NaturalONE

NaturalONE in a Nutshell

Version 9.3.1

February 2025

ADABAS & NATURAL

This document applies to NaturalONE Version 9.3.1 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: ONE-FIRSTSTEPS-931-20250213

Table of Contents

Preface

This documentation explains the basic concept of NaturalONE and helps you get started. It is organized under the following headings:

NaturalONE and the Eclipse Workspace	What is the advantage of using the workspace? Why is it recommended that you use local mode instead of Natural server mode? How do you get your sources from the Natural server into your Eclipse workspace? And other useful information.
All About Natural Projects	Describes the different possibilities of creating Natural projects.
Working in a Team	About versioning tools, private-mode libraries, and other information which is helpful when more than one developer works on one and the same application.
Performance Aspects	What actions and best practices can improve the performance? Especially when working with huge projects? This section has some answers.
The Local Natural Runtime	About the local Natural runtime which can be used for testing and debugging purposes, without having to access a Natural server.
The Development Lifecycle: Step by Step	Summarizes the required steps for the preferred development scenario.
Performance Analysis of Natural Applications	Describes how you can analyze the performance of Natural applications. Summarizes the basic required steps and provides hints on preferred analysis strategies.

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: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. 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://containers.softwareag.com/products** 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 GmbH 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.

2 NaturalONE and the Eclipse Workspace

Advantage of Using the Eclipse Workspace

NaturalONE is based on the concepts of Eclipse. It uses of a number of Eclipse features, which make up the product NaturalONE.

Eclipse makes use of a workspace into which you put your development work. NaturalONE and other tools downloaded from the Internet also make use of this workspace. The workspace acts as an interface between the different tools. In the Eclipse context, the tools are called “plug-ins” because they extend the functionality of Eclipse far beyond the default installation of Eclipse.

You maximize the productivity in your development lifecycle when your development work is done using the Eclipse workspace. This also provides the opportunity for many developers to work on the same application at the same time, without interfering each other.

Local Mode Versus Natural Server Mode

NaturalONE has a special name for working in the workspace. This is called “local mode”. In local mode, it is possible, for example, to store your sources in a version control system or to make use of other Eclipse-based Software AG products. Developers who are familiar with Java development in Eclipse can easily make use of the Natural tools and are therefore able to develop Natural-based applications within a very short time.

There is another mode, the so-called “Natural server mode”, which is very similar to Natural Studio. It offers a GUI environment for direct development on a Natural server. Keep in mind that this mode is *not* the preferred mode of NaturalONE, and that it is by far not as powerful as the local mode. Why? As mentioned above, when working in local mode, it is possible to use additional tools and functionality from other vendors. This is not the case when working in Natural server mode. In addition, in Natural server mode, each developer working on a library directly affects other developers working on the same library.

See also *Different Modes for Developing Natural Applications* in the *Introduction*.

Offloading your Sources from the Natural Server into the Workspace

The local mode is the preferred mode. When developing in Eclipse, the major difference between the local mode and the Natural server mode is that with local mode your sources have to be in the workspace. Compared to Natural development as it was done before the release of NaturalONE, this is very much of a “paradigm shift”.

When your applications are currently stored on a Natural server, you locate them in the **Natural Server** view and then add them either to a new project or an existing project in your workspace using the corresponding command in the context menu. This is only done once. When your applications are in the workspace, you may close the **Natural Server** view because it is literally no longer needed. From now on, you develop your applications in the workspace (in the **Project Explorer** view).

Cataloged Objects on the Server

In Natural, a cataloged object is the executable (compiled) form of a Natural source. Cataloged objects are not stored in the Eclipse workspace. When you build, rebuild or update your project, your sources are uploaded to the Natural environment and are compiled there.

With NaturalONE, you can manage the Natural environment directly from the workspace. For example, you can execute a program out of the workspace (even if just the Natural source is physically stored there). In this context, the term "Natural environment" includes the local Natural runtime which is installed with NaturalONE and the Natural server environment which is located on another computer. Since the local Natural runtime belongs directly to NaturalONE, it is not visible as a mapped environment in the **Natural Server** view. NaturalONE keeps the "link" between the source in the workspace and the cataloged Natural object. It knows in which Natural environment (local Natural runtime or Natural server) the appropriate Natural objects are stored and in which library the cataloged Natural objects are located. You may even start a debugging session without having the cataloged object in the workspace. The Natural source itself is sufficient for this purpose.

