProfileDBnr
- ProfileFnr
- UserId
- MainLibrary
- HasLinkedObjects
- LinkedObjects
- LinkedEntries
- Environment

- LinkedApplications

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplications**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### BaseApplication

TRUE if this is a base application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### MainframeApplication

TRUE if this is a base application on a mainframe platform. FALSE if this is a base application on an Open Systems platform or a compound application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Mapped

TRUE if this application is mapped into the application workspace.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Connected

TRUE if

- this is a base application, and
- the application is mapped into the application workspace, and
- there is a connection to a server session.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Active

TRUE if

- this is a base application, and
- the application is mapped into the application workspace, and
- there is a connection to a server session, and
- the application is the active one.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Name

Name of the application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Description

The description of the application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Host

The host name of the development server. Returns an empty string for a compound application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Port

The port number of the development server. Returns 0 for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Profile

The profile (mainframe) or `NATPARM` parameter file (Open Systems) under which the development server is running. Returns an empty string for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### ProfileDBnr

The profile database number of the development server. Returns 0 for compound applications and for base applications running on Open Systems platforms.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### ProfileFnr

The profile file number of the development server. Returns 0 for compound applications and for base applications running on Open Systems platforms.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### UserId

The user ID under which a base applicaton is mapped. Returns an empty string for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### MainLibrary

The main library of the application. Returns an empty string for compound applications and for base applications for which no main library has been defined.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### HasLinkedObjects

TRUE if a base application has linked objects. Always FALSE for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### LinkedObjects

Returns the list of objects linked to a base application, formatted as an XML document according to the DTD shown below. Returns an empty document for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

#### Document Type Description

```
<?xml version="1.0"?>
<!ELEMENT aobjects (ccount, aobject*)>
<!ELEMENT ccount (#PCDATA)>
<!ELEMENT aobject (atype, akey)>
<!ELEMENT atype (#PCDATA)>
<!ELEMENT akey (#PCDATA)>
```

| Element | Meaning |
|---|---|
| ccount | The number of objects in the list. |
| atype | The type of the object. This must be one of the predefined development object types that is allowed to be used as entry object of an application. |
| akey | The key that identifies the object within its type. |

### LinkedEntries

Returns the list of entry objects linked to a base application, formatted as an XML document according to the DTD shown below. Returns an empty document for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

#### Document Type Description

```
<?xml version="1.0"?>
<!ELEMENT aobjects (ccount, aobject*)>
<!ELEMENT ccount (#PCDATA)>
<!ELEMENT aobject (atype, akey)>
<!ELEMENT atype (#PCDATA)>
<!ELEMENT akey (#PCDATA)>
```

| Element | Meaning |
|---|---|
| ccount | The number of objects in the list. |
| atype | The type of the object. This must be one of the predefined development object types that is allowed to be used as entry object of an application. |
| akey | The key that identifies the object within its type. |

### Environment

Returns the Natural environment of a base application. Returns NULL-HANDLE for compound applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### LinkedApplications

Returns the collection of applications linked to a compound application. Returns NULL-HANDLE for base applications.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following methods are available:

- Map
- Unmap
- Connect
- Disconnect
- Activate
- Remove
- LinkObject
- UnlinkObject
- LinkEntry
- UnlinkEntry

### Map

Maps an application into the application workspace.

#### Parameters:

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| UserID | A | VT_BSTR | Optional |
| Password | A | VT_BSTR | Optional |
| ProfilePassword | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |
| IgnoreWarnings | L | VT_BOOL | Optional |

**UserID**

The user ID under which the application will be mapped.This parameter is ignored for compound applications

**Password**
The password of the user ID under which the application will be mapped. This parameter is ignored for compound applications.

**ProfilePassword**
The password for the profile which has been defined for the application. This parameter is ignored for compound applications.

**Quiet**
If set to FALSE or not specified, the **Map Application** dialog is shown if the session cannot be started with the given parameters.The dialog is then preset with the given parameters.

If set to TRUE, the **Map Application** dialog is not shown.

**IgnoreWarnings**
If set to FALSE or not specified, warnings that occur during mapping are treated like errors.

If set to TRUE, warnings are ignored.

### Unmap

Unmaps the application.

If this application was the active one, the previously active application gets activated.

### Connect

Connects an application to a development server session.

This method is not applicable to compound applications or base applications that are already connected.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| UserID | A | VT_BSTR | Optional |
| Password | A | VT_BSTR | Optional |
| ProfilePassword | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |
| IgnoreWarnings | L | VT_BOOL | Optional |

**UserID**
The user ID under which the application will be connected.

**Password**
The password of the user ID under which the application will be connected.

**ProfilePassword**
The password of the profile which is defined for the application.

**Quiet**
If set to FALSE or not specified, the **Map Application** dialog is shown if the session cannot be started with the given parameters. The dialog is then preset with the given parameters.

If set to TRUE, the **Map Application** dialog is not shown.

**IgnoreWarnings**
If set to FALSE or not specified, warnings that occur during connecting are treated like errors.

If set to TRUE, warnings are ignored.

### Disconnect

Disconnects the application (closes the development server session).

If this application was the active one, the previously active application gets activated.

This method is not applicable to compound applications or base application that are already disconnected.

### Activate

Makes this application the active one.

An application cannot explicitly be deactivated. An application is implicitly deactivated when another application is activated.

This method is not applicable to compound applications or base application that are disconnected or not mapped.

### Remove

Removes the application from the collection (effectively deletes it from the application server).

If this application was the active one, the previously active application gets activated.

### LinkObject

Links the specified development object to the application. Applicable only to base applications.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | |
| Object | A | VT_BSTR | |
| Library | A | VT_BSTR | Optional |

**Type**
The type numbers used here correspond to the type numbers described in the section *Predefined Node Types*.

**Object**
The name of the object.

**Library**
The library containing the object. This parameter is not applicable to DDMs.

### UnlinkObject

Unlinks the specified object from the application. Applicable only to base applications.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | |
| Object | A | VT_BSTR | |
| Library | A | VT_BSTR | Optional |

**Type**
The type numbers used here correspond to the type numbers described in the section *Predefined Node Types*.

**Object**
The name of the object.

**Library**
The library containing the object. This parameter is not applicable to DDMs.

### LinkEntry

Links the specified entry point to the application. Applicable only to base applications.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | |
| Object | A | VT_BSTR | |
| Library | A | VT_BSTR | |

**Type**
The type numbers used here correspond to the type numbers described in the section *Predefined Node Types*.

**Object**
The name of the entry point object.

**Library**
The library containing the entry point object.

### UnlinkEntry

Unlinks the specified entry point object from the application. Applicable only to base applications.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | |
| Object | A | VT_BSTR | |
| Library | A | VT_BSTR | |

**Type**
The type numbers used here correspond to the type numbers described in the section *Predefined Node Types*.

**Object**
The name of the entry point object.

**Library**
The library containing the entry point object.

# 9 INatAutoApplications

## Purpose

Collection of applications available on the current application server. Applications and the application server are only available with Natural Single Point of Development. See also *Remote Development Using SPoD*.

## Properties

The following properties are available:

- Parent
- Studio
- Count
- ActiveApplication
- AppServerEnvironment

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSystem**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of applications in the collection.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### ActiveApplication

Returns the currently active application. Returns NULL-HANDLE if no application is active.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | Get only |

### AppServerEnvironment

Returns the application server environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

## Methods

The following methods are available:

- Item
- Add

### Item

Returns a specific application from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The application identified by the value specified in Index.

**Index**
Identifies a specific application in the collection. This can be either the index of the application in the collection (a value between 1 and Count) or the name of the application.

### Add

Creates a new application, adds it to the collection, maps it, connects it and activates it. Returns the application.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | Optional |
| Name | A | VT_BSTR | |
| BaseApplication | L | VT_BOOL | Optional |
| Host | A | VT_BSTR | Optional |
| Port | I4 | VT_I4 | Optional |
| MainframeApplication | L | VT_BOOL | Optional |
| Profile | A | VT_BSTR | Optional |
| ProfileDBnr | I4 | VT_I4 | Optional |
| ProfileFnr | I4 | VT_I4 | Optional |
| ProfilePassword | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |
| IgnoreWarnings | L | VT_BOOL | Optional |

**Return value**
The newly added application.

**Name**
The name of the application.

**BaseApplication**
If set to TRUE, a base application is created. Otherwise a compound application is created. Creating a base application requires at least the specification of Host and Port.

**Host**
The host name of the development server. Must be specified for base applications.

**Port**
The port number of the development server. Must be specified for base applications.

**MainframeApplication**
If set to TRUE, a mainframe application is created.

**Profile**
The profile under which the development server is running.

**ProfileDBnr**
The profile database number of the development server.

**ProfileFnr**
The profile file number of the development server.

**ProfilePassword**
The profile password of the development server.

**Quiet**
If set to FALSE or not specified, the **Map Application** dialog is shown and is preset with the given parameters.

If set to TRUE, the **Map Application** dialog is not shown.

**IgnoreWarnings**
If set to FALSE or not specified, warnings that occur during mapping are treated like errors.

If set to TRUE, warnings are ignored.

# 10 INatAutoCommand

## Purpose

A command defined by a plug-in.

## Properties

The following properties are available:

- Parent
- Studio
- Caption
- ImageID
- CommandID
- Enabled
- Checked

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommands**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Caption

A string used to identify the command in menus and toolbars.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### ImageID

Index of the image that represents the command visually. This index can be used in the method `INatAutoCommands::Add` to specify an image for a new command.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### CommandID

Numeric ID of the command. When the user later selects the command in the user interface, this ID is passed to the plug-in in the method `INaturalStudioPlugIn::OnCommand`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Enabled

Indicates if the command shall be enabled or disabled, or if Natural Studio shall ask the plug-in for the status of the command on a regular basis through the method `OnCommandStatus`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I2 | VT_I2 | |

#### Values

| | |
|---|---|
| 0 | Natural Studio asks the plug-in for the enabled status of the command through the method `OnCommandStatus`. This is the default. |
| -1 | The command is disabled. |
| 1 | The command is enabled. |

### Checked

Indicates if the command shall have a check mark or not, or if Natural Studio shall ask the plug-in for the checked status of the command on a regular basis through the method `OnCommandStatus`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I2 | VT_I2 | |

#### Values

| | |
|---|---|
| 0 | Natural Studio asks the plug-in for the checked status of the command through the method `OnCommandStatus`. This is the default. |
| -1 | The command has no check mark. |
| 1 | The command has a check mark. |

# 11 INatAutoCommands

## Purpose

The collection of commands defined by plug-ins.

## Properties

The following properties are available:

- Parent
- Studio
- System

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### System

The number of commands that were registered by plug-ins.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_DISPATCH (**INatAutoSystem**) | Get only |

## Methods

The following methods are available:

- Add
- Item

### Add

Creates a new command and adds it to the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| CommandID | I4 | VT_I4 | |
| Caption | A | VT_BSTR | |
| Image | A<br>HANDLE OF OBJECT<br>I4 | VT_BSTR<br>VT_DISPATCH (IPictureDisp)<br>VT_I4 | Optional |

**Return value**
The newly added command.

**CommandID**
Numeric ID of the new command. The plug-in can choose any positive integer value. When the user later selects the command in the user interface, this ID is passed to the plug-in in the method `INaturalStudioPlugIn::OnCommand`.

**Caption**
A string used to identify the command in menus and toolbars.

**Image**
An image used to represent the command visually. The image must be a 16 color, 16x16 bitmap, using RGB(192,192,192) as the background color.

The image can be specified in three ways:

- As an absolute path name of a .bmp file.
- As an `IPictureDisp` interface. An `IPictureDisp` interface can be created in Natural using the method `INatAutoImages::LoadImage`. An `IPictureDisp` interface cannot be passed across process boundaries. This is due to a Microsoft restriction (MSDN Q150034). Therefore this alternative can only be used with plug-ins running as in-process servers. Natural written plug-ins always run as local servers and can therefore not use this alternative.

- As an index into a table of user images pre-defined in Natural Studio. These are the images that can be assigned to user commands in the **Customize** dialog.

### Item

Returns a specific command from the collection. Used to iterate through the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| Index | I4 | VT_I4 | |

**Return value**
The command identified by the value specified in Index.

**Index**
The index of the command in the collection (a value between 1 and Count).

# 12 INatAutoContextMenu

## Purpose

Gives access to a specific context menu.

## Properties

The following properties are available:

- Parent
- Studio
- Count
- Caption

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoContextMenus**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of items (commands, separators and pop-up menus) in the menu.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Caption

A string used to identify the menu, as defined when the menu was created.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following methods are available:

- Item
- SubMenu
- InsertCommand
- InsertSeparator
- InsertPopupMenu
- UpdateMenu

### Item

Returns a specific item contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The caption of the menu item (command or pop-up menu) identified by the value specified in Index. If the index identifies a separator, an empty string is returned.

**Index**
The index of the item in the menu (a value between 1 and Count).

### SubMenu

Returns a specific pop-up menu contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**

The pop-up menu identified by the value specified in Index. If the specified index does not identify a pop-up menu, but a command or a separator, a null interface pointer (NULL-HANDLE) is returned.

**Index**

As index either a number between 1 and Count or the caption of a pop-up menu can be specified.

### InsertCommand

Inserts a command into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Command | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| Index | I4 | VT_I4 | Optional |

**Command**

A command to be added to the menu. The command must have been defined before using the method `INatAutoCommands::Add`.

**Index**

The position in the menu where the command shall be inserted. If Index is omitted, the command is inserted at the last position.

### InsertSeparator

Inserts a separator into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Index | I4 | VT_I4 | Optional |

**Index**
The position in the menu where the separator shall be inserted. If Index is omitted, the separator is inserted at the last position.

### InsertPopupMenu

Creates a new pop-up menu and inserts it into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Caption | A | VT_BSTR | |
| Index | I4 | VT_I4 | Optional |

**Return value**
The newly created pop-up menu.

**Caption**
A string used to identify the pop-up menu.

**Index**
The position in the menu where the pop-up menu shall be inserted. If Index is omitted, the pop-up menu is inserted at the last position.

### UpdateMenu

Changes in a menu are not made visible immediately, in order to avoid flickering. After having finished modifying a menu, make the recent changes visible by calling this method.

# 13 INatAutoContextMenus

## Purpose

Collection of the available context menus.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of available context menus.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Add
- Item

### Add

Creates a new context menu and adds it to the collection. Dynamically created context menus are not persistently customizable in the **Customize** dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoContextMenu**) | |
| Caption | A | VT_BSTR | |

**Return value**
The newly added context menu.

**Caption**
A string used to identify the context menu.

### Item

Returns a specific context menu from the collection. Used to iterate through the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoContextMenu**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The context menu identified by the value specified in Index.

**Index**
Identifies a specific context menu in the collection. This can be either the index of the context menu in the collection (a value between 1 and Count) or the caption of the context menu (as indicated in the **Customize** dialog).

# 14 INatAutoControlBars

## Purpose

Contains collections used to access the Natural Studio toolbars, frame menus and context menus and to define new commands.

## Properties

The following properties are available:

- Parent
- Studio
- Images
- Commands
- ToolBars
- FrameMenus
- ContextMenus

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Images

Used to navigate to the `INatAutoImages` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoImages**) | Get only |

### Commands

Used to navigate to the `INatAutoCommands` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommands**) | Get only |

### ToolBars

Used to navigate to the `INatAutoToolBars` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoToolBars**) | Get only |

### FrameMenus

Used to navigate to the `INatAutoFrameMenus` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoFrameMenus**) | Get only |

### ContextMenus

Used to navigate to the `INatAutoContextMenus` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoContextMenus**) | Get only |

# 15 INatAutoDataArea

## Purpose

A data area open in a data area editor window. This comprises the following development object types: local data area, global data area and parameter data area.

## Properties

The following properties are available:

- Parent
- Studio
- Source
- Visible
- Type
- LineCount

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDataAreas**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Source

The source code of the data area in the syntax of the `DEFINE DATA` statement.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Visible

Shows or hides the editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### Type

The development object type. The type is identified by a numeric ID. The IDs of predefined types are described in the section *Predefined Node Types*.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### LineCount

The number of lines in the source code.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- StartEdit
- EndEdit
- Catalog
- Check
- Clear
- Close
- Search
- Replace
- Save
- Stow
- Title
- GetInfo

- DeleteLines
- GetLines
- InsertLines
- ReplaceLines

### StartEdit

This method should be called before calling a series of editing methods in order to increase editing performance. It converts (internally) the data area into source code according to the syntax of the `DEFINE DATA` statement.

### EndEdit

This method should be called after having called `StartEdit` and a series of editing methods. It converts (internally) the source code back into the data area editor.

### Catalog

Catalogs the data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Check

Checks the data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Clear

Clears the data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Close

Closes the editor and removes the data area from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Search

Searches for the first occurrence of a given string in the source code (in the syntax of the `DEFINE DATA` statement).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction. The default is FALSE.

### Replace

Replaces the first occurrence of a given string in the source code (in the syntax of the `DEFINE DATA` statement) with another one.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| ReplaceString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**ReplaceString**
The string which replaces the search string.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction.

### Save

Saves the data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Type | I4 | VT_I4 | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Saves the object under the given name.

**Library**
Saves the object in the given library.

**Type**
Saves the object under the given type.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Stow

Stows the data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Type | I4 | VT_I4 | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Stows the object under the given name.

**Library**
Stows the object in the given library.

**Type**
Stows the object under the given type.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Title

Titles an untitled data area.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |

**Name**
Assigns a name to the data area.

**Library**
Assigns a library to the data area.

### GetInfo

Returns information about an open object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | By reference |
| Name | A | VT_BSTR | By reference |
| Library | A | VT_BSTR | By reference |
| Fnr | I4 | VT_I4 | By reference |
| DBnr | I4 | VT_I4 | By reference |

**Type**
The type of the object.

**Name**
The name of the object.

**Library**
The library of the object.

**Fnr**
The system file file number of the object.

**DBnr**
The system file database number of the object.

### DeleteLines

Deletes a block of lines from the source code (in the syntax of the `DEFINE DATA` statement).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**StartLine**
The start line of the block to delete.

**LineCount**
The number of lines to delete. The default is 1.

### GetLines

Retrieves a block of lines from the source code (in the syntax of the `DEFINE DATA` statement).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
A block of source code lines. The lines are separated by carriage return / line feed characters.

**StartLine**
The start line of the block to return.

**LineCount**
The number of lines to return. The default is 1.

### InsertLines

Inserts a block of lines from the source code (in the syntax of the `DEFINE DATA` statement).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| InsertAfterLine | I4 | VT_I4 | Optional |

**Return value**
The line number passed in InsertAfterLine increased by the number of inserted lines.

**Code**
A block of source code lines to insert. The lines must be separated by carriage return / line feed characters.

**InsertAfterLine**
Line after which the code shall be inserted. The default is 0.

### ReplaceLines

Replaces a block of lines from the source code (in the syntax of the `DEFINE DATA` statement).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| ReplaceLine | I4 | VT_I4 | Optional |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
The line number passed in ReplaceLine increased by the number of inserted lines.

**Code**
A block of source code lines to replace the block that is defined by ReplaceLine and LineCount. The lines must be separated by carriage return / line feed characters.

**ReplaceLine**
The start line of the block to be replaced. The default is 1.

**LineCount**
The number of lines to be replaced by the given block. The default is 1.

# 16 INatAutoDataAreas

## Purpose

Collection of the development objects currently open in a data area editor window. This collection comprises the following development object types: local data area, parameter data area and global data area. The types are identified by a numeric ID. The IDs of predefined types are described in the section *Predefined Node Types*.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of development objects currently open in a data area editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

# Methods

The following methods are available:

- Item
- Add
- Open

## Item

Returns a specific development object from the collection.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDataArea**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |
| Type | I4 | VT_I4 | Optional |
| Library | A | VT_BSTR | Optional |

**Return value**
The development object identified by the value specified in Index.

**Index**
Identifies a specific development object in the collection. This can be either the index of the development object in the collection (a value between 1 and Count) or the name of the object.

**Type**
Used to identify a specific object by name (specified in Index) and type (specified in Type).

**Library**
Used to identify a specific object by name (specified in Index), type (specified in Type) and library (specified in Library).

### Add

Creates a new (untitled) development object and opens it in a data area editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDataArea**) | |
| Type | I4 | VT_I4 | |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly created development object.

**Type**
The type of the object to create.

**Visible**
Decides if the editor is opened visible or not. By default, the editor is opened visible.

### Open

Opens an existing development object in a data area editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDataArea**) | |
| Type | I4 | VT_I4 | |
| Name | A | VT_BSTR | |
| Library | A | VT_BSTR | Optional |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly opened development object.