Application Parameters in the Project Properties

When you download an application from the **Natural Server** view to a project in the workspace, the application parameters are also downloaded. You will find them in the project properties. For example, there is information about the server connection itself (such as host name and port number). And there is also information, for example, on the regional settings, parser parameters or steplibs which are associated with the application. Therefore, if you develop or maintain Natural applications in the workspace, you use the same settings as in the Natural environment. When you build your project in the workspace, the sources are transferred to the Natural environment onto the appropriate Natural system file. The Natural environment is "configured" using the previously downloaded parameters and the sources are cataloged.

The Natural Builder

The workspace is managed by the so-called Natural builder which keeps the information about the application in the workspace. The builder recognizes whether a source has been changed and whether it needs to be cataloged on the server. The builder is able to distinguish between identically named objects which are located in different libraries. It knows the search sequence for any defined steplibs and thus knows where to find the Natural objects that are used by a Natural program.

If you change, for example, a copycode, the builder knows which Natural objects make use of this copycode. When you build your project, only the changed copycode is transferred to the server and only the dependent objects are cataloged. In this case, the builder acts like a very intelligent `CATALL` since it includes all steplibs. This is different from the `CATALL` system command as it is used on the server which only processes the objects in the current library.

Project Structure on the Disk

There is always the need to have an anchor for an application in the workspace. Regardless of whether you develop Java clients, Ajax pages or Natural applications, this anchor is the project. Because the workspace is located somewhere on your PC's disk, the workspace is also the root node of your project structure. What you see in the project can also be found on the disk, with the identical structure and naming. It is important to understand this because the versioning tools, for example, need to know the physical representation of the workspace on the disk.

Project References

Eclipse provides the possibility to reference projects. This means that project A may make use of objects in project B. The Natural builder understands this Eclipse functionality and treats each linked project as a different set of steplibs. Therefore, it is very handy to put common Natural application parts (such as data areas, DDMs or API subprograms) into one project and the application parts on which you are working into a separate project.

3 All About Natural Projects

How to Create a Natural Project?

As already mentioned in the previous chapter, the anchor of a Natural application is the Natural project. Without a project, it is not possible to create new Natural objects or to add objects from the Natural server to the Eclipse workspace.

How are projects created? NaturalONE offers several possibilities for this purpose. The most common are:

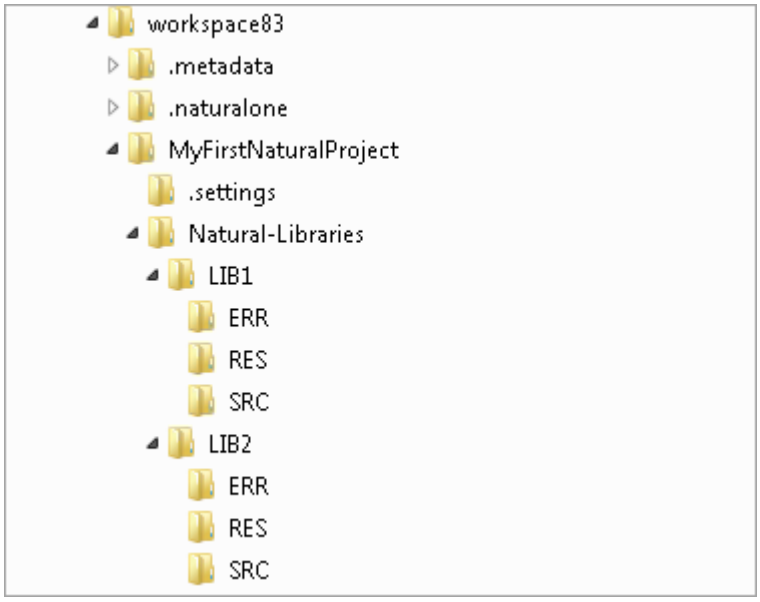
- Define a project from scratch on your own.
- Add existing Natural libraries and objects which are stored on a Natural server to a new project which is automatically created for the download.
- Check out an existing project from a versioning repository.
- Import an existing project from another location (for example, from another workspace or versioning repository).

This is described in the topics below.

Defining a Project from Scratch

What is a Natural project all about? A Natural project is defined in the Eclipse workspace.

The folders and files that you create for a project in the workspace can also be found in the file system, with the identical structure and naming. Therefore, the project is the root directory entry of your Natural application.



Information about the project is kept in the project properties. To see the current project settings, you select the project and choose the **Properties** command.

The project is used for two different purposes: to develop Natural applications in the workspace and to configure the associated Natural environment.

A project consists of the project node itself, the project properties, the Natural libraries and Natural objects, and several project-related configuration files.

Within the Natural libraries, you may have different subfolders in which specific components of the application are stored:

- Natural objects such as programs or subprograms are stored in the *SRC* folder.
- Application-related error messages are stored in the *ERR* folder.
- Resources required by the Natural application (for example, images or HTML files) are stored in the *RES* folder.

The naming conventions for Natural libraries also apply for NaturalONE libraries.

Depending on the type of project, whether the project is intended to be used for developing Ajax applications or web services, additional project entities are available.

If you want to define a new application, defining a project from scratch is the matter of choice. Using the New Natural Project wizard which can be invoked from the Eclipse menu bar, you define all required information for your Natural project. You create the required library and Natural objects, edit the objects in the corresponding Natural editors, prepare the API documentation with NATdoc, and then you are ready to test (that is, execute, test and debug) and to deploy your new application.

If you create a second library (let us call this library LIB2) which uses objects from the first library (LIB1), you have to define LIB1 as a steplib for LIB2. This is done in the library properties.

Adding Natural Libraries/Objects to a New Project

If you have existing Natural applications that are stored on a Natural server, this possibility is the matter of choice.

With the **Natural Server** view, you can directly access a Natural server. In this view, you "open" (map) your Natural environment to NaturalONE. The connection between NaturalONE and the runtime environment is TCP/IP-based. Therefore, you need TCP/IP access to the runtime environment (host name and related port number). On the server, a Natural Development Server (NDV) environment is required. Natural Development Server provides an API for NaturalONE which establishes the communication, understands the commands from NaturalONE and transfers data to NaturalONE. The underlying technology gives you seamless access to different Natural environments (mainframe, Linux and Windows) at the same time. With a single NaturalONE environment, you are able to develop applications for different Natural platforms. And you also have the advantage to use Eclipse for developing applications with different programming languages (such as Java).

To access a Natural server, you map this Natural server environment in the **Natural Server** view. You have to specify the host name, the port number and a user ID for this purpose. When you open the top-level node that is shown for the mapped server, you can see folders for your existing user libraries, Natural system libraries, and - depending on the platform or configuration - DDMs (see also *Contents of the Natural Server View* in *Using NaturalONE*). If you have a huge number of Natural libraries and Natural objects, you may define filters for both items. With an active filter, you gain performance when you expand the related nodes in the **Natural Server** view. You can easily detect the nodes for which a filter has been set: the icon for such a node contains an additional plus sign.

To download an existing library or object from the Natural server to your workspace, you use the **Add to New Project** command. You are asked to define a name for the Natural project. The Natural libraries and objects that you have selected are then downloaded to a new Natural project which is automatically created in the workspace. In addition to the Natural objects, a number of required Natural profile parameter settings is also downloaded to the project. This is necessary to "configure" the NaturalONE parser, for example, with the appropriate decimal character (DC), programming mode (SM) or code page setting (CP) - just to mention a few. You may check the parameter settings by opening the project properties. If you are going to change a profile parameter in the project properties, this change also influences the Natural server environment which is defined for this project. This functionality ensures that specific parameter settings are only defined once: in the right place, in the project.