**Type**
The type of the object to open.

**Name**
The name of the object to open.

**Library**
The library of the object to open.

**Visible**

Decides if the editor is opened visible or not. By default, the editor is opened visible.

# 17 INatAutoDialog

## Purpose

A dialog currently open in a dialog editor window.

## Properties

The following properties are available:

- Parent
- Studio
- Source
- Visible
- LineCount

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDialogs**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Source

The source code of the dialog.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Visible

Shows or hides the editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### LineCount

The number of lines in the source code.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- StartEdit
- EndEdit
- Catalog
- Check
- Clear
- Close
- Execute
- Search
- Replace
- Run
- Save
- Stow
- Title
- GetInfo
- DeleteLines
- GetLines
- InsertLines

- ReplaceLines

### StartEdit

This method should be called before calling a series of editing methods in order to increase editing performance. It converts (internally) the dialog specification into source code.

### EndEdit

This method should be called after having called `StartEdit` and a series of editing methods. It converts (internally) the source code back into a dialog specification.

### Catalog

Catalogs the dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Check

Checks the dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Clear

Clears the editor contents.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Close

Closes the editor and removes the dialog from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Execute

Executes the dialog.

### Search

Searches for the first occurrence of a given string.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction. The default is FALSE.

### Replace

Replaces the first occurrence of a given string with another one.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| ReplaceString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**ReplaceString**
The string which replaces the search string.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction.

### Run

Runs the dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Save

Saves the dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Saves the dialog under the given name.

**Library**
Saves the dialog in the given library.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Stow

Stows the dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Stows the dialog under the given name.

**Library**
Stows the dialog in the given library.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Title

Titles an untitled dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |

**Name**
Assigns a name to the dialog.

**Library**
Assigns a library to the dialog.

### GetInfo

Returns information about an open dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | By reference |
| Name | A | VT_BSTR | By reference |
| Library | A | VT_BSTR | By reference |
| Fnr | I4 | VT_I4 | By reference |
| DBnr | I4 | VT_I4 | By reference |

**Type**
The type of the object. Always a dialog.

**Name**
The name of the dialog.

**Library**
The library of the dialog.

**Fnr**
The system file file number of the object.

**DBnr**
The system file database number of the object.

### DeleteLines

Deletes a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**StartLine**
The start line of the block to delete.

**LineCount**
The number of lines to delete. The default is 1.

### GetLines

Retrieves a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
A block of source code lines. The lines are separated by carriage return / line feed characters.

**StartLine**
The start line of the block to return.

**LineCount**
The number of lines to return. The default is 1.

### InsertLines

Inserts a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| InsertAfterLine | I4 | VT_I4 | Optional |

**Return value**
The line number passed in InsertAfterLine increased by the number of inserted lines.

**Code**
A block of source code lines to insert. The lines must be separated by carriage return / line feed characters.

**InsertAfterLine**
Line after which the code shall be inserted. The default is 0.

### ReplaceLines

Replaces a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| ReplaceLine | I4 | VT_I4 | Optional |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
The line number passed in ReplaceLine increased by the number of inserted lines.

**Code**
A block of source code lines to replace the block that is defined by ReplaceLine and LineCount. The lines must be separated by carriage return / line feed characters.

**ReplaceLine**
The start line of the block to be replaced. The default is 1.

**LineCount**
The number of lines to be replaced by the given block. The default is 1.

# 18 INatAutoDialogs

## Purpose

Collection of the dialogs currently open in a dialog editor window.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of dialogs currently open in a data area editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Add
- Open

### Item

Returns a specific dialog from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDialog**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |
| Library | A | VT_BSTR | Optional |

**Return value**
The dialog identified by the value specified in Index.

**Index**
Identifies a specific dialog in the collection. This can be either the index of the dialog in the collection (a value between 1 and Count) or the name of the dialog.

**Library**
Used to identify a specific dialog by name (specified in Index) and library (specified in Library).

### Add

Creates a new (untitled) dialog and opens it in a dialog editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDialog**) | |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly created dialog.

**Visible**
Decides if the editor is opened visibly or not. By default, the editor is opened visibly.

### Open

Opens an existing dialog in a dialog editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDialog**) | |
| Name | A | VT_BSTR | |
| Library | A | VT_BSTR | Optional |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly opened dialog.

**Name**
The name of the dialog to open.

**Library**
The library of the dialog to open.

**Visible**
Decides if the editor is opened visibly or not. By default, the editor is opened visibly.

# 19 INatAutoEnvironment

## Purpose

An environment that has once been connected during the current Natural Studio session. This includes the local environment also. Remote environments are only available with Natural Single Point of Development. See also *Remote Development Using SPoD*.

## Properties

The following properties are available:

- Parent
- Studio
- Local
- Active
- Connected
- Name
- Host
- Port
- Alias
- CommandLine
- UserID
- Parameters
- SystemVariables

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironments**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Local

TRUE if this is the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Active

TRUE if this is the active environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Connected

TRUE if this environment is currently connected.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### Name

The name of the environment. This name can be used in the method `INatAutoEnvironments::Item` to identify a specific environment uniquely.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Host

The host name of the development server.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Port

The port number of the development server.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Alias

The alias name of the environment as displayed in the library workspace.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### CommandLine

A command line containing additional dynamic parameters under which the environment is running.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### UserID

The user ID under which the environment is mapped.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Parameters

Returns an interface to the `NATPARM` parameters of this environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNatParm**) | Get only |

### SystemVariables

Returns an interface to the system variables of the environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNatsvar**) | Get only |

## Methods

The following methods are available:

- Activate
- Disconnect
- Connect
- Unmap

### Activate

Makes this environment the active one. An environment cannot explicitly be deactivated. An environment is implicitly deactivated when another one is activated.

This method is not applicable to environments that are disconnected.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Visible | L | VT_BOOL | Optional |

**Visible**

This parameter can be used to temporarily activate a different environment and then reactivate the previous environment, without affecting the user interface.

If set to TRUE or not specified, the newly activated environment is selected in the library workspace.

If set to FALSE, the previously active environment stays selected.

### Disconnect

Disconnects the environment and closes the development server session. If this environment was the active one, the previously active environment gets activated.

This method is not applicable to the local environment.

### Connect

Reestablishes the connection to a previously disconnected environment. Activates the connected environment.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Password | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |
| IgnoreWarnings | L | VT_BOOL | Optional |

**Password**
The password of the user ID under which the environment was previously connected.

**Quiet**
If set to FALSE or not specified, the **Map Environment** dialog is shown and is preset with the given parameters.

If set to TRUE, the **Map Environment** dialog is not shown.

**IgnoreWarnings**
If set to FALSE or not specified, warnings that occur during connecting are treated like errors.

If set to TRUE, warnings are ignored.

### Unmap

Unmaps the environment, disconnects it and closes the development server session. If this environment was the active one, the previously active environment gets activated.

This method is not applicable to the local environment.

# 20 INatAutoEnvironments

## Purpose

Collection of the development environments that have once been connected during the current Natural Studio session. This includes the local environment. Remote environments are only available with Natural Single Point of Development. See also *Remote Development Using SPoD*.

## Properties

The following properties are available:

- Parent
- Studio
- Count
- LocalEnvironment
- RemoteEnvironment
- ActiveEnvironment

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSystem**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of environments in the collection.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### LocalEnvironment

Returns the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### RemoteEnvironment

This property is useful only for Natural system commands and utilities. If a Natural system command or utility is executed in the local environment, but is supposed to operate on a certain remote environment, this propery returns that remote environment. Otherwise it returns NULL-HANDLE.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### ActiveEnvironment

Returns the currently active environment. Returns NULL-HANDLE if no environment is active.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

## Methods

The following methods are available:

- Item

- Add

### Item

Retrieves a specific environment from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The environment identified by the value specified in Index.

**Index**
Identifies a specific environment in the collection. This can be either the index of the environment in the collection (a value between 1 and Count) or the name of the environment. The name of the environment is the value of the property `INatAutoEnvironment::Name`.

### Add

Maps a remote environment, adds it to the collection and activates it. Returns the environment.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | |
| Host | A | VT_BSTR | Optional |
| Port | I4 | VT_I4 | Optional |
| Alias | A | VT_BSTR | Optional |
| CommandLine | A | VT_BSTR | Optional |
| UserID | A | VT_BSTR | Optional |
| Password | A | VT_BSTR | Optional |
| Quiet | L | VT_BOOL | Optional |
| IgnoreWarnings | L | VT_BOOL | Optional |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly mapped environment.

**Host**
The host name of the development server.

**Port**
The port number of the development server.

**Alias**
An alias name for the environment that is displayed in the library workspace. If no alias name is specified, a unique name will be generated.

**CommandLine**
A command line containing additional dynamic parameters under which the environment will be running.

**UserID**
The user ID under which the environment will be mapped.

**Password**
The password of the user ID under which the environment will be mapped.

**Quiet**
If set to FALSE or not specified, the **Map Environment** dialog is shown and is preset with the given parameters.

If set to TRUE, the **Map Environment** dialog is not shown.

**IgnoreWarnings**
If set to FALSE or not specified, warnings that occur during mapping are treated like errors.

If set to TRUE, warnings are ignored.

**Visible**
If set to TRUE or not specified, the environment is displayed in the library workspace.

If set to FALSE, the environment is not displayed.

# 21 INatAutoFrameMenu

## Purpose

Gives access to a specific frame menu.

## Properties

The following properties are available:

- Parent
- Studio
- Count
- Caption

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoFrameMenus**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of items (commands, separators and pop-up menus) in the menu.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Caption

A string used to identify the menu, as defined when the menu was created.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following methods are available:

- Item
- SubMenu
- InsertCommand
- InsertSeparator
- InsertPopupMenu
- UpdateMenu

### Item

Returns a specific item contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_BSTR | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The caption of the menu item (command or pop-up menu) identified by the value specified in Index. If the index identifies a separator, an empty string is returned.

**Index**
The index of the item in the menu (a value between 1 and Count).

### SubMenu

Returns a specific pop-up menu contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**

The pop-up menu identified by the value specified in Index. If the specified index does not identify a pop-up menu, but a command or a separator, a null interface pointer (NULL-HANDLE) is returned.

**Index**

As index either a number between 1 and Count or the caption of a pop-up menu can be specified.

### InsertCommand

Inserts a command into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | | | None |
| Command | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| Index | I4 | VT_I4 | Optional |

**Command**

A command to be added to the menu. The command must have been defined before using the method `INatAutoCommands::Add`.

**Index**

The position in the menu where the command shall be inserted. If Index is omitted, the command is inserted at the last position.

### InsertSeparator

Inserts a separator into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | | | None |
| Index | I4 | VT_I4 | Optional |

**Index**
The position in the menu where the separator shall be inserted. If Index is omitted, the separator is inserted at the last position.

### InsertPopupMenu

Creates a new pop-up menu and inserts it into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Caption | A | VT_BSTR | |
| Index | I4 | VT_I4 | Optional |

**Return value**
The newly created pop-up menu.

**Caption**
A string used to identify the pop-up menu.

**Index**
The position in the menu where the pop-up menu shall be inserted. If Index is omitted, the pop-up menu is inserted at the last position.

### UpdateMenu

Changes in a menu are not made visible immediately, in order to avoid flickering. After having finished modifying a menu, make the recent changes visible by calling this method.

# 22 INatAutoFrameMenus

## Purpose

Collection of the available frame menus.

Frame menus must have a consistent layout throughout an application. It is therefore not useful for a Natural Studio plug-in to create arbitrary frame menus at will. The best approach for a plug-in is

- thinking about the document types it is going to represent in document windows;
- specifying a frame menu for each of these types;
- creating these frame menus as clones of the default frame menu for plug-ins;
- extending them in order to cover the needs of the document type, in a manner consistent with the document-specific frame menus in Natural Studio.

## Properties

The following properties are available:



### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number available frame menus.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Add
- Clone
- Item

### Add

Creates a new frame menu and adds it to the collection. Dynamically created frame menus are not persistently customizable in the **Customize** dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoFrameMenu**) | |
| Caption | A | VT_BSTR | |

**Return value**
The newly added frame menu.

**Caption**
A string used to identify the frame menu.

### Clone

Creates a new frame menu as a copy of an existing frame menu and adds it to the collection. Dynamically created frame menus are not persistently customizable in the **Customize** dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (INatAutoFrameMenu) | |
| Caption | A | VT_BSTR | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | Optional |

**Return value**
The newly added frame menu.

**Caption**
A string used to identify the frame menu.

**Index**
Identifies the frame menu to be cloned. This can be either the index of the frame menu in the collection (a value between 1 and Count) or the caption of the frame menu (as indicated in the **Customize** dialog). If this parameter is omitted, the built-in frame menu Plug-in MDI View is cloned.

### Item

Returns a specific frame menu from the collection. Used to iterate through the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (INatAutoFrameMenu) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The frame menu identified by the value specified in Index.

**Index**
Identifies a specific frame menu in the collection. This can be either the index of the frame menu in the collection (a value between 1 and Count) or the caption of the frame menu (as indicated in the **Customize** dialog).

# 23 INatAutoGenericDocument

## Purpose

A currently open generic document window.

## Properties

The following properties are available:

- Parent
- Studio
- DialogID
- PlugInID
- Caption

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericDocuments**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### DialogID

The dialog ID of the Natural dialog that implements the window. This is the dialog ID that was passed to the `Add` method during creation of the generic document window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### PlugInID

The ID of the plug-in that created the document window. A plug-in can use this property to iterate only across its own document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Caption

The caption of the document window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following method is available:

- Close

### Close

Closes the generic document window. This method is executed asynchronously. In particular, this means that if it is called from within the method `OnCommand` of a plug-in, the window will only really be closed after the method `OnCommand` has terminated.

## Notifications

A plug-in that has created a generic document window can expect to receive the following notifications through the method `OnNotify`:

- `PLUGIN-NOTIFY-ACTIVATE`
- `PLUGIN-NOTIFY-QUERYCLOSE`
- `PLUGIN-NOTIFY-CLOSE`
- `PLUGIN-NOTIFY-SELECTEDOBJECTS`
- `PLUGIN-NOTIFY-FOCUSOBJECT`
- `PLUGIN-NOTIFY-CONTEXTMENU`

- PLUGIN-NOTIFY-REFRESH

# 24 INatAutoGenericDocuments

## Purpose

Collection of the currently open generic document windows.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of currently open generic document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Add

### Item

Returns a specific generic document window from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericDocument**) | |
| Index | I4 | VT_I4 | |

**Return value**
The generic document window identified by the value specified in Index.

**Index**
Identifies a specific generic text object in the collection by its index in the collection (a value between 1 and Count).

### Add

Creates a new generic document window from a Natural dialog instance. Before creating a generic document window, the plug-in opens a Natural dialog of type “MDI plug-in window” with the `OPEN DIALOG` statement. In order to make the dialog appear in Natural Studio as a document window, the plug-in passes the dialog ID returned from the `OPEN DIALOG` statement to the method `Add` in the parameter `DialogID`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericDocument**) | |
| DialogID | I4 | VT_I4 | |
| Caption | A | VT_BSTR | Optional |
| IconFile | A | VT_BSTR | Optional |
| FrameMenu | A | VT_DISPATCH (**INatAutoFrameMenu**) | Optional |

**Return value**

An interface to the newly created generic document window.

**DialogID**

A dialog ID that was returned from a previous `OPEN DIALOG` statement with a Natural dialog of type “MDI plug-in window”.

**Caption**

The caption to be displayed in the generic document window. If Caption is omitted, the caption defined in the corresponding Natural dialog will be displayed.

**IconFile**

The file and path name of the .ico file that contains the icon to be displayed in the generic document window. If IconFile is omitted, the icon defined in the corresponding Natural dialog will be displayed.

**FrameMenu**

The frame menu to be displayed with the generic document window. If FrameMenu is omitted, the default frame menu for plug-in document windows will be used. But usually the plug-in will create its own frame menu by cloning (see `INatAutoFrameMenus::Clone`) this default menu and extending the clone according to its requirements.

# 25 INatAutoGenericText

## Purpose

A generic (non-Natural) text object currently open in a program editor window.

## Properties

The following properties are available:

- Parent
- Studio
- Source
- Visible
- Type
- Name
- LineCount

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericTexts**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Source

The text contained in the text object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Visible

Shows or hides the editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### Type

The type of the text object. The type is defined as a text string by the plug-in during the method `Open`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Name

The name of the text object. The name is defined as a text string by the plug-in during the method `Open`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### LineCount

The number of lines in the text object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:



### Clear

Clears the editor contents.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Close

Closes the editor and removes the object from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Renumber

Renumbers the text object.

### Search

Searches for the first occurrence of a given string.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction. The default is FALSE.

### Replace

Replaces the first occurrence of a given string with another one.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| ReplaceString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**ReplaceString**
The string which replaces the search string.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction.

### DeleteLines

Deletes a block of lines from the text object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**StartLine**
The start line of the block to delete.

**LineCount**
The number of lines to delete. The default is 1.

### GetLines

Retrieves a block of lines from the text object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
A block of text lines. The lines are separated by carriage return / line feed characters.

**StartLine**
The start line of the block to return.

**LineCount**
The number of lines to return. The default is 1.

### InsertLines

Inserts a block of lines into the text object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| InsertAfterLine | I4 | VT_I4 | Optional |

**Return value**
The line number passed in InsertAfterLine increased by the number of inserted lines.

**Code**
A block of text lines to insert. The lines must be separated by carriage return / line feed characters.

**InsertAfterLine**
Line after which the lines shall be inserted. The default is 0.

### ReplaceLines

Replaces a block of lines from the text object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| ReplaceLine | I4 | VT_I4 | Optional |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
The line number passed in ReplaceLine increased by the number of inserted lines.

**Code**
A block of source code lines to replace the block that is defined by ReplaceLine and LineCount. The lines must be separated by carriage return / line feed characters.

**ReplaceLine**
The start line of the block to be replaced. The default is 1.

**LineCount**
The number of lines to be replaced by the given block. The default is 1.

# 26 INatAutoGenericTexts

## Purpose

Collection of the generic (non-Natural) text objects currently open in a program editor window.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of generic text objects currently open in a program editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Open

### Item

Returns a specific generic text object from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericText**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |
| Type | A | VT_BSTR | Optional |

**Return value**
The generic text object identified by the value specified in Index.

**Index**
Identifies a specific generic text object in the collection. This can be either the index of the text object in the collection (a value between 1 and Count) or the name of the object.

**Type**
Used to identify a specific object by name (specified in Index) and type (specified in Type). The type is a string freely defined by the plug-in when the text object was opened in the editor.

### Open

Opens a generic text object in a program editor window.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericText**) | |
| Type | A | VT_BSTR | |
| Name | A | VT_BSTR | |
| Buffer | A | VT_BSTR | |
| Visible | L | VT_BOOL | Optional |

**Return value**

The newly opened generic text object.

**Type, Name**

Type and Name are freely defined by the calling plug-in to identify a generic text object to the user. Natural Studio takes these values to create a window caption for the editor window ("*name* - *type*") and to prompt users if they attempt to close an unsaved editing session ("Apply changes to *type name*?").

Note that the name space of *type* is shared between all callers of the interface. There are no means taken to inhibit different plug-ins from using the same type identifiers. In order to avoid confusing users, plug-ins should not choose their type identifiers too generic. Good example for a type identifier: "Package Description". Bad example: "Description".

**Buffer**

Contains the data that is passed to the editor initially. Line breaks in the text must be denoted with carriage return and line feed characters.

**Visible**

Decides if the editor is opened visibly or not. By default, the editor is opened visibly.

# 27 INatAutoImages

## Purpose

Used to define images that represent commands in menus and toolbars.

## Properties

The following properties are available:

- Parent
- Studio

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

## Methods

The following method is available:

- LoadImage

## LoadImage

Loads an image from a file. The resulting `IPictureDisp` interface can, for example, be used to assign the image to a command when adding a command to the `INatAutoCommands` collection. `IPictureDisp` is an Automation interface predefined in Windows.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (IPictureDisp) | |
| ImageFileName | A | VT_BSTR | |

**Return value**
The loaded image.

**ImageFileName**
Name of a bitmap file (.bmp) with full path name that contains the image to be loaded.

# 28 INatAutoLinkedApplications

## Purpose

Collection of applications that are linked to a compound application.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of applications in the collection.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Add
- Remove

### Item

Returns a specific application from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The application identified by the value specified in Index.