If you are adding Natural objects from an environment which is not protected by Natural Security (NSC), the steplib information of your runtime environment is also downloaded and saved in the

project properties. In this case, it is not possible to assign steplib information on library level because this would not be taken into account in the Natural runtime environment. In an environment which is protected by Natural Security and where each secured library may have up to eight steplibs, the steplib information is stored in the workspace, in the library properties. The first download from the server defines the type of environment (protected or unprotected) for which you are developing. In the case of a protected environment, there is no possibility to define steplibs on project level (in contrast to an unprotected environment).

Working with a protected Natural environment requires specific Natural Security settings. For detailed information, see *Protecting the Natural Development Environment in Eclipse* in the *Natural Security* documentation, which is part of the Natural documentation. If the Natural runtime environment is not set up correctly, you may not be able, for example, to add objects to the workspace.

Regardless of where the steplib information is defined, NaturalONE makes use of this information. In the **Dependency** view, you can always see the libraries from which the objects are taken.

Checking Out an Existing Project from a Versioning Repository

In the following example, it is assumed that you are already an experienced NaturalONE user and that your Natural project is stored in the repository of a version control system. You are currently working on one PC (let us call it PC1), but now you want to work with your project on a different PC (PC2).

On PC2, you switch to the appropriate versioning perspective - in our example, Subversion (SVN) is used. Select the appropriate repository location and navigate within the source tree to the appropriate project. When you invoke the context menu command for checking out the project, the project is read from the versioning repository and is placed in your NaturalONE workspace.

When NaturalONE is used to store the application in the versioning repository, NaturalONE will ensure that specific configuration files are versioned as well. For this purpose, it is important that you check out the entire project as such from the repository, and not just parts of the project.

Importing an Existing Project from another Location

You may add parts of applications to your Natural project using the import feature of Eclipse. When you invoke the **Import** command, the import wizard is started.

If you want to import from Subversion (SVN), see ***Checking Out an Existing Project from a Versioning Repository***.

But you may also import an existing project from a different workspace.

Deploying a Project

After finalizing a development step, it may be necessary to put the project back into the commonly-used Natural environment. The deployment wizard is used to read the project from the versioning repository, to upload the source to the Natural environment and to catalog (compile) it in the Natural environment. Using the deployment wizard, you can generate a deployment package which can be executed without the use of Eclipse. For detailed information, see *Deploying Applications* in *Using NaturalONE*.

4 Working in a Team

Key Requirement: A Versioning Tool

The major advantage of NaturalONE is that it supports out-of-the-box development in a team. Using the local mode, several developers may work on the same application at the same time. They do not interfere each other because each developer has his own workspace. But how do you synchronize the work of each developer? A versioning tool is required for this purpose. The versioning tool allows you to check out a project into the workspace, work on the project, and merge the changes back into the versioning repository. Team support is based on such versioning tools. It is key. Without a versioning tool, it is almost impossible to work with many developers synchronously on one and the same application at the same time.

Examples of versioning tools that are easy to use in Eclipse are CVS, Subversion (SVN) and GIT. Eclipse supports versioning tools via the **Team** menu and support for CVS is already built in. See also *Using a Version Control System* in the *Introduction*.

You must also set up a team repository server which stores the versioned items. For more information, see the documentation of your preferred versioning system.

Private-mode Libraries

The versioning tool solves the problem of synchronizing several developers when working on the Natural source. But what about the cataloged objects (also called “generated programs” or “GPs”)?

Just to remind you: The application was downloaded from specific libraries in the Natural environment into the workspace (either into a new project or an existing project, depending on the command that was used in the **Natural Server** view). The same libraries as in the Natural environment are also used in the workspace. When an application in the workspace is built, the sources are cataloged in the Natural environment, in the same libraries as in the workspace. If many developers are working on the same library, it is possible that one developer overwrites (recatalogs) the changes made by another developer. You avoid this problem by switching on “private mode” in the properties of the Natural project. What does this mean?