**Index**
Identifies a specific application in the collection. This can be either the index of the application in the collection (a value between 1 and Count) or the name of the application.

### Add

Adds the given application to the collection (effectively links it to the parent application).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | |
| Application | A | VT_BSTR | |

**Return value**
The newly linked application.

**Application**
Name of the application to be linked.

### Remove

Removes the application from the collection (effectively unlinks it from the parent application).

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Index**

Identifies a specific application in the collection. This can be either the index of the application in the collection (a value between 1 and Count) or the name of the application.

# 29 INatAutoNatparm

## Purpose

Gives access to certain parameters of a specific Natural development environment.

## Properties

The following properties are available:

- Parent
- Studio
- CurrentLibrary
- CurrentDBnr
- CurrentFnr
- FnatDBnr
- FnatFnr
- FuserDBnr
- FuserFnr
- FdicDBnr
- FdicFnr
- FddmDBnr
- FddmFnr
- ProfileParameters

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### CurrentLibrary

The name of the current logon library.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### CurrentDBnr

The database number of the system file where the current logon library is located.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### CurrentFnr

The file number of the system file where the current logon library is located.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FnatDBnr

The database number of the Natural system file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FnatFnr

The file number of the Natural system file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FuserDBnr

The database number of the user system file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FuserFnr

The file number of the user system file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FdicDBnr

The database number of the development server file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FdicFnr

The file number of the development server file.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FddmDBnr

The database number of the system file for DDMs.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### FddmFnr

The file number of the system file for DDMs.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### ProfileParameters

A string containing an XML document that contains the current values of a defined subset of the Natural profile parameters. For the meaning of the individual parameters, refer to the *Parameter Reference*. In the case of a remote environment, the document contains only a defined subset of the profile parameters.

For the local environment, the XML document is structured according to the DTD provided in the section ***DTD for INatAutoNatparm - Local Environment***.

For a remote environment, the XML document is structured according to the DTD provided in the section ***DTD for INatAutoNatparm - Remote Environment***.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

# 30 INatAutoNatsvar

## Purpose

Gives access to some of the system variables of a Natural development environment.

## Properties

The following properties are available:

- Parent
- Studio
- Language
- SystemVariables

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Language

The system variable `*LANGUAGE`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I2 | VT_I2 | Get only |

### SystemVariables

A string containing an XML document that contains the current values of a defined subset of the Natural system variables. For the meaning of the individual system variables, refer to the *System Variables* documentation. In the case of a remote environment, the system variables cannot be delivered and the property contains an empty string.

For the local environment, the XML document is structured according to the DTD provided in the section ***DTD for INatAutoNatsvar - Local Environment***.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

# 31 INatAutoNodeImages

## Purpose

A collection of images that shall be used to represent a user defined node type in tree views or list views. Each image is identified by an integer value.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoTypes**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of available node images.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following method is available:

- AddImage

### AddImage

Adds a new image to the collection.

A list view node requires a 16x16 bitmap to represent the node in the “Small icons” view and a 32x32 bitmap for the “Large icons” view. A tree view node requires two 16x16 bitmaps, one representing the closed state and one representing the open state.

In order to register a 16x16 bitmap, the plug-in passes the bitmap in the parameter `ImageSmall`. In order to register additionally a corresponding 32x32 bitmap, the plug-in passes the bitmap in the parameter `ImageLarge`.

In order to register images for a node that shall be represented both in list views and in tree views, the plug-in calls `AddImage` once with the 16x16 bitmap representing the closed state and the corresponding 32x32 bitmap. Then it calls `AddImage` a second time passing the 16x16 bitmap representing the open state and omitting the second parameter.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| ImageSmall | A<br>HANDLE OF OBJECT | VT_BSTR<br>VT_DISPATCH (IPictureDisp) | |
| ImageLarge | A<br>HANDLE OF OBJECT | VT_BSTR<br>VT_DISPATCH (IPictureDisp) | Optional |

**Return value**

An integer value that can later be used to refer to the image in the method `INatAutoNodeTypes::Add`.

**ImageSmall**

A 16x16 bitmap. Areas in the bitmap that contain the color RGB(0,128,128) will be displayed transparent.

The bitmap can be specified in two ways:

- As an absolute path name of a .bmp file.

- As an `IPictureDisp` interface. An `IPictureDisp` interface can be created in Natural using the method `INatAutoImages::LoadImage`. Note that an `IPictureDisp` interface cannot be passed across process boundaries. This is due to a Microsoft restriction (MSDN Q150034). Therefore this alternative can only be used with plug-ins running as in-process servers. Natural written plug-ins always run as local servers and can therefore not use this alternative.

**ImageLarge**

A 32x32 bitmap. Areas in the bitmap that contain the color RGB(0,128,128) will be displayed transparent.

The bitmap can be specified in two ways:

- As an absolute path name of a .bmp file.
- As an `IPictureDisp` interface. An `IPictureDisp` interface can be created in Natural using the method `INatAutoImages::LoadImage`. Note that an `IPictureDisp` interface cannot be passed across process boundaries. This is due to a Microsoft restriction (MSDN Q150034). Therefore this alternative can only be used with plug-ins running as in-process servers. Natural written plug-ins always run as local servers and can therefore not use this alternative.

# 32 INatAutoNodeType

## Purpose

A node type used in tree views and list views.

## Properties

The following properties are available:

- Parent
- Studio

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNodeTypes**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

# 33 INatAutoNodeTypes

## Purpose

Collection of all node types used in tree views and list views.

## Properties

The following properties are available:

- Parent
- Studio

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoTypes**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

## Methods

The following method is available:

- Add

## Add

Creates a new node type and adds it to the collection.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNodeType**) | |
| Type | I4 | VT_I4 | |
| Caption | A | VT_BSTR | Optional |
| ContextMenu | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoContextMenu**) | Optional |
| ImageIDDefault | I4 | VT_I4 | Optional |
| ImageIDOpen | I4 | VT_I4 | Optional |

**Return value**
The newly created node type.

**Type**
An integer number that identifies the new node type. An arbitrary positive integer value starting with 20000 can be chosen. Values below 20000 are reserved for predefined node types.

**Caption**
A name for the node type for the use in tree view and list view captions.

**ContextMenu**
A context menu that shall be displayed when the right mouse button is pressed on a node of this type.

**ImageIDDefault**
An index to the small (16x16) and large (32x32) version of the default bitmap representation of nodes of this type. If the plug-in has registered node bitmaps with the method `INatAutoNodeImages::AddImage`, it has received an index that can be used here. If the parameter is not specified, the nodes of this type are represented as closed folders.

**ImageIDOpen**
An index to the bitmap that represents an expanded node of this type in a tree view. If the plug-in has registered node bitmaps with the method `INatAutoNodeImages::AddImage`, it has received an index that can be used here. If the parameter is not specified, expanded nodes of this type are represented as open folders.

# 34 INatAutoObjectList

## Purpose

An open list view document window.

## Properties

The following properties are available:

- Parent
- Studio
- PlugInID

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectLists**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### PlugInID

The ID of the plug-in that created the list view document window. A plug-in can use this property to iterate only across its own document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following method is available:

- Close

### Close

Closes the list view document window. This method is executed asynchronously. In particular, this means that if it is called from within the method `OnCommand` of a plug-in, the window will only really be closed after the method `OnCommand` has terminated.

## Notifications

A plug-in that has created a list view document window can expect to receive the following notifications through its method `OnNotify`.

- `PLUGIN-NOTIFY-ACTIVATE`
- `PLUGIN-NOTIFY-CLOSE`
- `PLUGIN-NOTIFY-REFRESH`
- `PLUGIN-NOTIFY-HELP`

# 35 INatAutoObjectLists

## Purpose

Collection of the currently open list view document windows.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of currently open list view document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

# Methods

The following methods are available:

- Item
- Open

## Item

Returns a specific list view document window from the collection.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectList**) | |
| Index | I4 | VT_I4 | |

**Return value**
The list view document window identified by the value specified in Index.

**Index**
Identifies a specific list view document window in the collection. This is a value between 1 and Count.

## Open

Opens a new list view document window.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectList**) | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Caption | A | VT_BSTR | |
| Template | I4 | VT_I4 | Optional |
| PlugInID | A | VT_BSTR | Optional |
| Info | A | VT_BSTR | Optional |
| DisplayName | A | VT_BSTR | Optional |
| Children | A | VT_BSTR | Optional |
| NaturalType | I4 | VT_I4 | Optional |

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| NaturalKey | A | VT_BSTR | Optional |

**Return value**
The newly opened list view document.

**Type**
The node type of the root object. This must be a plug-in defined type that may correspond to a predefined Natural Studio type.

**Key**
The key that identifies the object within its type.

**Caption**
Determines the caption of the document window.

In order to generate a caption that matches the caption format used in Natural Studio document windows, the string passed in this parameter may contain the format specifier `%std`. This format specifier will be dynamically replaced by an identifier of the root object in the following way: If the root node corresponds to a predefined Natural Studio object, the format specifier `%std` is replaced by "*name* [*library*] -". Otherwise the identifier is built in the form "*node-type-caption display-name* -", where *node-type-caption* is the caption that was passed when the node type was created and *display-name* is the string passed in DisplayName.

**Template**
The value specified here is passed back to the plug-in when Natural Studio later calls the plug-in through the interface `INaturalStudioPlugInTree` in order to retrieve data to fill the view. This enables the plug-in to deliver different data for the same root object.

**PlugInID**
The ID of the plug-in that defined the type.

**Info**
Additional information that a plug-in may want to assign to the root object. Natural Studio does not interpret this information. It just passes it back to the plug-in when it later calls the plug-in through the interface `INaturalStudioPlugInTree` in order to retrieve data to fill the view.

**DisplayName**
The text to be displayed at the root node.

**Children**
If the plug-in already knows the child nodes of the root node, it can pass them in this parameter. The parameter must then contain an XML document describing the child nodes. The XML document is structured according to the DTD defined in `INaturalStudioPlugInTree::GetChildren`. If this parameter is specified, Natural Studio implicitly assumes that the root node has child nodes and performs no call through the interface `INaturalStudioPlugInTree` to determine if the root node has child nodes.

**NaturalType**

If the root node corresponds to a predefined Natural Studio object, this property contains the type of the corresponding predefined object.

**NaturalKey**

If the root node corresponds to a predefined Natural Studio object, this property contains the key of the corresponding predefined object.

# 36 INatAutoObjects

## Purpose

Used to navigate to the collections of development objects and other related collections.

## Properties

The following properties are available:

- Parent
- Studio
- Programs
- Dialogs
- DataAreas
- ObjectTrees
- ObjectLists
- GenericTexts
- GenericDocuments
- SelectedObjects
- RefreshObject

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Programs

Used to navigate to the `INatAutoPrograms` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPrograms**) | Get only |

### Dialogs

Used to navigate to the `INatAutoDialogs` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDialogs**) | Get only |

### DataAreas

Used to navigate to the `INatAutoDataAreas` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoDataAreas**) | Get only |

### ObjectTrees

Used to navigate to the `INatAutoObjectTrees` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTrees**) | Get only |

### ObjectLists

Used to navigate to the `INatAutoObjectLists` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectLists**) | Get only |

### GenericTexts

Used to navigate to the `INatAutoGenericTexts` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericTexts**) | Get only |

### GenericDocuments

Used to navigate to the `INatAutoGenericDocuments` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoGenericDocuments**) | Get only |

### SelectedObjects

Used to navigate to the `INatAutoSelectedObjects` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSelectedObjects**) | Get only |

### RefreshObject

Used to navigate to the `INatAutoRefreshObject` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoRefreshObject**) | Get only |

## Methods

The following method is available:

- ActiveObject

### ActiveObject

Returns the currently active document window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH | |
| Class | A | VT_BSTR | By reference |

**Return value**

A handle to the currently active document window. This can be a handle to one of the following:

- INatAutoProgram
- INatAutoDialog
- INatAutoDataArea
- INatAutoGenericText
- INatAutoGenericDocument
- INatAutoObjectTree
- INatAutoObjectList

**Class**

A string that identifies the class of the document window. The string contains one of the following:

- INatAutoProgram
- INatAutoDialog
- INatAutoDataArea
- INatAutoGenericText
- INatAutoGenericDocument
- INatAutoObjectTree
- INatAutoObjectList

# 37 INatAutoObjectTree

## Purpose

An open tree view document window.

## Properties

The following properties are available:

- Parent
- Studio
- Profile
- PlugInID

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTrees**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Profile

A string containing an XML document that describes the behavior of this tree view document window. If the profile of a tree view is modified, the new settings apply from that time on. The already visible part of the tree is not redrawn according to the new profile settings. The XML document is structured according to the following DTD:

```
<!ELEMENT TreeViewProfile (Recursion?)>
<!ELEMENT Recursion (Detect?, Mark?, Expand?)>
<!ELEMENT Detect (#PCDATA)>
<!ELEMENT Mark   (#PCDATA)>
<!ELEMENT Expand (#PCDATA)>
```

| Element | Meaning | |
|---|---|---|
| Detect | Not specified or 0 | Recursion is detected by comparing type and key of two nodes. |
| | 1 | Recursion is detected by comparing only the key of two nodes. |
| | 2 | Recursion is detected by comparing type, key and info of two nodes. |
| Mark | Not specified | Recursive nodes are not marked. |
| | Specified | Recursive nodes are marked with the specified string. |
| Expand | Not specified or 0 | Recursive nodes cannot be further expanded. |
| | 1 | Recursive nodes can be further expanded only manually. They stay unexpanded during Expand All. |
| | 2 | Recursive nodes are further expanded even during Expand All. In this case the expansion can only be stopped with the ESC key. |

### PlugInID

The ID of the plug-in that created the tree view document window. A plug-in can use this property to iterate only across its own document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following methods are available:

- Close
- Cancel
- GetRootNode

- GetSelectedNode

### Close

Closes the tree view document window. This method is executed asynchronously. In particular, this means that if it is called from within the method `OnCommand` of a plug-in, the window will only really be closed after the method `OnCommand` has terminated.

### Cancel

Cancels an Expand All operation. This method has the same effect as pressing the ESC key.

### GetRootNode

Returns the root node of this MDI tree view.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTreeNode**) | |

**Return value**

The root node of this MDI tree view.

### GetSelectedNode

Returns the node currently selected in this MDI tree view.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTreeNode**) | |

**Return value**

The node currently selected in this MDI tree view.

## Notifications

A plug-in that has created a list view document window can expect to receive the following notifications through its method `OnNotify`:

- `PLUGIN-NOTIFY-ACTIVATE`
- `PLUGIN-NOTIFY-CLOSE`
- `PLUGIN-NOTIFY-EXPANDALL`
- `PLUGIN-NOTIFY-REFRESH`
- `PLUGIN-NOTIFY-HELP`

# 38 INatAutoObjectTreeNode

## Purpose

This interface represents a node in an MDI tree view. It contains methods to navigate through the nodes of a view, expand and collapse nodes and to access the development objects represented by the nodes.

## Properties

The following properties are available:

- Parent
- Studio
- IsExpanded
- IsSelected
- HasChildren

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | DISPATCH (**INatAutoObjectTree**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### IsExpanded

Indicates whether the node is expanded.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### IsSelected

Indicates whether the node is selected.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

### HasChildren

Indicates whether the node has child nodes.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

## Methods

The following methods are available:

- GetRoot
- GetParent
- GetChild
- GetNext
- GetPrevious
- GetObject
- Expand
- Collapse
- MakeVisible

- Select

### GetRoot

Returns the root node of the MDI tree view to which this node belongs.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTreeNode**) | |

**Return value**

The root node of the MDI tree view or list view to which this node belongs.

### GetParent

Returns the parent node of this node.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTreeNode**) | |

**Return value**

The parent node of this node.

### GetChild

Returns the first child node of this node.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTreeNode**) | |

**Return value**

The first child node of this node. If the node does not have children, NULL-HANDLE is returned.

### GetNext

Returns the next sibling node of this node.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (INatAutoObjectTreeNode) | |

**Return value**

The next sibling node of this node. If the node does not have a next sibling, NULL-HANDLE is returned.

### GetPrevious

Returns the previous sibling node of this node.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (INatAutoObjectTreeNode) | |

**Return value**

The previous sibling node of this node. If the node does not have a previous sibling, NULL-HANDLE is returned.

### GetObject

Returns the development object that this node represents.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (INatAutoSelectedObject) | |

**Return value**

The development object that this node represents.

### Expand

Expands the node.

### Collapse

Collapses the node.

### MakeVisible

Ensures that this node is in the visible part of the view. Scrolls the view as necessary.

### Select

Selects this node.

# 39 INatAutoObjectTrees

## Purpose

Collection of the currently open tree view document windows.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of currently open tree view document windows.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

# Methods

The following methods are available:

- Item
- Open

## Item

Returns a specific tree view document window from the collection.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTree**) | |
| Index | I4 | VT_I4 | |

**Return value**
The tree view document window identified by the value specified in Index.

**Index**
Identifies a specific tree view document window in the collection. This is a value between 1 and Count.

## Open

Opens a new tree view document window.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjectTree**) | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Caption | A | VT_BSTR | |
| Template | I4 | VT_I4 | Optional |
| PlugInID | A | VT_BSTR | Optional |
| Info | A | VT_BSTR | Optional |
| DisplayName | A | VT_BSTR | Optional |
| HasChildren | I4 | VT_I4 | Optional |
| Children | A | VT_BSTR | Optional |

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| NaturalType | I4 | VT_I4 | Optional |
| NaturalKey | A | VT_BSTR | Optional |

**Return value**
The newly opened tree view document.

**Type**
The node type of the root object. This must be a plug-in defined type that may correspond to a predefined Natural Studio type.

**Key**
The key that identifies the object within its type.

**Caption**
Determines the caption of the document window.

In order to generate a caption that matches the caption format used in Natural Studio document windows, the string passed in this parameter may contain the format specifier `%std`. This format specifier will be dynamically replaced by an identifier of the root object in the following way: If the root node corresponds to a predefined Natural Studio object, the format specifier `%std` is replaced by "*name* [*library*] -". Otherwise the identifier is built in the form "*node-type-caption display-name* -", where *node-type-caption* is the caption that was passed when the node type was created and *display-name* is the string passed in DisplayName.

**Template**
The value specified here is passed back to the plug-in when Natural Studio later calls the plug-in through the interface `INaturalStudioPlugInTree` in order to retrieve data to fill the view. This enables the plug-in to deliver different data for the same root object.

**PlugInID**
The ID of the plug-in that defined the type.

**Info**
Additional information that a plug-in may want to assign to the root object. Natural Studio does not interpret this information. It just passes it back to the plug-in when it later calls the plug-in through the interface `INaturalStudioPlugInTree` in order to retrieve data to fill the view.

**DisplayName**
The text to be displayed at the root node.

**HasChildren**
Indicates if the root node has child nodes and shall hence be displayed with a plus-sign as expandable. A value of 1 means that the root node has children, -1 that the root node has no children, 0 or not specified means that the plug-in does not know yet. Natural Studio will then retrieve this information from the plug-in in a later call through the interface `INaturalStudioPlugInTree`.

**Children**

If the plug-in already knows not only that the root node has child nodes, but also already knows the child nodes themselves, it can pass them in this parameter as a subtree of arbitrary depth. The parameter must then contain an XML document describing the child nodes. The XML document is structured according to the DTD defined in `INaturalStudioPlugInTree::GetChildren`. If this parameter is specified, Natural Studio implicitly assumes that the root node has child nodes, ignores what is specified in HasChildren and performs no call through the interface `INaturalStudioPlugInTree` to determine if the root node has child nodes.

**NaturalType**

If the root node corresponds to a predefined Natural Studio object, this property contains the type of the corresponding predefined object.

**NaturalKey**

If the root node corresponds to a predefined Natural Studio object, this property contains the key of the corresponding predefined object.

# 40 INatAutoPlugIn

## Purpose

An installed plug-in.