Let us assume the following:

Two developers are working on the same application which is mapped to a Natural server environment (very similar to the local Natural runtime of NaturalONE). Each developer is changing the application in his own workspace. For testing purposes, each developer builds the project on the server. The Natural builder always checks whether a so-called “private-mode library” for a developer exists on the server (that is, whether there is a separate library for each developer). The naming convention for such a private-mode library can be defined in the Natural preferences. When private mode has been switched on, the source is cataloged in the private-mode library. When a developer invokes the **Execute** command for a source in the workspace, the builder ensures

that the appropriate cataloged object is executed on the Natural server. Thus, both developers may test their changes without interfering each other. In a Natural Security environment, the private-mode library inherits the same credentials as the original library by default. There is no need for a system administrator to define this private-mode library in Natural Security. However, the administrator can set so-called "development mode options" in Natural Security. These options determine how Natural Security controls the use of Natural server actions triggered by **Project Explorer** view actions. For detailed information, see *Protecting the Natural Development Environment in Eclipse* in the Natural Security documentation, which is part of the Natural documentation.

Note: The private-mode libraries are created automatically when building projects in private mode. It is not possible to change a private-mode library in the **Natural Server** view, and it is not possible to change any object in a private-mode library (for example, it is not possible to edit such an object). The corresponding commands in the context menu are unavailable (gray). For further information, see the description for **Private Mode** in *Changing the Project Properties*.

When all tests have been completed, each developer has to merge his changes into the versioning repository. Based on the final repository contents, the application has to be deployed to the Natural server. NaturalONE provides a deployment tool for this purpose. As a result of merging the changes, each developer has the same revision of the application in his own workspace (after an update from the repository). And, via deployment, the related cataloged objects are now available on the Natural server. By default, the objects are deployed to the application libraries, not to the private-mode libraries.

Experience shows that about 60 to 70 percent of changes on an existing application can be merged to some extent automatically, *without* user interaction. The remaining portion of merging the sources has to be done by the developer manually, but the versioning tools provide functionality which supports manual merging of sources.

Using Private-mode Libraries in Batch

When you execute a Natural batch application in private mode, the batch Natural must be aware of the library search order (LSO) used for the private mode application execution. This information is kept in a specifically designed Natural member, the so-called *LSO member*. The LSO member is saved in a special private mode library, referred to as *LSO container library*.

The following steps need to be performed:

- **Generate the LSO (library search order) container library (optional)**
  If any private-mode library has been created for the project, the Natural node of the project properties offers a button to generate the LSO container library. After the LSO container library has been created, its name is displayed. This name is required for the private-mode batch execution. For further information, see *Natural* in *Changing the Project Properties*.

- **Save the private-mode library search order**
  In the context menu of the project select **NaturalONE > Save Private-mode Library Search Order**. This saves the current private-mode library search order into the LSO member. If the LSO container library has not yet been generated, it is automatically created and its name can be found in the project properties as described above.
- **Modify the batch command input**
  Enter the following system command as first input into the primary command input data set `CMSYNIN` of your Natural batch application:

```
SYSLSO LSO-container-library
```

  Where *LSO-container-library* is the name of the LSO container library generated before.

Notes:

1. Whenever the private-mode library search order has changed, the private-mode library search order has to be saved again so that the batch Natural finds the correct content. The private-mode library search order changes for example when a new private-mode library is added or if you change the steplibs. In general, you are recommended to save the private-mode library search order right before you execute the batch application.
2. The LSO member is a Natural text member named `OLSO` in the LSO container library. You may list its content to see how the library search order is currently set. But you should not modify it because in this case it will no longer be usable.
3. NaturalONE offers a function to delete an LSO container library. For further information, see *Natural* in *Changing the Project Properties*.
4. As a system administrator, you do not have to define the LSO container library in Natural Security. For detailed information, see *Protecting the Natural Development Environment in Eclipse* in the *Natural Security* documentation, which is part of the Natural documentation.
5. When the `SYSLSO` command is executed in an NSC environment, the steplib settings in the logon library must match the entries in the `OLSO` member. Otherwise, the `SYSLSO` command will fail.

Application-Specific Messages in Private Mode

Message files can be created or downloaded to a private-mode project in the same way as described in *Creating Application-Specific Messages*. When the message file is uploaded, it is written to the corresponding private-mode library on the server. To read the message, applications should use the `USR3320N` API. This API considers private-mode libraries while reading application-specific messages.

Separating Natural Projects into Logical Parts

Referencing of projects is a good way to separate logical parts of an application. For example, you can define a project which contains just the DDM and data area sources. Assuming that these sources are not changed very often in an existing application, the separation gives the development team a better overview of the different parts of the application. Each developer can just use his own parts in a project and does not need to store these common parts in his own project.

Referencing of projects requires that the steplib chain is set accordingly.

Keeping the Development Environments Separate

There is always a 1:1 relationship between an Eclipse instance and a workspace. It is not possible to share one workspace among different Eclipse instances. This concept ensures that one developer is not disturbed by the work of any other developer. NaturalONE also follows this concept. If you look at your workspace on disc, you will see a directory named *.naturalone*. Within this directory, you will find the *fuser* directory which contains all Natural objects of the local runtime.

In the NaturalONE context, *fuser* is the root directory entry for the Natural environment. "FUSER" is a Natural term which stands for "Natural system file for user programs". On Linux and Windows platforms, the `FUSER` is located on disc. On mainframes, the `FUSER` is located in a database called Adabas. This is the reason why it is called a system file.

Having the `FUSER` located in your own workspace supports the idea of separating the development environments (that is, the workspaces in the Eclipse world). If you make use of a Natural server, you should work in private mode (see above). This also ensures that the development works of different developers are kept in separate places.

Storing the Project Settings

As mentioned earlier, NaturalONE keeps the Natural parameters that are used in the Natural environment also locally in the workspace. This ensures that the Eclipse environment and the Natural environment are configured identically. When the project on which you are working has been checked out from a versioning repository, you will also get the parameter settings for this application from the repository (provided that the corresponding files have been checked in). You can imagine that it would be dangerous to commit your own parameter changes to the repository (for example, if you change the `DC` parameter for some testing reasons) and then make them available for the entire development group, where no one knows (and does not expect) that, for example, `DC` has been changed.

To circumvent this issue, NaturalONE makes use of the temporary store of Eclipse. You may change any Natural parameter settings locally in your workspace. By default, they are not committed to the versioning repository with the changed Natural sources. After you have tested the changed parameter settings and they work fine, you have to make your parameter settings globally available. To do so, invoke the project properties, select **Natural** in the tree of the resulting dialog box, and then choose the **Store new Defaults** button. The next time you commit source changes to the repository, the changed parameter settings are also stored in the repository.

If you are working with a Natural environment, the locally stored parameters are used to configure the Natural environment (see also ***Application Parameters in the Project Properties***).

5 Performance Aspects

When working with NaturalONE, following these aspects can improve the performance - especially when working with huge projects:

Keep your Objects in Local Projects...

... instead of working from the **Natural Server** view.

The preferred working method with NaturalONE is the local mode, where you work with Natural projects in the Eclipse workspace (see also ***NaturalONE and the Eclipse Workspace***).

If you work in the local mode, the connections to the Natural development (NDV) server will be reduced to a minimum and you can take advantage of the variety of features provided for Natural projects and Eclipse projects in general. Only to mention a few:

- The **Natural Builder** keeps the information which sources have been changed. Moreover, it knows which other objects depend on the changed sources and allows to catalog only the relevant objects on the server.
- The **Natural Navigator** view offers several options for changing the arrangement of the Natural project's objects. Moreover, it is possible to define various types of filters.
- The **Deployment Wizard** collects all required application information (such as the access information for the versioning repository and the Natural server) and writes it to an Ant script which is used to start the deployment process. This Ant script makes the deployment task highly configurable and repeatable and allows you to run the deployment process unattended. For more information see section *Deploying Natural Applications* of *Using NaturalONE*.
- The **Dependencies View** shows the dependencies to other Natural objects that are referenced from the source in the active editor. It is described in the *Associated Views* sections that are available for each editor described in section *Using the Natural Editors* of *Using NaturalONE*.
- The **Debugger** which allows to debug the different types of applications that can be created with NaturalONE.
- The **Application Testing** component allows testing your application, creating Unit tests for your application and generating Ant scripts for starting these tests automatically.

Define Filters...

... to display only relevant objects.