## Properties

The following properties are available:

- Parent
- Studio
- ID
- Type
- Name
- CLSID
- ProgID
- Active
- Automatic
- OptionValues

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPlugIns**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### ID

The global unique ID by which the plug-in is identified in Natural Studio. It is equal to the ID of the ActiveX component implementing the plug-in.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Type

The type of the plug-in.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

The possible values are:

| Value | Meaning |
|---|---|
| Single server | The plug-in runs in its own Natural server process, distinct from all other Natural-written plug-ins. |
| Shared server | The plug-in runs in the same Natural server process as the Plug-in Manager. |

### Name

The descriptive name of the plug-in that is displayed in the Plug-in Manager.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### CLSID

The CLSID of the ActiveX component implementing the plug-in. For a Natural-written plug-in this is the ID defined in the `DEFINE CLASS` statement.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### ProgID

The ProgID of the ActiveX component implementing the plug-in. For a Natural-written plug-in this is the class name defined in the `DEFINE CLASS` statement.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Active

Indicates if the plug-in is currently active. Modifying this property activates and deactivates the plug-in respectively.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### Automatic

Indicates if the plug-in is automatically activated when Natural Studio is started, or if it must be manually activated with the Plug-in Manager. Modifying this property changes the activation mode accordingly.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### OptionValues

A string containing an XML document that describes the current option value setting for this plug-in.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

The XML document is formatted according to the following DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT optvals (optval*)>
<!ELEMENT optval (name, value)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

| Element | Meaning |
|---|---|
| name | The name of the option. If this name does not correspond to a name defined in the `DefineOptions` method, this is a hidden option. This means that the user cannot see and modify the option value in the **Options** dialog. Otherwise the option is represented in a property page in the **Options** dialog as specified in the method `DefineOptions`. |
| value | The option value. |

## Methods

The following methods are available:

- DefineOptions
- GetCustomInterface
- OnCommand
- OnCommandStatus

### DefineOptions

Defines the options for this plug-in and their layout in a property page in the **Options** dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| OptionDefinition | A | VT_BSTR | |

**OptionDefinition**

A string containing an XML document that describes the options of the plug-in and their layout in a property page in the **Options** dialog. The XML document is formatted according to the following DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT optdef (caption, helpfile?, helptopic?, option*, groupbox*, statictext*)>
<!ELEMENT option (name, (checkbox | edittext | radiobuttons | spinbutton))>
<!ELEMENT checkbox (left, top, title, default)>
<!ELEMENT edittext (left, top, width, height, default)>
<!ELEMENT radiobuttons (radiobutton+, default)>
<!ELEMENT radiobutton (left, top, title, value)>
<!ELEMENT spinbutton (left, top, width, height, min, max, default)>
<!ELEMENT groupbox (title, left, top, width, height)>
<!ELEMENT statictext (title, left, top, width, height)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT helpfile (#PCDATA)>
<!ELEMENT helptopic (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

```
<!ELEMENT default (#PCDATA)>
<!ELEMENT left (#PCDATA)>
<!ELEMENT top (#PCDATA)>
<!ELEMENT width (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT min (#PCDATA)>
<!ELEMENT max (#PCDATA)>
```

| Element | Meaning |
|---|---|
| caption | The caption displayed on the property page. |
| helpfile | The full path name to a help file containing help information about the options. |
| helptopic | A help topic to be displayed when the user chooses the **Help** button on the property page. |
| name | The name of the option. This name is used only internally to refer to the option in the **OptionValues** DTD. |
| value | The value assigned to a radio button. |
| default | The default value of the option. |
| term | The term displayed with a dialog control. |
| left, top, width, height | The position and size of a dialog control. |
| min, max | The minimum and maximum value displayed in a spinbutton control. |

### GetCustomInterface

Returns the custom interface of this plug-in, or NULL-HANDLE, if the plug-in does not provide a custom interface.

Plug-ins can provide a custom interface in order to provide services to clients other than Natural Studio itself. These clients can be, for instance, other plug-ins or programs running inside or outside Natural Studio. In order to provide a custom interface, a plug-in must implement an additional interface beside the two predefined interfaces `INaturalStudioPlugIn` and `INaturalStudioPlugInTree` and make this interface the default dispatch interface. For a plug-in implemented in Natural this means placing this interface at the first position in the `DEFINE CLASS` statement.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH | |

**Return value**

The custom interface of this plug-in.

### OnCommand

Sends a specific command to the plug-in.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| CommandID | I4 | VT_I4 | |

**CommandID**

Contains the command ID the plug-in has chosen when it defined the command with the method `Add` of the interface `INatAutoCommands`.

### OnCommandStatus

Checks whether a specific command of the plug-in is currently enabled or checked.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| CommandID | I4 | VT_I4 | |
| Enabled | L | VT_BOOL | By reference |
| Checked | L | VT_BOOL | By reference |

**CommandID**

Contains the command ID the plug-in has chosen when it defined the command with the method `Add` of the interface `INatAutoCommands`.

**Enabled**

If the command is currently enabled, the method returns TRUE in this parameter.

**Checked**

If the command currently has a check mark, the method returns TRUE in this parameter.

# 41 INatAutoPlugIns

## Purpose

Collection of the currently installed plug-ins.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of installed plug-ins.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following method is available:

- Item

### Item

Returns a specific plug-in from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPlugIn**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The plug-in identified by the value specified in Index.

**Index**
Identifies a specific plug-in in the collection. This can be either the index of the plug-in in the collection (a value between 1 and Count) or the Plug-in ID (the value of the ID property of the corresponding `INatAutoPlugIn` object).

# 42 INatAutoPopupMenu

## Purpose

Gives access to a specific pop-up menu within a context menu, frame menu or pop-up menu.

Changes in a menu are not made visible immediately, in order to avoid flickering. After having finished modifying a menu, make the recent changes visible by calling the method `UpdateMenu` of the context menu or frame menu that contains this pop-up menu.

## Properties

The following properties are available:

- Studio
- Count
- Caption

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of items (commands, separators and pop-up menus) in the menu.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Caption

A string used to identify the menu, as defined when the menu was created.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

## Methods

The following methods are available:

- Item
- SubMenu
- InsertCommand
- InsertSeparator
- InsertPopupMenu

### Item

Returns a specific item contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**
The caption of the menu item (command or pop-up menu) identified by the value specified in Index. If the index identifies a separator, an empty string is returned.

**Index**
The index of the item in the menu (a value between 1 and Count).

### SubMenu

Returns a specific pop-up menu contained in the menu, based on an index.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |

**Return value**

The pop-up menu identified by the value specified in Index. If the specified index does not identify a pop-up menu, but a command or a separator, a null interface pointer (NULL-HANDLE) is returned.

**Index**

As index either a number between 1 and Count or the caption of a pop-up menu can be specified.

### InsertCommand

Inserts a command into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Command | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| Index | I4 | VT_I4 | Optional |

**Command**

A command to be added to the menu. The command must have been defined before using the method `INatAutoCommands::Add`.

**Index**

The position in the menu where the command shall be inserted. If Index is omitted, the command is inserted at the last position.

### InsertSeparator

Inserts a separator into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Index | I4 | VT_I4 | Optional |

**Index**

The position in the menu where the separator shall be inserted. If Index is omitted, the separator is inserted at the last position.

### InsertPopupMenu

Creates a new pop-up menu and inserts it into the menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPopupMenu**) | |
| Caption | A | VT_BSTR | |
| Index | I4 | VT_I4 | Optional |

**Return value**

The newly created pop-up menu.

**Caption**

A string used to identify the pop-up menu.

**Index**

The position in the menu where the pop-up menu shall be inserted. If Index is omitted, the pop-up menu is inserted at the last position.

# 43 INatAutoProgram

## Purpose

A development object currently open in a program editor window. This comprises the following development object types: program, subprogram, subroutine, function, helproutine, copycode, text and class. The types are identified by a numeric ID. The IDs of predefined types are described in the section ***Predefined Node Types***.

## Properties

The following properties are available:

- Parent
- Studio
- Source
- Visible
- Type
- LineCount

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPrograms**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Source

The source code of the development object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | |

### Visible

Shows or hides the editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

### Type

The development object type. The type is identified by a numeric ID. The IDs of predefined types are described in the section *Predefined Node Types*.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### LineCount

The number of lines in the source code.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Catalog
- Check
- Clear
- Close
- Execute
- Format
- Mode
- Renumber

- Search
- Replace
- Run
- Save
- Stow
- Title
- GetInfo
- DeleteLines
- GetLines
- InsertLines
- ReplaceLines

## Catalog

Catalogs the object. Applicable to program, subprogram, subroutine, function, helproutine and class.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

## Check

Checks the object. Applicable to program, subprogram, subroutine, function, helproutine and class.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Clear

Clears the editor contents.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Close

Closes the editor and removes the object from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Execute

Executes the object. Applicable to program.

### Format

Formats the source code.

### Mode

Sets several modes of the object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Structured | L | VT_BOOL | Optional |
| Uppercase | L | VT_BOOL | Optional |
| IgnoreTextConstants | L | VT_BOOL | Optional |

**Structured**
Sets structured mode. The default is determined by the Natural parameter settings.

**Uppercase**
Sets uppercase mode. The source code will then be converted to upper case during Save. The default is FALSE.

**IgnoreTextConstants**
Makes sure that text constants are left untouched during upper case conversion. The default is FALSE.

### Renumber

Renumbers the source code.

### Search

Searches for the first occurrence of a given string.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| SearchString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |

**Return value**
TRUE if a match was found.

**SearchString**
The string to search for.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction. The default is FALSE.

### Replace

Replaces the first occurrence of a given string with another one.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| SearchString | A | VT_BSTR | |
| Line | I4 | VT_I4 | By reference |
| Column | I4 | VT_I4 | By reference |
| ReplaceString | A | VT_BSTR | |
| CaseSensitive | L | VT_BOOL | Optional |
| WholeWords | L | VT_BOOL | Optional |
| Up | L | VT_BOOL | Optional |
| Return value | BOOL | VT_BOOL | |

**Note:** Specify the parameters in the sequence as listed in the table.

**SearchString**
The string to search for.

**Line**
Contains the start line for the search on input. Contains the line of the first match on return.

**Column**
Contains the start column for the search on input. Contains the column of the first match on return.

**ReplaceString**
The string which replaces the search string.

**CaseSensitive**
Searches case sensitively. The default is FALSE.

**WholeWords**
Searches only for whole words that match the search string. The default is FALSE.

**Up**
Searches in upward direction. The default is FALSE.

**Return value**
TRUE if a match was found.

### Run

Runs the object. Applicable to program.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Quiet | L | VT_BOOL | Optional |

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Save

Saves the object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Type | I4 | VT_I4 | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Saves the object under the given name.

**Library**
Saves the object in the given library.

**Type**
Saves the object under the given type.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Stow

Stows the object. Applicable to program, subprogram, subroutine, function, helproutine and class.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |
| Type | I4 | VT_I4 | Optional |
| Quiet | L | VT_BOOL | Optional |

**Name**
Stows the object under the given name.

**Library**
Stows the object in the given library.

**Type**
Stows the object under the given type.

**Quiet**
If set to TRUE, the method is performed without user interaction. The default is FALSE.

### Title

Titles an untitled object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Name | A | VT_BSTR | Optional |
| Library | A | VT_BSTR | Optional |

**Name**
Assigns a name to the object.

**Library**
Assigns a library to the object.

### GetInfo

Returns information about an open object.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Type | I4 | VT_I4 | By reference |
| Name | A | VT_BSTR | By reference |
| Library | A | VT_BSTR | By reference |
| Fnr | I4 | VT_I4 | By reference |
| DBnr | I4 | VT_I4 | By reference |

**Type**
The type of the object.

**Name**
The name of the object.

**Library**
The library of the object.

**Fnr**
The system file file number of the object.

**DBnr**
The system file database number of the object.

### DeleteLines

Deletes a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**StartLine**
The start line of the block to delete.

**LineCount**
The number of lines to delete. The default is 1.

### GetLines

Retrieves a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| StartLine | I4 | VT_I4 | |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
A block of source code lines. The lines are separated by carriage return / line feed characters.

**StartLine**
The start line of the block to return.

**LineCount**
The number of lines to return. The default is 1.

### InsertLines

Inserts a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| InsertAfterLine | I4 | VT_I4 | Optional |

**Return value**
The line number passed in InsertAfterLine increased by the number of inserted lines.

**Code**
A block of source code lines to insert. The lines must be separated by carriage return / line feed characters.

**InsertAfterLine**
Line after which the code shall be inserted. The default is 0.

### ReplaceLines

Replaces a block of lines from the source code.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Code | A | VT_BSTR | |
| ReplaceLine | I4 | VT_I4 | Optional |
| LineCount | I4 | VT_I4 | Optional |

**Return value**
The line number passed in ReplaceLine increased by the number of inserted lines.

**Code**
A block of source code lines to replace the block that is defined by ReplaceLine and LineCount. The lines must be separated by carriage return / line feed characters.

**ReplaceLine**
The start line of the block to be replaced. The default is 1.

**LineCount**
The number of lines to be replaced by the given block. The default is 1.

# 44 INatAutoPrograms

## Purpose

Collection of the development objects currently open in a program editor window. This collection comprises the following development object types: program, subprogram, subroutine, function, helproutine, copycode, text and class. The types are identified by a numeric ID. The IDs of pre-defined types are defined in the section *Predefined Node Types*.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of development objects currently open in a program editor window.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Add
- Open

### Item

Returns a specific development object from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoProgram**) | |
| Index | I4<br>A | VT_I4<br>VT_BSTR | |
| Type | I4 | VT_I4 | Optional |
| Library | A | VT_BSTR | Optional |
| Fnr | I4 | VT_I4 | Optional |
| DBnr | I4 | VT_I4 | Optional |

**Return value**
The development object identified by the value specified in Index.

**Index**
Identifies a specific development object in the collection. This can be either the index of the development object in the collection (a value between 1 and Count) or the name of the object.

**Type**
Used to identify a specific object by name (specified in Index) and type (specified in Type).

**Library**
Used to identify a specific object by name (specified in Index), type (specified in Type) and library (specified in Library).

**Fnr, DBnr**
Used to identify a specific object by name (specified in Index), type (specified in Type), library (specified in Library) and system file (specified in Fnr and DBnr).

### Add

Creates a new (untitled) development object and opens it in a program editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoProgram**) | |
| Type | I4 | VT_I4 | |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly created development object.

**Type**
The type of object to create.

**Visible**
Decides if the editor is opened visible or not. By default, the editor is opened visible.

### Open

Opens an existing development object in a program editor window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoProgram**) | |
| Type | I4 | VT_I4 | |
| Name | A | VT_BSTR | |
| Library | A | VT_BSTR | Optional |
| Visible | L | VT_BOOL | Optional |
| ReadOnly | L | VT_BOOL | Optional |

**Return value**
The newly opened development object.

**Type**
The type of object to open.

**Name**
The name of object to open.

**Library**
The library of object to open.

**Visible**

Decides whether the editor is opened visible or not. By default, the editor is opened visible.

**ReadOnly**

Decides whether the object is listed only instead of opened. In this case, the object is not locked and cannot be modified. If the option ReadOnly is specified, also the types dialog, local data area, parameter data area and global data area can be specified in the parameter Type. This is the case because Natural Studio lists also these object types in the program editor.

# 45 INatAutoProgressIndicator

## Purpose

A progress indicator is used to inform the user about the progress of a time consuming operation. A plug-in can create a progress indicator with the method `INatAutoStudio::ProgressIndicator`.

## Properties

The following properties are available:

- Parent
- Studio
- StatusBarText
- GradientBarText
- DialogText
- Canceled

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### StatusBarText

The text to be displayed in the status bar. This property is used with progress indicators of style status bar and gradient bar.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Put only |

### GradientBarText

The text to be displayed in the gradient bar. This property is used with progress indicators of style gradient bar.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Put only |

### DialogText

The text to be displayed in the animated dialog. This property is used with progress indicators of style Dialog.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Put only |

### Canceled

Indicates if the user has pressed the ESC key or (in case of a progress indicator of style Dialog) the **Cancel** button, in order to abort the operation.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

## Methods

The following methods are available:

- Start
- Step
- Terminate
- StopAnimation

- PlayAnimation

### Start

Starts the progress indicator.

### Step

Advances the progress indicator.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Steps | I4 | VT_I4 | |

**Steps**
The number of steps to advance.

### Terminate

Terminates the progress indicator.

### StopAnimation

Stops running the animation that was assigned to the progress indicator when it was created. The animation can be resumed again by calling PlayAnimation.

### PlayAnimation

Continues running the animation that was assigned to the progress indicator when it was created. The animation can be stopped by calling StopAnimation.

# 46 INatAutoRefreshObject

## Purpose

The object currently being refreshed. This interface is available during a Refresh operation. While handling the notification `PLUGIN-NOTIFY-REFRESH`, a plug-in can use this interface to retrieve the details about the object currently being refreshed.

## Properties

The following properties are available:

- Parent
- Studio
- PlugInID
- Type
- Key
- Info
- NaturalType
- NaturalKey
- NaturalName
- Environment
- Application
- Current

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### PlugInID

The ID of the plug-in that defined the type. Not filled for objects of predefined types.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Type

The node type of the object. This can either be a predefined type or a user defined type.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Key

The key that identifies the object within its type.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Info

Additional information that a plug-in may have assigned to the object. Not filled for objects of predefined types.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### NaturalType

If the object has been defined by a plug-in, but corresponds to an object of a predefined Natural Studio node type, this property contains the type of the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### NaturalKey

If the object has been defined by a plug-in, but corresponds to an object of a predefined Natural Studio node type, this property contains the key of the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### NaturalName

If the object has been defined by a plug-in, but corresponds to an object of the predefined Natural Studio node type subroutine, function or class, this property contains the function name or class nameof the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Environment

The environment the object belongs to. If the object belongs to the currently active environment or to an application, the value is NULL-HANDLE.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### Application

The application the object belongs to. If the object belongs to the currently active application or to no application at all, the value is NULL-HANDLE.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | Get only |

### Current

True, if the object belongs to the current environment or application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

# 47 INatAutoResultView

## Purpose

An open result view.

## Properties

The following properties are available:

- Parent
- Studio
- Active

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoResultViews**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Active

Indicates if this result view is currently the active one or not. Setting Active to TRUE makes the result view the active one. Setting Active to FALSE has no effect.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

## Methods

The following methods are available:

- InsertRows
- Update
- SetVisible
- Clear
- Close

### InsertRows

Inserts a number of rows at the end of the result view. Each row is displayed in the result view as a node with attributes. Before referring to a node type in this method, the plug-in must have registered the node type through the interface `INatAutoNodeTypes`.

To avoid flickering, the result view is not redrawn after each call to `InsertRows`. After a series of calls to `InsertRows` the method `Update` should be called to redraw the result view.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4) | |
| Rows | A | VT_BSTR | |

**Return value**
The number of rows contained in the result view after the insertion.

**Rows**
Contains an XML document that describes the rows to be inserted. The XML document is structured according to the following DTD.

```
<!ELEMENT rows (row*)>
<!ELEMENT row (pitem, attributevalues?)>
<!ELEMENT pitem (pguid?, ptype, pkey, pinfo?, pname?, (ntype, nkey)?)>
<!ELEMENT pguid (#PCDATA)>
<!ELEMENT ptype (#PCDATA)>
<!ELEMENT pkey  (#PCDATA)>
<!ELEMENT pinfo (#PCDATA)>
<!ELEMENT pname (#PCDATA)>
<!ELEMENT ntype (#PCDATA)>
<!ELEMENT nkey  (#PCDATA)>
<!ELEMENT attributevalues (attval*)>
<!ELEMENT attval     (akey, avalue)>
```

```
<!ELEMENT akey  (#PCDATA)>
<!ELEMENT avalue     (#PCDATA)>
```

| Element | Meaning |
|---|---|
| pctype | The type of the node to be inserted. |
| pkey | The key that identifies the node within its type. |
| pguid | Needs not to be filled if the node type has been defined by the plug-in itself. It is used to refer to node types of other plug-ins, if these are known. |
| pinfo | Additional information about the node that the plug-in wants to receive back whenever Natural Studio later refers to the node. Natural Studio never considers the content of this element, put just passes it back and forth. |
| pname | The text to be displayed with the node. |
| ntype | If the node has been defined by a plug-in, but corresponds to a node of a predefined Natural Studio type, this element contains the type of the corresponding predefined node. |
| nkey | If the node has been defined by a plug-in, but corresponds to a node of a predefined Natural node type, this element contains the key of the corresponding predefined node. |
| attributevalues | The attribute values to be displayed with the node. These are specified as an XML document according to the DTD used with the method `INaturalStudioPlugInTree::GetAttributeValues`. |

### Update

To avoid flickering, the result view is not redrawn after each call to `InsertRows`. After a series of calls to `InsertRows` the method `Update` should be called to redraw the result view.

### SetVisible

Makes the specified row visible and scrolls the result view if necessary.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Row | I4 | VT_I4 | |

**Row**

Contains the number of the row to be scrolled into view. To position to the last inserted row, use the row number that was returned from the method `InsertRows`.

### Clear

Removes all rows from the result view.

### Close

Closes the result view.

# 48 INatAutoResultViews

## Purpose

Collection of the currently open result views.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of currently open result views.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- Show
- Open

### Item

Returns a specific plug-in defined result view from the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoResultView**) | |
| Index | I4 | VT_I4 | |

**Return value**
The result view identified by the value specified in Index. If the result view was not defined by a plug-in, the method returns NULL-HANDLE.

**Index**
The index of the result view in the collection (a value between 1 and Count).

### Show

Shows or hides the entire result view control bar. This corresponds to checking or unchecking the View/Results command in the Natural Studio menu.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Show | L | VT_BOOL | Optional |

**Show**
Decides if the result view control bar is shown or not. The default is TRUE.

### Open

Opens a new result view.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoResultView**) | |
| Caption | A | VT_BSTR | |
| Image | A<br>HANDLE OF OBJECT | VT_BSTR<br>VT_DISPATCH | Optional |
| Headers | A | VT_BSTR | Optional |

**Return value**
The newly opened result view.

**Caption**
The caption to be displayed on the result view tab.

**Image**
An image to be displayed on the result view tab. The image must be a 16 color, 16x16 bitmap, using RGB(192,192,192) as the background color.

The image can be specified in two ways:

- As an absolute path name of a .bmp file.
- As an `IPictureDisp` interface. An `IPictureDisp` interface can be created in Natural using the method `INatAutoImages::LoadImage`. An `IPictureDisp` interface cannot be passed across process boundaries. This is due to a Microsoft restriction (MSDN Q150034). Therefore this alternative can only be used with plug-ins running as in-process servers. Natural written plug-ins always run as local servers and can therefore not use this alternative.

**Headers**
Defines the attributes of the nodes to be displayed in the result view and their respective captions. It contains the attribute definitions as an XML document according to the DTD used with the method `INaturalStudioPlugInTree::GetAttributes`.

# 49 INatAutoSelectedObject

## Purpose

A currently selected object.

## Properties

The following properties are available:

- Parent
- Studio
- PlugInID
- Type
- Key
- Info
- NaturalType
- NaturalKey
- NaturalName
- Environment
- Application
- Current

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSelectedObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### PlugInID

The ID of the plug-in that defined the type. Not filled for objects of predefined types.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Type

The node type of the object. This can either be a predefined type or a user-defined type.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### Key

The key that identifies the object within its type.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Info

Additional information that a plug-in may have assigned to the object. Not filled for objects of predefined types.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### NaturalType

If the object has been defined by a plug-in, but corresponds to an object of a predefined Natural Studio node type, this property contains the type of the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### NaturalKey

If the object has been defined by a plug-in, but corresponds to an object of a predefined Natural Studio node type, this property contains the key of the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### NaturalName

If the object has been defined by a plug-in, but corresponds to an object of the predefined Natural Studio node type subroutine, function or class, this property contains the function name or class name of the corresponding predefined object.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Environment

The environment the object belongs to. If the object belongs to the currently active environment or to an application, the value is NULL-HANDLE.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Get only |

### Application

The application the object belongs to. If the object belongs to the currently active application or to no application at all, the value is NULL-HANDLE.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplication**) | Get only |

### Current

True, if the object belongs to the current environment or application.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Get only |

# 50 INatAutoSelectedObjects

## Purpose

Collection of the Natural Studio objects that the user has currently selected. Each object is contained only once in the collection, even if the user has selected several visualizations of the same object.

The collection of selected objects can be processed in either of two ways:

- Through the property `SelectedObjects` the selected objects can be retrieved as an XML document. This document can then be processed with XML processing functions and statements.
- The `Item` method can be used to iterate across the collection in the usual way.

## Properties

The following properties are available:



### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of selected objects.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

### SelectedObjects

A string containing an XML document that describes the selected objects. The XML document is structured according to the following DTD:

```
<?xml version="1.0"?>
<!ELEMENT itemset (sitem*)>
<!ELEMENT sitem(sguid?, stype, skey, sinfo?,
(ntype, nkey, nname?, (nenv | napp)?)?)>
<!ELEMENT sguid (#PCDATA)>
<!ELEMENT stype (#PCDATA)>
<!ELEMENT skey  (#PCDATA)>
<!ELEMENT sinfo (#PCDATA)>
<!ELEMENT ntype (#PCDATA)>
<!ELEMENT nkey  (#PCDATA)>
<!ELEMENT nname (#PCDATA)>
<!ELEMENT nenv  (#PCDATA)>
<!ELEMENT napp  (#PCDATA)>
```

| Element | Meaning |
|---|---|
| sguid | The ID of the plug-in that defined the type. Not filled for objects of predefined types. |
| stype | The node type of the object. This can either be a predefined type or a user defined type. |
| skey | The key that identifies the node within its type. |
| sinfo | Additional information that a plug-in may have assigned to the object. Not filled for objects of predefined types. |
| ntype | If the object has been defined by a plug-in, but corresponds to an object of a predefined Natural Studio node type, this property contains the type of the corresponding predefined object. |
| nkey | If the node has been defined by a plug-in, but corresponds to a node of a predefined Natural node type, this element contains the key of the corresponding predefined node. |
| nname | If the object has been defined by a plug-in, but corresponds to an object of the predefined Natural Studio node type subroutine, function or class, this property contains the function name or class name of the corresponding predefined object. |
| nenv | Key of the environment the object belongs to. The key can be used to access the corresponding environment using the method `INatAutoEnvironments::Item`. If the object belongs to the currently active environment or to an application, the element is empty. |
| napp | Key of the application the object belongs to. The key can be used to access the corresponding application using the method `INatAutoApplications::Item`. If the object belongs to the currently active application or to no application at all, the element is empty. |

### Parameters

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### FocusObject

The index of the object that currently has the focus.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Item
- ContainsObjectType

### Item

Returns a specific selected object from the collection.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSelectedObject**) | |
| Index | I4 | VT_I4 | |

**Return value**
The selected object identified by the value specified in Index.

**Index**
The index of the selected object in the collection (a value between 1 and Count).

### ContainsObjectType

Checks if the current selection contains at least one object of a given type. This quick check is often sufficient to decide if a specific command is applicable to the current selection, without iterating explicitly across the selected objects.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | BOOL | VT_BOOL | |
| Type | I4 | VT_I4 | |
| PlugInID | A | VT_BSTR | Optional |

**Return value**
TRUE, if the current selection contains at least one object of the given type.

**Type**
A predefined or user defined node type.

**PlugInID**
The global unique ID of the plug-in that defined the type. This is the value of the plug-in's ID property.

# 51 INatAutoStudio

## Purpose

The root interface of the Natural Studio Automation interface. A handle to this interface is passed to each plug-in during activation (`INaturalStudioPlugIn::OnActivate`) and deactivation (`INaturalStudioPlugIn::OnDeactivate`).

## Properties

The following properties are available:

- Objects
- ControlBars
- Types
- PlugIns
- ResultViews
- System

### Objects

Used to navigate to the `INatAutoObjects` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoObjects**) | Get only |

### ControlBars

Used to navigate to the `INatAutoControlBars` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Types

Used to navigate to the `INatAutoTypes` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoTypes**) | Get only |

### PlugIns

Used to navigate to the `INatAutoPlugIns` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoPlugIns**) | Get only |

### ResultViews

Used to navigate to the `INatAutoResultViews` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoResultViews**) | Get only |

### System

Used to navigate to the `INatAutoSystem` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSystem**) | Get only |

## Methods

The following methods are available:

- Refresh
- MessageBox
- ShowHelp

- ProgressIndicator

### Refresh

Initiates an automatic refresh in Natural Studio.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| RefreshObject | A | VT_BSTR | Optional |

**RefreshObject**

This parameter is either not specified (unspecific refresh) or contains the refresh object formatted as an XML document (specific refresh). The object is formatted according to the following DTD. The meaning of the individual elements is analog to the DTD describing `INatAutoSelectedObjects::SelectedObjects`.

```
<!ELEMENT ritem (rguid?, rtype, rkey, rinfo?, (ntype, nkey, nname?)?)>
<!ELEMENT rguid (#PCDATA)>
<!ELEMENT rtype (#PCDATA)>
<!ELEMENT rkey  (#PCDATA)>
<!ELEMENT rinfo (#PCDATA)>
<!ELEMENT ntype (#PCDATA)>
<!ELEMENT nkey  (#PCDATA)>
<!ELEMENT nname (#PCDATA)>
```

### MessageBox

Displays a standard message box in Natural Studio.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Text | A | VT_BSTR | |
| Caption | A | VT_BSTR | Optional |
| Style | I4 | VT_I4 | Optional |

**Return value**

Indicates the button that the user pressed in response to the message box.

1: OK
2: CANCEL
3: ABORT

4: RETRY
5: IGNORE
6: YES
7: NO

**Text**
The text to be displayed in the message box.

**Caption**
The caption of the message box.

**Style**
The message box style is specified by adding one of the following styles

0: OK
1: OKCANCEL
2: ABORTRETRYIGNORE
3: YESNOCANCEL
4: YESNO
5: RETRYCANCEL

to one of the following styles:

16: MB_ICONHAND
32: MB_ICONQUESTION
48: MB_ICONEXCLAMATION
64: MB_ICONASTERISK

The default is 0.

### ShowHelp

Displays a specific help topic in a specific help file.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | L | VT_BOOL | |
| HelpTopic | I4 | VT_I4 | Optional |
| HelpFile | A | VT_BSTR | Optional |

**Return value**
If the specified help file or topic is not found, FALSE is returned, otherwise TRUE.

**HelpTopic**
A topic in the specified help file. If HelpTopic is omitted, the contents page of the help file is displayed.

**HelpFile**
A help file specified with full path name. If HelpFile is omitted, the specified topic in the Natural Studio help file is displayed. If both HelpTopic and HelpFile are omitted, the contents page of the Natural Studio help file is displayed.

### ProgressIndicator

Creates and returns a new progress indicator.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoProgressIndicator**) | |
| Steps | I4 | VT_I4 | |
| Style | I4 | VT_I4 | Optional |
| Frequency | I4 | VT_I4 | Optional |
| Caption | A | VT_BSTR | Optional |
| Animation | A | VT_BSTR | Optional |

**Return value**
The newly created progress indicator.

**Steps**
The number of steps this by which the progress indicator can be advanced.

**Style**
The style of the progress indicator. Possible values are:

| | |
|---|---|
| 0 | Status bar. The progress of the operation is displayed as a text in the status bar. |
| 1 | Gradient bar. The progress of the operation is displayed as a percentually growing gradient bar. Optionally an additional text can be displayed in the status bar. |
| 2 | Dialog. The progress of the operation is displayed as a dialog box containing an animation. |

**Frequency**
By default, the progress indicator is redrawn after each step. If Frequency is specified, it is re-drawn only each Steps/Frequency steps. This can be used to avoid flickering.

**Caption**
Applicable with progress indicators of style Dialog. The caption to display in the dialog.

**Animation**
Applicable with progress indicators of style Dialog. Path and file name of an animation file (.avi) to display in the dialog.

# 52 INatAutoSysmain

## Purpose

This interface contains methods related to the utility `SYSMAIN`. These methods include copying and moving Natural development objects between system files and environments, importing files as Natural development objects into a Natural system file and deleting and renaming Natural development objects.

Using properties, each instance of this interface can be configured independently of other instances. The properties define on which environments and system files the subsequently called methods will work. The properties control also certain options that influence the behavior of the subsequently called methods.

## Properties

The following properties are available:

- Parent
- Studio
- SourceEnvironment
- SourceDBnr
- SourceFnr
- TargetEnvironment
- TargetDBnr
- TargetFnr
- OptionType
- OptionTimestamp
- OptionUser
- OptionImportSM

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSystem**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### SourceEnvironment

Specifies the source environment for the subsequent operations. Default is the local environment. If the property is changed to a different environment, the properties `SourceDBnr` and `SourceFnr` are automatically changed to the database number and file number of the user system file of that environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Put only |

### SourceDBnr

Specifies the database number of the source system file for the subsequent operations. Default is the database number of the user system file of the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### SourceFnr

Specifies the file number of the source system file for the subsequent operations. Default is the file number of the user system file of the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### TargetEnvironment

Specifies the target environment of the subsequent operations. Default is the local environment. If the property is changed to a different environment, the properties `TargetDBnr` and `TargetFnr` are automatically changed to the database number and file number of the user system file of that environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironment**) | Put only |

### TargetDBnr

Specifies the database number of the target system file for the subsequent operations. Default is the database number of the user system file of the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### TargetFnr

Specifies the file number of the target system file for the subsequent operations. Default is the file number of the user system file of the local environment.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

### OptionType

Specifies the Natural development object type on which the subsequent operation applies.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | |

The supported types are listed below. The default is 0.

| Type Number | Type Name |
|---|---|
| 0000 | All types |
| 1001 | Parameter data area |
| 1002 | Copycode |
| 1003 | DDM |
| 1004 | Global data area |

| Type Number | Type Name |
|---|---|
| 1005 | Helproutine |
| 1006 | Local data area |
| 1007 | Map |
| 1008 | Subprogram |
| 1009 | Program |
| 1010 | Subroutine |
| 1011 | Text |
| 1012 | View |
| 1013 | Dialog |
| 1014 | Class |
| 1015 | Command processor |
| 1017 | Mainframe DDM |
| 1018 | Function |
| 1019 | Shared resource |
| 1020 | Error message file |
| 1021 | Adapter |

### OptionTimestamp

The subsequent operations apply to Natural development objects that have been saved or cataloged after the point in time specified in this option. The default is the lowest possible value of a Natural variable of format T.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| T | VT_DATE | Put only |

### OptionUser

The subsequent operations apply to Natural development objects that have been saved or cataloged by the specified user. The default is an empty string.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Put only |

### OptionImportSM

If this option is set to TRUE, files to be imported as Natural development objects with the method `Import` are assumed to be in structured mode. If this turns out to be not the case, the import will fail with an error.

If this option is set to FALSE, files to be imported as Natural development objects with the method `Import` are assumed to be in report mode. If this turns out to be not the case, the import will fail with an error.

The default is the value that is specified for the Natural parameter `SM`.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | Put only |

## Methods

The following methods are available:

- Reset
- FindLibraries
- Find
- Copy
- Move
- Delete
- Rename
- Import

### Reset

Resets all properties of this interface instance to their default values.

### FindLibraries

Returns a string containing an XML document that describes the Natural libraries contained in the Natural system file specified by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr`. The XML document is structured according to the following DTD:

```
<?xml version="1.0"?>
<!ELEMENT flibs (flib+)>
<!ELEMENT flib (#PCDATA)>
```

| Element | Meaning |
|---|---|
| flib | The library name. |

## Find

Returns a string containing an XML document that describes the Natural development objects contained in the Natural library specified in the parameter `Library` and by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr`. The XML document is structured according to the following DTD:

```
<?xml version="1.0"?>
<!ELEMENT fitems (fitem+)>
<!ELEMENT fitem (ftype, fkey, fname, fcat, fuid)>
<!ELEMENT ftype (#PCDATA)>
<!ELEMENT fkey (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT fcat (#PCDATA)>
<!ELEMENT fuid (#PCDATA)>
```

| Element | Meaning |
|---|---|
| ftype | The object type. See the list of types that is available for the property `OptionType`. |
| fkey | The object name. |
| fname | For Natural classes: the class name. For Natural subroutines: the subroutine name. For Natural functions: the function name. |
| fcat | Indicates if a source, a generated program or both exists for the object. See the values defined for the parameter `Category`. |
| fuid | The user ID of the user who saved or cataloged the object. |

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Name | A | VT_BSTR | By value |
| Library | A | VT_BSTR | By value |
| Category | I2 | VT_I2 | By value |

**Return value**

A null BSTR (in Natural an empty string). Reserved for future use.

**Name**
A pattern that qualifies the names of the Natural development objects to be retrieved. The pattern may contain the wildcard characters "?" and "*", where "?" stands for one character and "*" for several characters.

**Library**
The name of the Natural library from which Natural development objects shall be retrieved.

**Category**
Specifies whether sources or generated programs shall be retrieved.

| Value | Meaning |
|---|---|
| 0 | Natural objects where either a source or a generated program exists are retrieved. |
| 1 | Only Natural objects where a source exists are retrieved. |
| 2 | Only Natural objects where a generated program exists are retrieved. |
| 3 | Only Natural objects where both a source and a generated program exist are retrieved. |

## Copy

Copies Natural development objects from the library specified by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr` and the parameter `SourceLibrary` to the library specified by the properties `TargetEnvironment`, `TargetDBnr` and `TargetFnr` and the parameter `TargetLibrary`.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Name | A | VT_BSTR | By value |
| SourceLibrary | A | VT_BSTR | By value |
| TargetLibrary | A | VT_BSTR | By value |
| Category | I2 | VT_I2 | By value |

**Return value**
A null BSTR (in Natural an empty string). Reserved for future use.

**Name**
A pattern that qualifies the names of the Natural development objects to be copied. The pattern may contain the wildcard characters "?" and "*", where "?" stands for one character and "*" for several characters.

**SourceLibrary**
The name of the Natural library from which Natural development objects shall be copied.

**TargetLibrary**
The name of the Natural library to which Natural development objects shall be copied.

**Category**
Specifies whether sources, generated programs or both shall be copied. For possible values, see the `Find` method.

### Move

Moves Natural development objects from the library specified by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr` and the parameter `SourceLibrary` to the library specified by the properties `TargetEnvironment`, `TargetDBnr` and `TargetFnr` and the parameter `TargetLibrary`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Name | A | VT_BSTR | By value |
| SourceLibrary | A | VT_BSTR | By value |
| TargetLibrary | A | VT_BSTR | By value |
| Category | I2 | VT_I2 | By value |

**Return value**
A null BSTR (in Natural an empty string). Reserved for future use.

**Name**
A pattern that qualifies the names of the Natural development objects to be moved. The pattern may contain the wildcard characters "?" and "*", where "?" stands for one character and "*" for several characters.

**SourceLibrary**
The name of the Natural library from which Natural development objects shall be moved.

**TargetLibrary**
The name of the Natural library to which Natural development objects shall be moved.

**Category**
Specifies whether sources, generated programs or both shall be moved. For possible values, see the `Find` method.

### Delete

Deletes Natural development objects from the library specified by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr` and the parameter `Library`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Name | A | VT_BSTR | By value |
| Library | A | VT_BSTR | By value |
| Category | I2 | VT_I2 | By value |

**Return value**
A null BSTR (in Natural an empty string). Reserved for future use.

**Name**
A pattern that qualifies the names of the Natural development objects to be deleted. The pattern may contain the wildcard characters "?" and "*", where "?" stands for one character and "*" for several characters.

**Library**
The name of the Natural library from which Natural development objects shall be deleted.

**Category**
Specifies whether sources, generated programs or both shall be deleted. For possible values, see the `Find` method.

### Rename

Renames the Natural development object specified by the properties `SourceEnvironment`, `SourceDBnr` and `SourceFnr` and the parameters `Name` and `Library` to the name specified by the parameter `NewName`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| Name | A | VT_BSTR | By value |
| NewName | A | VT_BSTR | By value |
| Library | A | VT_BSTR | By value |
| Category | I2 | VT_I2 | By value |

**Return value**
A null BSTR (in Natural an empty string). Reserved for future use.

**Name**
The name of the Natural development object to be renamed.

**NewName**
The new name for the Natural development object.

**Library**
The name of the Natural library that contains the Natural development object to be renamed.

**Category**
Specifies whether the source, the generated program or both shall be renamed. For possible values, see the `Find` method.

### Import

Imports the files specified by the parameters `File` and `Path` as Natural development objects into the library specified by the properties `TargetEnvironment`, `TargetDBnr` and `TargetFnr` and the parameter `Library`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |
| File | A | VT_BSTR | By value |
| Path | A | VT_BSTR | By value |
| Library | A | VT_BSTR | By value |

**Return value**
A null BSTR (in Natural an empty string). Reserved for future use.

**File**
A pattern that qualifies the names of the files to be imported. The pattern may contain the wildcard characters "?" and "*", where "?" stands for one character and "*" for several characters.

**Path**
The path that contains the files to be imported.

**Library**
The name of the Natural library into which the files shall be imported.

# 53 INatAutoSystem

## Purpose

Gives access to certain system functions and to the available development environments and applications.

## Properties

The following properties are available:

- Parent
- Studio
- Environments
- Applications
- Sysmain

### Parent

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Environments

Used to navigate to the `INatAutoEnvironments` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoEnvironments**) | Get only |

### Applications

Used to navigate to the `INatAutoApplications` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoApplications**) | Get only |

### Sysmain

Used to create a new instance of the `INatAutoSysmain` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoSysmain**) | Get only |

## Methods

The following methods are available:

- Quit
- SysCreateGuid
- CMPALUTL
- Logon

### Quit

Terminates Natural Studio.

### SysCreateGuid

Creates a global unique ID (GUID). A plug-in might need this method when generating a class.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | A | VT_BSTR | |

**Return value**

A fresh global unique ID (GUID) in registry format (that is: enclosed in curly braces). Returns an empty string, if the creation failed.

### CMPALUTL

Used to call the development server through the utility protocol. Currently the utility protocol is only used internally.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| UtilityID | I2 | VT_I2 | By value |
| BufferLength | I4 | VT_I4 | By value |
| Buffer | A | VT_BSTR | By reference |

**Return value**
Depends on the specific utility being called.

**UtilityID**
The utility being called.

**BufferLength**
The length of the data buffer passed in Buffer.

**Buffer**
The data buffer passed to and returned from the utility. The contents of the buffer on input and output depend on the utility being called. The utility protocol requires that the buffer be large enough to hold the maximum expected result of the utility request. This size depends on the utility being called.

### Logon

Used to perform a logon to a specific library in the active environment.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Library | A | VT_BSTR | By value |
| Fnr | I4 | VT_I4 | Optional |
| DBnr | I4 | VT_I4 | Optional |

**Library**
The library to logon to.

**Fnr, DBnr**
The file number and database number of the system file the library belongs to. Usually these parameters need not be specified, because the system file is determined by the library name.

# 54 INatAutoToolBar

## Purpose

Gives access to a specific toolbar.

## Properties

The following properties are available:

- Parent
- Studio
- Caption
- Visible

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoToolBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Caption

A string used to identify the toolbar, as defined when the toolbar was created.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| A | VT_BSTR | Get only |

### Visible

Indicates if the toolbar is currently visible or not. Modifying this property hides or shows the toolbar.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| L | VT_BOOL | |

## Methods

The following methods are available:

- InsertCommand
- InsertSeparator
- Dock

### InsertCommand

Inserts a command into the toolbar.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Command | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoCommand**) | |
| Index | I4 | VT_I4 | Optional |

**Command**

A command to be added to the toolbar. The command must have been defined before using the method `INatAutoCommands::Add`.

**Index**

The position in the toolbar where the command shall be inserted. If Index is omitted, the command is inserted at the last position.

### InsertSeparator

Inserts a separator into the toolbar.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | | | None |
| Index | I4 | VT_I4 | Optional |

**Index**
The position in the toolbar where the separator shall be inserted. If Index is omitted, the separator is inserted at the last position.

### Dock

Docks the toolbar to another toolbar or to the Natural Studio frame window. The docking position of dynamically created toolbars is not retained persistently between Natural Studio sessions.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| AtToolbar | A | VT_BSTR | Optional |

**AtToolbar**
If the toolbar specified in AtToolBar is docked horizontally on the top or bottom of the frame window, the current toolbar is docked on the right of this toolbar.

If the toolbar specified in AtToolBar is docked vertically on the left or right hand side of the frame window, the current toolbar is docked below this toolbar.

If AtToolBar is not specified or the specified toolbar does not exist or is not visible or is not docked, the current toolbar is docked at the top of the frame window.

# 55 INatAutoToolBars

## Purpose

Collection of the available toolbars.

## Properties

The following properties are available:

- Parent
- Studio
- Count

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoControlBars**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Count

The number of available toolbars.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| I4 | VT_I4 | Get only |

## Methods

The following methods are available:

- Add
- Item

### Add

Creates a new toolbar and adds it to the collection. Dynamically created toolbars are not persistently customizable in the **Customize** dialog.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoToolBar**) | |
| Caption | A | VT_BSTR | |
| Visible | L | VT_BOOL | Optional |

**Return value**
The newly added toolbar.

**Caption**
A string used to identify the toolbar.

**Visible**
Decides if the toolbar is created visibel or not. By default, the toolbar is created visible.

### Item

Returns a specific toolbar from the collection. Used to iterate through the collection.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoToolBar**) | |
| Index | I4<br>A | VT_I4 VT_BSTR | |

**Return value**
The toolbar identified by the value specified in Index.

**Index**

Identifies a specific toolbar in the collection. This can be either the index of the toolbar in the collection (a value between 1 and Count) or the caption of the toolbar.

# 56 INatAutoTypes

## Purpose

Contains collections that are used to define new tree view and list view node types.

## Properties

The following properties are available:

- Parent
- Studio
- NodeImages
- NodeTypes

### Parent

Used to navigate to the parent interface of this interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### Studio

Used to navigate to the root interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | Get only |

### NodeImages

Used to navigate to the `INatAutoNodeImages` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNodeImages**) | Get only |

### NodeTypes

Used to navigate to the `INatAutoNodeTypes` interface.

| Natural Data Format | Variant Type | Remark |
|---|---|---|
| HANDLE OF OBJECT | VT_DISPATCH (**INatAutoNodeTypes**) | Get only |

# 57 INaturalStudioPlugIn

## Purpose

This is the primary interface a plug-in must provide. Natural Studio uses this interface to activate and deactivate the plug-in and to send commands and notifications to it.

In order to provide the interface, plug-ins written in Natural include the interface module (copy-code) `NSTPLG-I` from the example library `SYSEXPLG` in their class definition. Plug-ins written in other languages use the type library `naturalstudioplugin.tlb`. This type library is also contained in the example library `SYSEXPLG`.

## Methods

The following methods are available:

- OnActivate
- OnDeactivate
- OnCommand
- OnCommandStatus
- OnNotify

### OnActivate

Natural Studio calls this method when it activates the plug-in. The plug-in should use this opportunity to define its commands and to make them visible in the Natural Studio user interface. Also it might store a handle to the Natural Studio Automation root interface (`INatAutoStudio`) for further use.

If a plug-in determines that it cannot activate because certain resources or prerequisites are missing, it should set `*ERROR-NR` to 9002 on return. This causes the plug-in framework to call the method `OnDeactivate` for the necessary cleanup and to leave the plug-in in inactive status. The plug-in itself is responsible to alert the user in an appropriate way.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| NaturalStudio | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | |

**NaturalStudio**
Contains a handle to the Natural Studio Automation root interface.

### OnDeactivate

Natural Studio calls this method when it deactivates the plug-in. The plug-in should use this opportunity to close windows, files and network connections etc. and to clean up other used resources. It does not need to remove the commands, menu items and toolbar items that is might have created in the method `OnActivate`. This is done by Natural Studio automatically.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| NaturalStudio | HANDLE OF OBJECT | VT_DISPATCH (**INatAutoStudio**) | |

**NaturalStudio**
Contains a handle to the Natural Studio Automation root interface.

### OnCommand

Natural Studio calls this method when the user selects one of the commands the plug-in has defined. Usually the plug-in will then apply the command to the set of objects that are currently selected. It retrieves this set through the interface `INatAutoSelectedObjects`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| CommandID | I4 | VT_I4 | |

**CommandID**
Contains the command ID the plug-in has chosen when it defined the command with the method `Add` of the interface `INatAutoCommands`.

### OnCommandStatus

Natural Studio calls this method when one of the commands the plug-in has defined becomes visible in the user interface, for instance, when the user opens a pop-up menu that contains one of these commands. The plug-in decides if the command is to be enabled or not and if it is to have a check mark or not. By default, all plug-in defined commands are disabled and unchecked. Usually the plug-in will decide about the command status based on the set of objects that are currently selected. It retrieves this set through the interface `INatAutoSelectedObjects`.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| CommandID | I4 | VT_I4 | |
| Enabled | L | VT_BOOL | By reference |
| Checked | L | VT_BOOL | By reference |

**CommandID**
Contains the command ID the plug-in has chosen when it defined the command with the method `Add` of the interface `INatAutoCommands`.

**Enabled**
If the command is to be enabled, the plug-in returns TRUE in this parameter.

**Checked**
If the command is to have a check mark, the plug-in returns TRUE in this parameter.

### OnNotify

Natural Studio calls this method to notify the plug-in about certain events in Natural Studio that might be of interest for the plug-in. However, the plug-in does not have to use any of these notifications.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| Return value | I4 | VT_I4 | |
| Notification | I4 | VT_I4 | |
| LongParam | I4 | VT_I4 | By reference |
| ObjectParam | HANDLE OF OBJECT | VT_DISPATCH | By reference |
| StringParam | A | VT_BSTR | By reference |

**Return value, LongParam, ObjectParam, StringParam**
The usage and meaning of these parameters depends on the specific notification. Please refer to the specification of the individual notifications in the following.

**Notification**
A number that identifies the notification. The individual notifications are specified in the following. There are constant definitions available for the notification numbers in the local data area `NSTPLG-L` in the example library `SYSEXPLG`.

## Notifications

The following notifications are available:

- PLUGIN-NOTIFY-ACTIVATE
- PLUGIN-NOTIFY-QUERYCLOSE
- PLUGIN-NOTIFY-CLOSE
- PLUGIN-NOTIFY-SAVE
- PLUGIN-NOTIFY-EXPANDALL
- PLUGIN-NOTIFY-SELECTEDOBJECTS
- PLUGIN-NOTIFY-FOCUSOBJECT
- PLUGIN-NOTIFY-CONTEXTMENU
- PLUGIN-NOTIFY-REFRESH
- PLUGIN-NOTIFY-HELP
- PLUGIN-NOTIFY-OPTIONSVALIDATE
- PLUGIN-NOTIFY-OPTIONSMODIFIED

### PLUGIN-NOTIFY-ACTIVATE

Natural Studio sends this notification to plug-ins that have created tree view document windows, list view document windows or generic document windows. It sends it to inform the plug-in about the activation status of one of these windows.

**Return value**

Not used.

**LongParam**

| | |
|---|---|
| 0 | if the window is being deactivated. |
| 1 | if the window is being activated. |
| 2 | if the window is already active and the user clicks a mouse button inside the window. |

**ObjectParam**

A handle to the document window.

**StringParam**

A string that identifies the type of document window: `INatAutoObjectTree`, `INatAutoObjectList` or `INatAutoGenericDocument`.

### PLUGIN-NOTIFY-QUERYCLOSE

Natural Studio sends this notification to plug-ins that created generic document windows. It sends it to inform the plug-in that the Natural Studio user is attempting to close one of these windows, or that the `Close` method has been called. The plug-in might use this notification to check if there are uncommitted changes in the document and to take appropriate actions if this is the case.

**Return value**

The plug-in returns:

| | |
|---|---|
| 0 | if it accepts that the window is closed. |
| 1 | to prevent closing the window. |

**LongParam**

Not used.

**ObjectParam**

A handle to the document window.

**StringParam**

A string that identifies the type of document window: `INatAutoGenericDocument`.

### PLUGIN-NOTIFY-CLOSE

Natural Studio sends this notification to plug-ins that created tree view document windows, list view document windows or generic document windows. It sends it to inform the plug-in that the Natural Studio user is attempting to close one of these windows, or that the `Close` method has been called.

**Return value**

Not used.

**LongParam**

Not used.

**ObjectParam**

A handle to the document window.

**StringParam**

A string that identifies the type of document window: `INatAutoObjectTree`, `INatAutoObjectList` or `INatAutoGenericDocument`.

### PLUGIN-NOTIFY-SAVE

Natural Studio sends this notification to plug-ins that have opened generic text objects in the program editor. It sends it to inform the plug-in that the user has triggered the Save command. This enables the plug-in to retrieve the edited text and to save it.

**Return value**
The plug-in returns:

| | |
|---|---|
| 1 | to indicate that it has successfully saved the text. |
| 0 | otherwise. |

**LongParam**
Not used.

**ObjectParam**
A handle to the document window.

**StringParam**
A string that identifies the type of document window: `INatAutoGenericText`.

### PLUGIN-NOTIFY-EXPANDALL

Natural Studio sends this notification to plug-ins that have created tree view document windows. It sends it to inform the plug-in that an Expand All has been started or has finished on one of these windows. This enables the plug-in to apply possible optimizations when Natural Studio later calls it repetitively while performing the Expand All.

**Note:** The user issues an Expand All by pressing the Multiply key on the numeric keypad while a tree view node is selected. This causes the tree view node to be expanded recursively.

**Return value**
Not used.

**LongParam**

| | |
|---|---|
| 1 | if Expand All has started. |
| 0 | if Expand All has finished. |

**ObjectParam**
A handle to the document window.

**StringParam**
A string that identifies the type of document window: `INatAutoObjectTree`.

### PLUGIN-NOTIFY-SELECTEDOBJECTS

Natural Studio sends this notification to plug-ins that have created generic document windows. It sends it to retrieve the currently selected objects in the currently active generic document window.

**Return value**
Not used.

**LongParam**
Not used.

**ObjectParam**
Not used.

**StringParam**
The plug-in returns the set of selected objects formatted as an XML document. The document must comply to the DTD specified for the property `SelectedObjects` of the interface `INatAutoSelectedObjects`.

### PLUGIN-NOTIFY-FOCUSOBJECT

Natural Studio sends this notification to plug-ins that have created a generic document window. It sends it to retrieve the object that currently has the focus in the currently active generic document window.

#### Return value

Not used.

#### LongParam

Not used.

#### ObjectParam

Not used.

#### StringParam

The plug-in returns the focus object formatted as an XML document. The document must comply with the DTD specified for the property `SelectedObjects` of the interface `INatAutoSelectedObjects`.

### PLUGIN-NOTIFY-CONTEXTMENU

Natural Studio sends this notification to plug-ins that have created generic document windows. It sends it to inform the plug-in that the user tries to open a context menu on one of these windows by clicking the right mouse button or pressing the context menu key. This enables the plug-in to have different context menus displayed depending on the mouse position or to display a default context menu for the window as a whole.

**Return value**

The plug-in returns:

| | |
|---|---|
| 1 | if a context menu is to be displayed. |
| 0 | otherwise. |

**LongParam**

The current mouse position in the form x * (2**16) + y.

This means: If the mouse position is for instance, (50,100), the value in LongParam will be 50 * (2**16) + 100 = 3276900. A value of -1 indicates that the default (position independent) context menu was requested.

**ObjectParam**

The plug-in returns a `INatAutoContextMenu` interface to a context menu it has created or retrieved before.

**StringParam**

Not used.

### PLUGIN-NOTIFY-REFRESH

Natural Studio sends this notification to plug-ins that have created tree view document windows, list view document windows or generic document windows. It sends it to inform the plug-in that one of these windows possibly needs to be refreshed. While handling this notification, the plug-in has access to the interface `INatAutoRefreshObject`. This interface and its properties allow retrieving the details about the object currently being refreshed.

**Return value**

The plug-in returns:

| | |
|---|---|
| 1 | if the window is to be refreshed. |
| 0 | if it is not to be refreshed. |

**LongParam**

| | |
|---|---|
| On Input | The value 0 indicates that Natural Studio just queries whether a refresh is to be performed. Natural Studio passes this value if the notification is sent with respect to a tree view document window or list view document window. In these cases Natural Studio can perform the refresh itself. The plug-in has just to decide whether it wants the view to be refreshed or not.<br><br>The value 1 indicates that Natural Studio advises the plug-in to refresh the window. Natural Studio passes this value if the notification is sent with respect to a generic document window. In this case, the plug-in is in charge of performing the refresh. |
| On Return | The value 0 indicates that the plug-in wants to have this refresh executed as a specific refresh. This means: only the visualizations of the current refresh object (`INatAutoRefreshObject`) is to be refreshed.<br><br>The value 1 indicates that the plug-in wants to have this refresh executed as an unspecific refresh. This means: the whole window is to be refreshed.<br><br>**Note:** For list view document windows Natural Studio currently makes no difference between a specific and an unspecific refresh. In both cases the whole window will be refreshed. |

**ObjectParam**

A handle to the document window.

**StringParam**

A string that identifies the type of document window: `INatAutoObjectTree`, `INatAutoObjectList` or `INatAutoGenericDocument`.

### PLUGIN-NOTIFY-HELP

Natural Studio sends this notification to plug-ins that have created tree view document windows or list view document windows. It sends it to inform the plug-in that the user has pressed the F1 key (and thus requested help), while one of these windows was active. While handling this notification, the plug-in has access to the interface `INatAutoSelectedObjects`. This interface and its properties allow retrieving details about the objects currently being selected and about the focus object, in order to display context-specific help. In order to display a specific help topic, the plug-in uses the method `INatAutoStudio::ShowHelp`.

**Return value**

The plug-in returns:

| | |
|---|---|
| 1 | if it has handled the help request by displaying a help topic. |
| 0 | if not. |

**LongParam**
Not used.

**ObjectParam**
A handle to the document window.

**StringParam**
A string that identifies the type of document window: `INatAutoObjectTree` or `INatAutoObjectList`.

### PLUGIN-NOTIFY-OPTIONSVALIDATE

Natural Studio sends this notification to plug-ins that have specified options. It sends it to inform the plug-in that the user is attempting to modify the option values and allows the plug-in to validate the new values. The notification is sent when the user has pressed **OK** or **Apply** in the **Options** dialog or if the user switches to a different property page after having modified the plug-in options.

**Note:** A plug-in can specify options by using the method `INatAutoPlugIn::DefineOptions`. It can retrieve and set the option values by using the property `INatAutoPlugIn::OptionValues`. The user can modify the option values interactively in the Natural Studio **Options** dialog.

**Return value**
The plug-in returns:

| | |
|---|---|
| 1 | if the modified option values are valid. |
| 0 | if they are invalid. |

Additionally it might do whatever necessary to alert the user, for instance, display a message box.

**LongParam**
Not used.

**ObjectParam**
Not used.

**StringParam**
Contains the modified option setting as an XML document according to the Option values DTD. This DTD is defined in `INatAutoPlugIn::OptionValues`.

### PLUGIN-NOTIFY-OPTIONSMODIFIED

Natural Studio sends this notification to plug-ins that have specified options. It sends it to inform the plug-in that the user has successfully modified the option values and allows the plug-in to react to the change appropriately.

**Note:** A plug-in can specify options through the method `INatAutoPlugIn::DefineOptions`. It can retrieve and set the option values through the property `INatAutoPlugIn::OptionValues`. The user can modify the option values interactively in the Natural Studio **Options** dialog.

**Return value**
Not used.

**LongParam**
Not used.

**ObjectParam**
Not used.

**StringParam**
Contains the modified option setting as an XML document according to the Option values DTD. This DTD is defined in `INatAutoPlugIn::OptionValues`.

# 58 INaturalStudioPlugInTree

## Purpose

A plug-in provides this interface in order to provide information about tree view and list view nodes to Natural Studio. Natural Studio calls the methods of this interface if the plug-in has defined its own node types and has opened a tree view or list view document window with a node of one of these types as root node. Natural Studio calls the methods to gather information about these nodes whenever this is required to expand or refresh a tree view or list view.

**Note:** A plug-in defines its own node types by using the interface `INatAutoNodeTypes`.

In order to provide the interface, plug-ins written in Natural include the interface module (copy-code) `NSTPLG-T` from the example library `SYSEXPLG` in their class definition. Plug-ins written in other languages use the type library `naturalstudioplugin.tlb`. This type library is also contained in the example library `SYSEXPLG`.

## Methods

The following methods are available:

- GetData
- GetChildren
- HasChildren
- GetAttributes
- GetAttributeValues

### GetData

Natural Studio calls this method to retrieve additional information about a plug-in defined node identified by type and key.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| ReturnValue | L | VT_BOOL | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Info | A | VT_BSTR | |
| Template | I4 | VT_I4 | |
| Data | A | VT_BSTR | By reference |

**ReturnValue**
The plug-in returns TRUE if it knows the node, FALSE otherwise.

**Type**
The type of the node.

**Key**
The key that identifies the node within its type.

**Info**
An additional information string that the plug-in has previously assigned to the node.

**Template**
The TemplateID that the plug-in has passed to the method `INatAutoObjectTrees::Open` or `INatAutoObjectLists::Open` when opening the tree view or list view document window.

**Data**
The plug-in returns a string containing an XML document that describes the node. The XML document is structured according to the following DTD.

```
<?xml version="1.0"?>
<!ELEMENT data (pinfo?, pname?, (ntype, nkey)?)>
<!ELEMENT pinfo (#PCDATA)>
<!ELEMENT pname (#PCDATA)>
<!ELEMENT ntype (#PCDATA)>
<!ELEMENT nkey  (#PCDATA)>
```

| Element | Meaning |
|---|---|
| pinfo | Additional information about the node that the plug-in wants to receive back whenever Natural Studio later refers to the node. Natural Studio never considers the content of this element, but just passes it back and forth. |
| pname | The text to be displayed with the node in a tree view or list view. |
| ntype | If the node has been defined by a plug-in, but corresponds to an node of a predefined Natural Studio type, this element contains the type of the corresponding predefined node. |
| nkey | If the node has been defined by a plug-in, but corresponds to an node of a predefined Natural node type, this element contains the key of the corresponding predefined node. |

### GetChildren

Natural Studio calls this method to retrieve the child nodes of a node defined by a plug-in. This node is identified by type and key.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| ReturnValue | I4 | VT_I4 | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Info | A | VT_BSTR | |
| Template | I4 | VT_I4 | |
| Children | A | VT_BSTR | By reference |

**ReturnValue**
Indicates the number of child nodes.

**Type**
The type of the node.

**Key**
The key that identifies the node within its type.

**Info**
An additional information string that the plug-in has previously assigned to the node.

**Template**
The TemplateID that the plug-in has passed to the method `INatAutoObjectTrees::Open` or `INatAutoObjectLists::Open` when opening the tree view or list view document window. The plug-in can interpret the parameter `Template` to return different tree structures for different values of `Template`.

**Children**
The plug-in returns a string containing an XML document that describes the child nodes. The XML document is structured according to the following DTD.

```
<?xml version="1.0"?>
<!ELEMENT children (child*)>
<!ELEMENT child (ptype, pkey, pinfo?, pname?,
phch?, children?, (ntype, nkey)?)>
<!ELEMENT ptype (#PCDATA)>
<!ELEMENT pkey  (#PCDATA)>
<!ELEMENT pinfo (#PCDATA)>
<!ELEMENT pname (#PCDATA)>
<!ELEMENT phch  (#PCDATA)>
<!ELEMENT ntype (#PCDATA)>
<!ELEMENT nkey  (#PCDATA)>
```

| Element | Meaning |
|---|---|
| ptype | The type of the child node. |
| pkey | The key that identifies the child node within its type. |
| pinfo | Additional information about the child node that the plug-in wants to receive back whenever Natural Studio later refers to the node. Natural Studio never considers the content of this element, but just passes it back and forth. |
| pname | The text to be displayed with the child node in a tree view or list view. |
| children | Allows specifying a subtree of child nodes in arbitrary depth. If this element is specified, it is implicitly assumed that the child node itself has children and the method `HasChildren` is not called for this child node. |
| phch | Allows specifying in advance if the given child node itself has child nodes.<br>■ A value of 1 means that the child node itself has child nodes.<br>■ A value of -1 means that the child node itself has no child nodes.<br>■ A value of 0 (default) means that the plug-in cannot determine now if the child node itself has child nodes and wants to be asked in a subsequent call to the method `HasChildren`. |
| ntype | If the node has been defined by a plug-in, but corresponds to a node of a predefined Natural Studio type, this element contains the type of the corresponding predefined node. |
| nkey | If the node has been defined by a plug-in, but corresponds to a node of a predefined Natural node type, this element contains the key of the corresponding predefined node. |

## HasChildren

Natural Studio calls this method to check if the plug-in defined node identified by type and key has child nodes of any type. This is used to decide if the node will be shown as expandable in a tree view document window.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| ReturnValue | L | VT_BOOL | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Info | A | VT_BSTR | |
| Template | I4 | VT_I4 | |
| Children | A | VT_BSTR | By reference |

**ReturnValue**
The plug-in returns TRUE if the node has child nodes, FALSE otherwise.

**Type**
The type of the node.

**Key**
The key that identifies the node within its type.

**Info**
An additional information string that the plug-in has previously assigned to the node.

**Template**
The TemplateID that the plug-in has passed to the method `INatAutoObjectTrees::Open` or `INatAutoObjectLists::Open` when opening the tree view or list view document window. The plug-in can interpret the parameter `Template` to return different tree structures for different values of `Template`.

**Children**
If the plug-in not only knows that the given node has child nodes, but knows also the child nodes themselves, it can return them in this parameter. The plug-in then returns a string containing an XML document that describes the child nodes. The XML document is structured according to the DTD defined with the method `GetChildren`.

### GetAttributes

Natural Studio calls this method to retrieve meta information about the attributes of the given node type. This method is called when the attributes of the node are to be displayed in a list view document window.

#### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| ReturnValue | I4 | VT_I4 | |
| Type | I4 | VT_I4 | |
| Template | I4 | VT_I4 | |
| Attributes | A | VT_BSTR | By reference |

**ReturnValue**
The number of defined attributes.

**Type**
The type of the node.

**Template**
The TemplateID that the plug-in has passed to the method `INatAutoObjectTrees::Open` or `INatAutoObjectLists::Open` when opening the tree view or list view document window. The plug-in can interpret the parameter `Template` to return different attribute sets for different values of `Template`.

**Attributes**
The plug-in returns a string containing an XML document that describes the attributes. The XML document is structured according to the following DTD.

```
<?xml version="1.0"?>
<!ELEMENT attributes (attdef*)>
<!ELEMENT attdef     (akey, acaption?)>
<!ELEMENT akey     (#PCDATA)>
<!ELEMENT acaption (#PCDATA)>
```

| Element | Meaning |
|---|---|
| akey | A key that identifies the attribute internally. This key is freely defined by the plug-in. |
| acaption | A caption that is used to display the attribute externally, for instance, in list view column headers. |

For nodes that correspond to predefined Natural Studio node types the attributes of the corresponding Natural node type can also be specified. In order to refer to these attributes, the following predefined attribute keys are used. If one of these attribute keys is used, no attribute caption needs to be specified, because the captions are predefined by Natural Studio.

| Key | Caption |
|---|---|
| NOLibrary | Library name |
| NOFnr | File number |
| NODBID | Database number |
| NOType | Natural object type |
| NOMode | Mode (Structured/Report) |
| NOUserID | User ID |
| NOSrcDate | Date of last save |
| NOSrcSize | Source size |
| NOCatDate | Date of last catalog |
| NOCatSize | GP size |

### GetAttributeValues

Natural Studio calls this method to retrieve the attribute values of the node defined by a plug-in. This node is identified by type and key.

### Parameters

| Name | Natural Data Format | Variant Type | Remark |
|---|---|---|---|
| ReturnValue | I4 | VT_I4 | |
| Type | I4 | VT_I4 | |
| Key | A | VT_BSTR | |
| Info | A | VT_BSTR | |
| Template | I4 | VT_I4 | |
| AttributeValues | A | VT_BSTR | By reference |

**ReturnValue**
The number of attribute values.

**Type**
The type of the node.

**Key**
The key that identifies the node within its type.

**Info**
An additional information string that the plug-in has previously assigned to the node.

**Template**
The TemplateID that the plug-in has passed to the method `INatAutoObjectTrees::Open` or `INatAutoObjectLists::Open` when opening the tree view or list view document window. The plug-in can interpret the parameter `Template` to return different attribute sets for different values of `Template`.

**AttributeValues**
The plug-in returns a string containing an XML document that describes the attribute values. The XML document is structured according to the following DTD.

```
<?xml version="1.0"?>
<!ELEMENT attributevalues (attval*)>
<!ELEMENT attval  (akey, avalue)>
<!ELEMENT akey (#PCDATA)>
<!ELEMENT avalue (#PCDATA)>
```

| Element | Meaning |
|---|---|
| akey | A key that identifies the attribute internally. The key must match one of the keys previously returned in a call to `GetAttributes`, otherwise the element is ignored. |
| avalue | The attribute value. |

# III DTDs

Because of their length the DTDs listed below are provided separately in this part. All other DTDs which are used by the Natural Studio interfaces are documented together with the corresponding interface.

# 59 DTD for INatAutoNatparm - Local Environment

Applies to `INatAutoNatparm::ProfileParameters`.

For the local environment, the XML document is structured according to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ACIPATT (#PCDATA)>
<!ELEMENT ACIVERS (#PCDATA)>
<!ELEMENT ACTIVATED (#PCDATA)>
<!ELEMENT ACTPOLICY (#PCDATA)>
<!ELEMENT ADA (ET, ETID, MFSET, RCFIND, RCGET, OPRB, WH)>
<!ELEMENT ADM_LFL_COUNT (#PCDATA)>
<!ELEMENT AUTO (#PCDATA)>
<!ELEMENT AUTOREGISTER (#PCDATA)>
<!ELEMENT AUTORPC (#PCDATA)>
<!ELEMENT BATCH (BMCONTROL, BMBLANK, NATLOG, BMSIM, BMTIME, BMVERSION, BMTITLE, CC,
CMOBJIN, CMPRINT, CMSYNIN, ECHO, ENDMSG, FRAME)>
<!ELEMENT BMBLANK (#PCDATA)>
<!ELEMENT BMCONTROL (#PCDATA)>
<!ELEMENT BMSIM (#PCDATA)>
<!ELEMENT BMTIME (#PCDATA)>
<!ELEMENT BMTITLE (#PCDATA)>
<!ELEMENT BMVERSION (#PCDATA)>
<!ELEMENT BPSFI (#PCDATA)>
<!ELEMENT BUFSIZES (SSIZE, USIZE, DSLM, SORTSZE)>
<!ELEMENT CALLNAT (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT CDYNAM (#PCDATA)>
<!ELEMENT CF (#PCDATA)>
<!ELEMENT CHARS (CF, CLEAR, DC, FC, HI, IA, ID, TDS, THSEPCH)>
<!ELEMENT CIPHER (#PCDATA)>
<!ELEMENT CLEAR (#PCDATA)>
<!ELEMENT CLOSEMODE (#PCDATA)>
<!ELEMENT CLR (#PCDATA)>
<!ELEMENT CM (#PCDATA)>
<!ELEMENT CMOBJIN (#PCDATA)>
<!ELEMENT CMPRINT (#PCDATA)>
<!ELEMENT CMSYNIN (#PCDATA)>
<!ELEMENT CO (#PCDATA)>
```

```
<!ELEMENT COMPOPT (DBSHORT, ENDIAN, GFID, THSEP, MASKCME, PCHECK)>
<!ELEMENT COMPR (#PCDATA)>
<!ELEMENT COMSERVER (#PCDATA)>
<!ELEMENT CP (#PCDATA)>
<!ELEMENT CSCPATT (#PCDATA)>
<!ELEMENT CVMIN (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT DBID (#PCDATA)>
<!ELEMENT DBMS (UDB, ADM_LFL_COUNT, ETDB, ETEOP, TF_CNT, LFILMAX, LFL-ADM149, LFL-ADM150,
LFL-ADM151, LFL-ADM190)>
<!ELEMENT DBSHORT (#PCDATA)>
<!ELEMENT DBUPD (#PCDATA)>
<!ELEMENT DC (#PCDATA)>
<!ELEMENT DCOM (AUTOREGISTER, COMSERVER, ACTPOLICY)>
<!ELEMENT DD (#PCDATA)>
<!ELEMENT DEV00 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV01 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV02 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV03 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV04 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV05 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV06 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV07 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV08 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV09 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV10 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV11 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV12 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV13 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV14 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV15 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV16 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV17 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV18 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV19 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV20 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV21 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV22 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV23 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV24 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV25 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV26 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV27 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV28 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV29 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV30 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV31 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV32 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEV33 (LINESIZE, PAGESIZE, MAXPAGE, LOG_NAME, NAME, STATUS, PRT_OUTPUT, METHOD)>
<!ELEMENT DEVICES (DEV00, DEV01, DEV02, DEV03, DEV04, DEV05, DEV06, DEV07, DEV08,
DEV09, DEV10, DEV11, DEV12, DEV13, DEV14, DEV15, DEV16, DEV17, DEV18, DEV19,
DEV20, DEV21, DEV22, DEV23, DEV24, DEV25, DEV26, DEV27, DEV28, DEV29, DEV30,
DEV31, DEV32, DEV33, REP)>
<!ELEMENT DFOUT (#PCDATA)>
<!ELEMENT DFSTACK (#PCDATA)>
<!ELEMENT DFTITLE (#PCDATA)>
<!ELEMENT DISPDBG (#PCDATA)>
<!ELEMENT DSLM (#PCDATA)>
```

```
<!ELEMENT DTFORM (#PCDATA)>
<!ELEMENT DU (#PCDATA)>
<!ELEMENT DYNPARM (#PCDATA)>
<!ELEMENT ECHO (#PCDATA)>
<!ELEMENT ECPMOD (#PCDATA)>
<!ELEMENT EDITOR (#PCDATA | EDTBPSIZE | EDTLFILES | EDTRB)*>
<!ELEMENT EDTBPSIZE (#PCDATA)>
<!ELEMENT EDTLFILES (#PCDATA)>
<!ELEMENT EDTRB (#PCDATA)>
<!ELEMENT EJ (#PCDATA)>
<!ELEMENT EMFM (#PCDATA)>
<!ELEMENT ENDIAN (#PCDATA)>
<!ELEMENT ENDMSG (#PCDATA)>
<!ELEMENT ERROR (IKEY, MSGSF, SA, SNAT)>
<!ELEMENT ESCAPE (#PCDATA)>
<!ELEMENT ESX (ESXDB)>
<!ELEMENT ESXDB (#PCDATA)>
<!ELEMENT ET (#PCDATA)>
<!ELEMENT ETA (#PCDATA)>
<!ELEMENT ETDB (#PCDATA)>
<!ELEMENT ETEOP (#PCDATA)>
<!ELEMENT ETID (#PCDATA)>
<!ELEMENT FC (#PCDATA)>
<!ELEMENT FCDP (#PCDATA)>
<!ELEMENT FDDM (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT FDIC (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT FNAT (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT FNR (#PCDATA)>
<!ELEMENT FRAME (HB, LLC, LRC, ULC, URC, VB)>
<!ELEMENT FREEGDA (#PCDATA)>
<!ELEMENT FS (#PCDATA)>
<!ELEMENT FSEC (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT FUSER (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT GFID (#PCDATA)>
<!ELEMENT GRAPHIC (#PCDATA)>
<!ELEMENT HB (#PCDATA)>
<!ELEMENT HI (#PCDATA)>
<!ELEMENT IA (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT IKEY (#PCDATA)>
<!ELEMENT IM (#PCDATA)>
<!ELEMENT INITLIB (#PCDATA)>
<!ELEMENT KC (#PCDATA)>
<!ELEMENT KEYS (ACTIVATED, CLR, PA1, PA2, PA3, PF1, PF2, PF3, PF4, PF5, PF6, PF7,
PF8, PF9, PF10, PF11, PF12, PF13, PF14, PF15, PF16, PF17, PF18, PF19, PF20, PF21,
PF22, PF23, PF24)>
<!ELEMENT LC (#PCDATA)>
<!ELEMENT LDB (#PCDATA)>
<!ELEMENT LE (#PCDATA)>
<!ELEMENT LFILMAX (#PCDATA)>
<!ELEMENT LFL-ADM149 (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT LFL-ADM150 (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT LFL-ADM151 (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT LFL-ADM190 (DBID, FNR, PASSWD, CIPHER, ROSY)>
<!ELEMENT LIMITS (LDB, LE, LT, MADIO, MAXCL, SD)>
<!ELEMENT LINESIZE (#PCDATA)>
<!ELEMENT LLC (#PCDATA)>
<!ELEMENT LOGN (#PCDATA)>
```

```
<!ELEMENT LOGONRQ (#PCDATA)>
<!ELEMENT LOG_NAME (#PCDATA)>
<!ELEMENT LRC (#PCDATA)>
<!ELEMENT LS (#PCDATA)>
<!ELEMENT LT (#PCDATA)>
<!ELEMENT MADIO (#PCDATA)>
<!ELEMENT MAINPR (#PCDATA)>
<!ELEMENT MASKCME (#PCDATA)>
<!ELEMENT MAXBUFF (#PCDATA)>
<!ELEMENT MAXCL (#PCDATA)>
<!ELEMENT MAXPAGE (#PCDATA)>
<!ELEMENT METHOD (#PCDATA)>
<!ELEMENT MFSET (#PCDATA)>
<!ELEMENT MISC (AUTO, BPSFI, CC, CM, CO, DBUPD, DD, DU, DYNPARM, ESCAPE, FCDP,
FS, GRAPHIC, IM, KC, ML, NC, NENTRY, OPF, PC, PD, RECAT, REINP, SM, STACK,
SYMGEN, SYNERR, TD, TQ, TS, ULANG, XREF, ZD, CVMIN, TMPSORTUNIQ, NOAPPLERR)>
<!ELEMENT ML (#PCDATA)>
<!ELEMENT MSGSF (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT NATENV (STEP_CNT, USER, EDITOR, SHELL, STEPLIBS)>
<!ELEMENT NATLOG (#PCDATA)>
<!ELEMENT NATRPC (ACIPATT, AUTORPC, COMPR, CSCPATT, DISPDBG, LOGONRQ, MAXBUFF,
NO_OF_RDS, SERVER, RPCSIZE, SRVNAME, SRVNODE, SRVUSER, TIMEOUT, TRACE, TRANSP,
TRYALT, ACIVERS, CP, SERVDIR, TRACEONERROR, RPC-DFS)>
<!ELEMENT NATSVAR (INITLIB, STARTUP)>
<!ELEMENT NC (#PCDATA)>
<!ELEMENT NCFVERS (#PCDATA)>
<!ELEMENT NENTRY (#PCDATA)>
<!ELEMENT NOAPPLERR (#PCDATA)>
<!ELEMENT NODE (#PCDATA)>
<!ELEMENT NO_OF_RDS (#PCDATA)>
<!ELEMENT OPF (#PCDATA)>
<!ELEMENT OPRB (#PCDATA)>
<!ELEMENT PA1 (#PCDATA)>
<!ELEMENT PA2 (#PCDATA)>
<!ELEMENT PA3 (#PCDATA)>
<!ELEMENT PAGESIZE (#PCDATA)>
<!ELEMENT PARAMETER-FILE-HEADER (NAME, VERSION, DATE)>
<!ELEMENT PASSWD (#PCDATA)>
<!ELEMENT PC (#PCDATA)>
<!ELEMENT PCHECK (#PCDATA)>
<!ELEMENT PD (#PCDATA)>
<!ELEMENT PERSIST (#PCDATA)>
<!ELEMENT PF1 (#PCDATA)>
<!ELEMENT PF10 (#PCDATA)>
<!ELEMENT PF11 (#PCDATA)>
<!ELEMENT PF12 (#PCDATA)>
<!ELEMENT PF13 (#PCDATA)>
<!ELEMENT PF14 (#PCDATA)>
<!ELEMENT PF15 (#PCDATA)>
<!ELEMENT PF16 (#PCDATA)>
<!ELEMENT PF17 (#PCDATA)>
<!ELEMENT PF18 (#PCDATA)>
<!ELEMENT PF19 (#PCDATA)>
<!ELEMENT PF2 (#PCDATA)>
<!ELEMENT PF20 (#PCDATA)>
<!ELEMENT PF21 (#PCDATA)>
<!ELEMENT PF22 (#PCDATA)>
```

```
<!ELEMENT PF23 (#PCDATA)>
<!ELEMENT PF24 (#PCDATA)>
<!ELEMENT PF3 (#PCDATA)>
<!ELEMENT PF4 (#PCDATA)>
<!ELEMENT PF5 (#PCDATA)>
<!ELEMENT PF6 (#PCDATA)>
<!ELEMENT PF7 (#PCDATA)>
<!ELEMENT PF8 (#PCDATA)>
<!ELEMENT PF9 (#PCDATA)>
<!ELEMENT PLOAD (CDYNAM, ETA, FREEGDA, PROGRAM, PRGPAR, ROSY, PERSIST)>
<!ELEMENT PM (#PCDATA)>
<!ELEMENT PRGPAR (#PCDATA)>
<!ELEMENT PROFILE (PROFILE0, PROFILE1, PROFILE2, PROFILE3, PROFILE4, PROFILE5,
PROFILE6, PROFILE7, PROFILE8, PROFILE9, PROFILE10, PROFILE11, PROFILE12,
PROFILE13, PROFILE14, PROFILE15, PROFILE16, PROFILE17, PROFILE18, PROFILE19,
PROFILE20, PROFILE21, PROFILE22, PROFILE23, PROFILE24, PROFILE25, PROFILE26,
PROFILE27, PROFILE28, PROFILE29, PROFILE30, PROFILE31)>
<!ELEMENT PROFILE0 (#PCDATA)>
<!ELEMENT PROFILE1 (#PCDATA)>
<!ELEMENT PROFILE10 (#PCDATA)>
<!ELEMENT PROFILE11 (#PCDATA)>
<!ELEMENT PROFILE12 (#PCDATA)>
<!ELEMENT PROFILE13 (#PCDATA)>
<!ELEMENT PROFILE14 (#PCDATA)>
<!ELEMENT PROFILE15 (#PCDATA)>
<!ELEMENT PROFILE16 (#PCDATA)>
<!ELEMENT PROFILE17 (#PCDATA)>
<!ELEMENT PROFILE18 (#PCDATA)>
<!ELEMENT PROFILE19 (#PCDATA)>
<!ELEMENT PROFILE2 (#PCDATA)>
<!ELEMENT PROFILE20 (#PCDATA)>
<!ELEMENT PROFILE21 (#PCDATA)>
<!ELEMENT PROFILE22 (#PCDATA)>
<!ELEMENT PROFILE23 (#PCDATA)>
<!ELEMENT PROFILE24 (#PCDATA)>
<!ELEMENT PROFILE25 (#PCDATA)>
<!ELEMENT PROFILE26 (#PCDATA)>
<!ELEMENT PROFILE27 (#PCDATA)>
<!ELEMENT PROFILE28 (#PCDATA)>
<!ELEMENT PROFILE29 (#PCDATA)>
<!ELEMENT PROFILE3 (#PCDATA)>
<!ELEMENT PROFILE30 (#PCDATA)>
<!ELEMENT PROFILE31 (#PCDATA)>
<!ELEMENT PROFILE4 (#PCDATA)>
<!ELEMENT PROFILE5 (#PCDATA)>
<!ELEMENT PROFILE6 (#PCDATA)>
<!ELEMENT PROFILE7 (#PCDATA)>
<!ELEMENT PROFILE8 (#PCDATA)>
<!ELEMENT PROFILE9 (#PCDATA)>
<!ELEMENT PROGRAM (#PCDATA)>
<!ELEMENT PROT (#PCDATA)>
<!ELEMENT PRT_OUTPUT (#PCDATA)>
<!ELEMENT PS (#PCDATA)>
<!ELEMENT ProfileParameters (PARAMETER-FILE-HEADER, ADA, BATCH, BUFSIZES,
CHARS, COMPOPT, DBMS, EDITOR, DCOM, DEVICES, PROFILE, ERROR, ESX, KEYS,
LIMITS, MISC, NATENV, NATSVAR, PLOAD, RDEBUG, SPOD, REMOTE, REPO,
NATRPC, SYSTEM-FILES, WORKF, YEAR2000)>
<!ELEMENT RCFIND (#PCDATA)>
```

```
<!ELEMENT RCGET (#PCDATA)>
<!ELEMENT RDACTIVE (#PCDATA)>
<!ELEMENT RDEBUG (RDACTIVE, RDNODE, RDPORT)>
<!ELEMENT RDNODE (#PCDATA)>
<!ELEMENT RDPORT (#PCDATA)>
<!ELEMENT RECAT (#PCDATA)>
<!ELEMENT REINP (#PCDATA)>
<!ELEMENT REMOTE (USEDIC, USEREP)>
<!ELEMENT REP (REP0, REP1, REP2, REP3, REP4, REP5, REP6, REP7, REP8, REP9,
REP10, REP11, REP12, REP13, REP14, REP15, REP16, REP17, REP18, REP19, REP20,
REP21, REP22, REP23, REP24, REP25, REP26, REP27, REP28, REP29, REP30, REP31,
CLOSEMODE)>
<!ELEMENT REP0 (#PCDATA)>
<!ELEMENT REP1 (#PCDATA)>
<!ELEMENT REP10 (#PCDATA)>
<!ELEMENT REP11 (#PCDATA)>
<!ELEMENT REP12 (#PCDATA)>
<!ELEMENT REP13 (#PCDATA)>
<!ELEMENT REP14 (#PCDATA)>
<!ELEMENT REP15 (#PCDATA)>
<!ELEMENT REP16 (#PCDATA)>
<!ELEMENT REP17 (#PCDATA)>
<!ELEMENT REP18 (#PCDATA)>
<!ELEMENT REP19 (#PCDATA)>
<!ELEMENT REP2 (#PCDATA)>
<!ELEMENT REP20 (#PCDATA)>
<!ELEMENT REP21 (#PCDATA)>
<!ELEMENT REP22 (#PCDATA)>
<!ELEMENT REP23 (#PCDATA)>
<!ELEMENT REP24 (#PCDATA)>
<!ELEMENT REP25 (#PCDATA)>
<!ELEMENT REP26 (#PCDATA)>
<!ELEMENT REP27 (#PCDATA)>
<!ELEMENT REP28 (#PCDATA)>
<!ELEMENT REP29 (#PCDATA)>
<!ELEMENT REP3 (#PCDATA)>
<!ELEMENT REP30 (#PCDATA)>
<!ELEMENT REP31 (#PCDATA)>
<!ELEMENT REP4 (#PCDATA)>
<!ELEMENT REP5 (#PCDATA)>
<!ELEMENT REP6 (#PCDATA)>
<!ELEMENT REP7 (#PCDATA)>
<!ELEMENT REP8 (#PCDATA)>
<!ELEMENT REP9 (#PCDATA)>
<!ELEMENT REPO (DTFORM, EJ, EMFM, LC, LS, MAINPR, PM, PS, SF, ZP)>
<!ELEMENT ROSY (#PCDATA)>
<!ELEMENT RPC-DFS (NAME, NODE, CALLNAT, LOGN, PROT)>
<!ELEMENT RPCSIZE (#PCDATA)>
<!ELEMENT SA (#PCDATA)>
<!ELEMENT SD (#PCDATA)>
<!ELEMENT SERVDIR (#PCDATA)>
<!ELEMENT SERVER (#PCDATA)>
<!ELEMENT SF (#PCDATA)>
<!ELEMENT SHELL (#PCDATA)>
<!ELEMENT SM (#PCDATA)>
<!ELEMENT SNAT (#PCDATA)>
<!ELEMENT SORTSZE (#PCDATA)>
<!ELEMENT SPOD (SPODDEBUGPORT)>
```

```
<!ELEMENT SPODDEBUGPORT (#PCDATA)>
<!ELEMENT SRVNAME (#PCDATA)>
<!ELEMENT SRVNODE (#PCDATA)>
<!ELEMENT SRVUSER (#PCDATA)>
<!ELEMENT SSIZE (#PCDATA)>
<!ELEMENT STACK (#PCDATA)>
<!ELEMENT STARTUP (#PCDATA)>
<!ELEMENT STATUS (#PCDATA)>
<!ELEMENT STEPLIB (#PCDATA)>
<!ELEMENT STEPLIBS (STEPLIB, STEPLIB_1, STEPLIB_2, STEPLIB_3, STEPLIB_4,
STEPLIB_5, STEPLIB_6, STEPLIB_7, STEPLIB_8)>
<!ELEMENT STEPLIB_1 (#PCDATA)>
<!ELEMENT STEPLIB_2 (#PCDATA)>
<!ELEMENT STEPLIB_3 (#PCDATA)>
<!ELEMENT STEPLIB_4 (#PCDATA)>
<!ELEMENT STEPLIB_5 (#PCDATA)>
<!ELEMENT STEPLIB_6 (#PCDATA)>
<!ELEMENT STEPLIB_7 (#PCDATA)>
<!ELEMENT STEPLIB_8 (#PCDATA)>
<!ELEMENT STEP_CNT (#PCDATA)>
<!ELEMENT SYMGEN (#PCDATA)>
<!ELEMENT SYNERR (#PCDATA)>
<!ELEMENT SYSTEM-FILES (FDDM, FDIC, FNAT, FSEC, FUSER)>
<!ELEMENT TD (#PCDATA)>
<!ELEMENT TDS (#PCDATA)>
<!ELEMENT TF_CNT (#PCDATA)>
<!ELEMENT THSEP (#PCDATA)>
<!ELEMENT THSEPCH (#PCDATA)>
<!ELEMENT TIMEOUT (#PCDATA)>
<!ELEMENT TMPSORTUNIQ (#PCDATA)>
<!ELEMENT TQ (#PCDATA)>
<!ELEMENT TRACE (#PCDATA)>
<!ELEMENT TRACEONERROR (#PCDATA)>
<!ELEMENT TRANSP (#PCDATA)>
<!ELEMENT TRYALT (#PCDATA)>
<!ELEMENT TS (#PCDATA)>
<!ELEMENT UDB (#PCDATA)>
<!ELEMENT ULANG (#PCDATA)>
<!ELEMENT ULC (#PCDATA)>
<!ELEMENT URC (#PCDATA)>
<!ELEMENT USEDIC (#PCDATA)>
<!ELEMENT USER (#PCDATA)>
<!ELEMENT USEREP (#PCDATA)>
<!ELEMENT USIZE (#PCDATA)>
<!ELEMENT VB (#PCDATA)>
<!ELEMENT VERSION (#PCDATA)>
<!ELEMENT WFOPFA (#PCDATA)>
<!ELEMENT WH (#PCDATA)>
<!ELEMENT WORK (#PCDATA)>
<!ELEMENT WORKATTR1 (#PCDATA)>
<!ELEMENT WORKATTR10 (#PCDATA)>
<!ELEMENT WORKATTR11 (#PCDATA)>
<!ELEMENT WORKATTR12 (#PCDATA)>
<!ELEMENT WORKATTR13 (#PCDATA)>
<!ELEMENT WORKATTR14 (#PCDATA)>
<!ELEMENT WORKATTR15 (#PCDATA)>
<!ELEMENT WORKATTR16 (#PCDATA)>
<!ELEMENT WORKATTR17 (#PCDATA)>
```

```
<!ELEMENT WORKATTR18 (#PCDATA)>
<!ELEMENT WORKATTR19 (#PCDATA)>
<!ELEMENT WORKATTR2 (#PCDATA)>
<!ELEMENT WORKATTR20 (#PCDATA)>
<!ELEMENT WORKATTR21 (#PCDATA)>
<!ELEMENT WORKATTR22 (#PCDATA)>
<!ELEMENT WORKATTR23 (#PCDATA)>
<!ELEMENT WORKATTR24 (#PCDATA)>
<!ELEMENT WORKATTR25 (#PCDATA)>
<!ELEMENT WORKATTR26 (#PCDATA)>
<!ELEMENT WORKATTR27 (#PCDATA)>
<!ELEMENT WORKATTR28 (#PCDATA)>
<!ELEMENT WORKATTR29 (#PCDATA)>
<!ELEMENT WORKATTR3 (#PCDATA)>
<!ELEMENT WORKATTR30 (#PCDATA)>
<!ELEMENT WORKATTR31 (#PCDATA)>
<!ELEMENT WORKATTR32 (#PCDATA)>
<!ELEMENT WORKATTR4 (#PCDATA)>
<!ELEMENT WORKATTR5 (#PCDATA)>
<!ELEMENT WORKATTR6 (#PCDATA)>
<!ELEMENT WORKATTR7 (#PCDATA)>
<!ELEMENT WORKATTR8 (#PCDATA)>
<!ELEMENT WORKATTR9 (#PCDATA)>
<!ELEMENT WORKCLOSEMODE (#PCDATA)>
<!ELEMENT WORKF (ECPMOD, NCFVERS, WFOPFA, WORK, WORKFILE1, WORKFILE2,
WORKFILE3, WORKFILE4, WORKFILE5, WORKFILE6, WORKFILE7, WORKFILE8,
WORKFILE9, WORKFILE10, WORKFILE11, WORKFILE12, WORKFILE13, WORKFILE14,
WORKFILE15, WORKFILE16, WORKFILE17, WORKFILE18, WORKFILE19, WORKFILE20,
WORKFILE21, WORKFILE22, WORKFILE23, WORKFILE24, WORKFILE25, WORKFILE26,
WORKFILE27, WORKFILE28, WORKFILE29, WORKFILE30, WORKFILE31, WORKFILE32,
WORKTYPE, WORKCLOSEMODE, WORKATTR1, WORKATTR2, WORKATTR3, WORKATTR4,
WORKATTR5, WORKATTR6, WORKATTR7, WORKATTR8, WORKATTR9, WORKATTR10,
WORKATTR11, WORKATTR12, WORKATTR13, WORKATTR14, WORKATTR15, WORKATTR16,
WORKATTR17, WORKATTR18, WORKATTR19, WORKATTR20, WORKATTR21, WORKATTR22,
WORKATTR23, WORKATTR24, WORKATTR25, WORKATTR26, WORKATTR27, WORKATTR28,
WORKATTR29, WORKATTR30, WORKATTR31, WORKATTR32)>
<!ELEMENT WORKFILE1 (#PCDATA)>
<!ELEMENT WORKFILE10 (#PCDATA)>
<!ELEMENT WORKFILE11 (#PCDATA)>
<!ELEMENT WORKFILE12 (#PCDATA)>
<!ELEMENT WORKFILE13 (#PCDATA)>
<!ELEMENT WORKFILE14 (#PCDATA)>
<!ELEMENT WORKFILE15 (#PCDATA)>
<!ELEMENT WORKFILE16 (#PCDATA)>
<!ELEMENT WORKFILE17 (#PCDATA)>
<!ELEMENT WORKFILE18 (#PCDATA)>
<!ELEMENT WORKFILE19 (#PCDATA)>
<!ELEMENT WORKFILE2 (#PCDATA)>
<!ELEMENT WORKFILE20 (#PCDATA)>
<!ELEMENT WORKFILE21 (#PCDATA)>
<!ELEMENT WORKFILE22 (#PCDATA)>
<!ELEMENT WORKFILE23 (#PCDATA)>
<!ELEMENT WORKFILE24 (#PCDATA)>
<!ELEMENT WORKFILE25 (#PCDATA)>
<!ELEMENT WORKFILE26 (#PCDATA)>
<!ELEMENT WORKFILE27 (#PCDATA)>
<!ELEMENT WORKFILE28 (#PCDATA)>
<!ELEMENT WORKFILE29 (#PCDATA)>
```

```
<!ELEMENT WORKFILE3 (#PCDATA)>
<!ELEMENT WORKFILE30 (#PCDATA)>
<!ELEMENT WORKFILE31 (#PCDATA)>
<!ELEMENT WORKFILE32 (#PCDATA)>
<!ELEMENT WORKFILE4 (#PCDATA)>
<!ELEMENT WORKFILE5 (#PCDATA)>
<!ELEMENT WORKFILE6 (#PCDATA)>
<!ELEMENT WORKFILE7 (#PCDATA)>
<!ELEMENT WORKFILE8 (#PCDATA)>
<!ELEMENT WORKFILE9 (#PCDATA)>
<!ELEMENT WORKTYPE (#PCDATA)>
<!ELEMENT XREF (#PCDATA)>
<!ELEMENT YEAR2000 (YSLW, DFOUT, DFSTACK, DFTITLE)>
<!ELEMENT YSLW (#PCDATA)>
<!ELEMENT ZD (#PCDATA)>
<!ELEMENT ZP (#PCDATA)>
```

# 60 DTD for INatAutoNatparm - Remote Environment

Applies to `INatAutoNatparm::ProfileParameters`.

For a remote environment, the XML document is structured according to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ADA (WH)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT CF (#PCDATA)>
<!ELEMENT CHARS (CF, DC, IA, ID)>
<!ELEMENT COMPOPT (DBSHORT, ENDIAN, GFID)>
<!ELEMENT COMPR (#PCDATA)>
<!ELEMENT DBSHORT (#PCDATA)>
<!ELEMENT DC (#PCDATA)>
<!ELEMENT DFOUT (#PCDATA)>
<!ELEMENT DFSTACK (#PCDATA)>
<!ELEMENT DFTITLE (#PCDATA)>
<!ELEMENT DTFORM (#PCDATA)>
<!ELEMENT DU (#PCDATA)>
<!ELEMENT EDITOR (EDTBPSIZE)>
<!ELEMENT EDTBPSIZE (#PCDATA)>
<!ELEMENT EJ (#PCDATA)>
<!ELEMENT ENDIAN (#PCDATA)>
<!ELEMENT ERROR (SA)>
<!ELEMENT FCDP (#PCDATA)>
<!ELEMENT FS (#PCDATA)>
<!ELEMENT GFID (#PCDATA)>
<!ELEMENT IA (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT IM (#PCDATA)>
<!ELEMENT LE (#PCDATA)>
<!ELEMENT LIMITS (LE, LT)>
<!ELEMENT LS (#PCDATA)>
<!ELEMENT LT (#PCDATA)>
<!ELEMENT MISC (CC, DU, FCDP, FS, IM, ML, NC, OPF, PD, REINP, SM, SYMGEN, TS, XREF, ZD)>
<!ELEMENT ML (#PCDATA)>
<!ELEMENT NATRPC (COMPR, TIMEOUT, TRYALT, RPC-DFS)>
<!ELEMENT NC (#PCDATA)>
<!ELEMENT OPF (#PCDATA)>
<!ELEMENT PD (#PCDATA)>
<!ELEMENT PM (#PCDATA)>
<!ELEMENT PS (#PCDATA)>
<!ELEMENT ProfileParameters (ADA, CHARS, COMPOPT, EDITOR, ERROR, LIMITS, MISC, REPO, NATRPC, YEAR2000)>
<!ELEMENT REINP (#PCDATA)>
```

```
<!ELEMENT REPO (DTFORM, EJ, LS, PM, PS, SF, ZP)>
<!ELEMENT RPC-DFS (#PCDATA)>
<!ELEMENT SA (#PCDATA)>
<!ELEMENT SF (#PCDATA)>
<!ELEMENT SM (#PCDATA)>
<!ELEMENT SYMGEN (#PCDATA)>
<!ELEMENT TIMEOUT (#PCDATA)>
<!ELEMENT TRYALT (#PCDATA)>
<!ELEMENT TS (#PCDATA)>
<!ELEMENT WH (#PCDATA)>
<!ELEMENT XREF (#PCDATA)>
<!ELEMENT YEAR2000 (DFOUT, DFSTACK, DFTITLE)>
<!ELEMENT ZD (#PCDATA)>
<!ELEMENT ZP (#PCDATA)>
```

# 61 DTD for INatAutoNatsvar - Local Environment

Applies to `INatAutoNatsvar::SystemVariables`.

For the local environment, the XML document is structured according to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT APPLIC-ID (#PCDATA)>
<!ELEMENT APPLIC-NAME (#PCDATA)>
<!ELEMENT DEVICE (#PCDATA)>
<!ELEMENT ERROR-TA (#PCDATA)>
<!ELEMENT ETID (#PCDATA)>
<!ELEMENT GROUP (#PCDATA)>
<!ELEMENT HARDCOPY (#PCDATA)>
<!ELEMENT HARDWARE (#PCDATA)>
<!ELEMENT HOSTNAME (#PCDATA)>
<!ELEMENT INIT-ID (#PCDATA)>
<!ELEMENT INIT-PROGRAM (#PCDATA)>
<!ELEMENT INIT-USER (#PCDATA)>
<!ELEMENT LANGUAGE (#PCDATA)>
<!ELEMENT LIBRARY-ID (#PCDATA)>
<!ELEMENT MACHINE-CLASS (#PCDATA)>
<!ELEMENT NATVERS (#PCDATA)>
<!ELEMENT NET-USER (#PCDATA)>
<!ELEMENT OPSYS (#PCDATA)>
<!ELEMENT OS (#PCDATA)>
<!ELEMENT OSVERS (#PCDATA)>
<!ELEMENT PARM-USER (#PCDATA)>
<!ELEMENT PATCH-LEVEL (#PCDATA)>
<!ELEMENT PID (#PCDATA)>
<!ELEMENT SERVER-TYPE (#PCDATA)>
<!ELEMENT STARTUP (#PCDATA)>
<!ELEMENT STEPLIB (#PCDATA)>
<!ELEMENT SystemVariables (APPLIC-ID, APPLIC-NAME, DEVICE, ERROR-TA, ETID,
GROUP, HARDCOPY, HARDWARE, HOSTNAME, INIT-ID, INIT-PROGRAM, INIT-USER,
LANGUAGE, LIBRARY-ID, MACHINE-CLASS, NATVERS, NET-USER, OPSYS, OS, OSVERS,
PARM-USER, PATCH-LEVEL, PID, SERVER-TYPE, STARTUP, STEPLIB, TP, TPSYS,
```

```
TPVERS, WINMGR, UI, USER, USER-NAME, WINMGRVERS)>
<!ELEMENT TP (#PCDATA)>
<!ELEMENT TPSYS (#PCDATA)>
<!ELEMENT TPVERS (#PCDATA)>
<!ELEMENT UI (#PCDATA)>
<!ELEMENT USER (#PCDATA)>
<!ELEMENT USER-NAME (#PCDATA)>
<!ELEMENT WINMGR (#PCDATA)>
<!ELEMENT WINMGRVERS (#PCDATA)>
```