If you are working with a huge number of objects within your libraries and projects, you should consider working with filters to reduce the number of visible Natural objects to the number of objects you are currently working with.

This can improve the performance dramatically.

Filters can be defined in the Natural Navigator view and the Natural Server view.

Let the Natural Builder Decide...

... which objects must be updated.

The Natural Builder keeps the information which objects must be updated and cataloged on the server.

If you have huge projects, it is faster to update only these objects instead of rebuilding the entire project. So, after changing a source of a Natural project you should better use the command **Build Natural Project** instead of **Update**, **Upload** or **Rebuild Natural Project**.

In case you are using Natural deployment, you should use the **Deployment Mode** Incremental instead of Full.

When a Natural Ant build file is used the first time for deployment, the complete project is deployed and a cache file (*cache* `_< deployFileName >_<projectName>`.*properties*) is created. This cache file will be used for the next incremental deployment and must not be deleted from the project.

Close Projects...

... which you do not currently work on.

Yes, it is that simple: Close all projects in your workspace, which you do not need for your current work.

Look Closer at Git Label Decorations

Using a source repository is a must in modern software development. If you are working with Git as your repository, projects will show decorators that indicate, in what state the source currently is.

However, the Git label decorations could cause general performance problems with all active label decorations, because if a source change occurs, the Git Eclipse plugin sends update events for all visible project tree nodes (could be Navigator, Natural Navigator or Project Explorer view). This causes a label decoration calculation for all these nodes from all plugins with activated label decorations.

The following hints can reduce the label decoration drawing performance problem. If possible, you should:

- reduce the number of visible project tree nodes and/or,
- work with the **Natural Navigator** view, which is faster than the Project Explorer view if you have a huge number of visible project tree nodes (> 10000) and/or,
- define filters in the **Natural Navigator** view to reduce the number of visible project nodes.

If all above mentioned tips are not feasible in your environment, you could also deselect the Git label decorations under the **Eclipse Preferences > Appearance > Label Decorations**. If you want to check your Git changes, use the Team Synchronize view or the Git Staging view in Eclipse. Moreover, there are several other tools which offer a very good Git support outside of Eclipse.

Improve Mainframe Access

When many libraries are available on the Natural system file, or a huge number of objects are contained in a library on the mainframe, the listing of these objects in the **Natural Server** view can take a long time. Natural provides the following hyperdescriptors that can significantly improve the database access required for this purpose:

- H1 – Fast read of Natural object names (requires Natural for Mainframes version 9.1.2)
- H2 – Fast read of Natural library names (requires Natural for Mainframes version 9.2.1)

If a hyperdescriptor is available, Natural Development Server (NDV) uses it for a high-performance read. The installation of the hyperdescriptor exit and the allocation of the associated hyperdescriptor are described in the following sections:

- Installing the Hyperdescriptor Exit
- Allocating the Hyperdescriptors
- Performance Example

Installing the Hyperdescriptor Exit

1. Identify the Natural system file to be improved.
2. Add the following ADARUN card to the Adabas startup job of the database which contains the Natural system file:

```
ADARUN HEX15=NATHX15
```

3. Copy the hyperdescriptor exit NATHX15 to an (APF-authorized) Adabas load library. For Natural for mainframe version, as of, 8.2.8 and 9.1.2 the hyperdescriptor exit is delivered as load fix (NAT828L001 and NAT912L001, respectively). For later versions it is delivered in the Natural load library PRD.NATvrs.MVSLOAD.
4. Restart the Adabas nucleus.

Allocating the Hyperdescriptors

1. To allocate the H1 hyperdescriptor, execute the Adabas ADAINV utility with the following settings:

```
ADAINV  INVERT FILE=fnr
ADAINV  HYPDE='15,H1,18,A,NU=LJ,LK,LL,LM'
```

   where *fnr* is the file number of the Natural system file to be improved.

2. To allocate the H2 hyperdescriptor, execute the Adabas ADAINV utility with the following settings:

```
ADAINV  INVERT FILE=fnr
ADAINV  HYPDE='15,H2,9,A,NU=LJ,LL,LO'
```

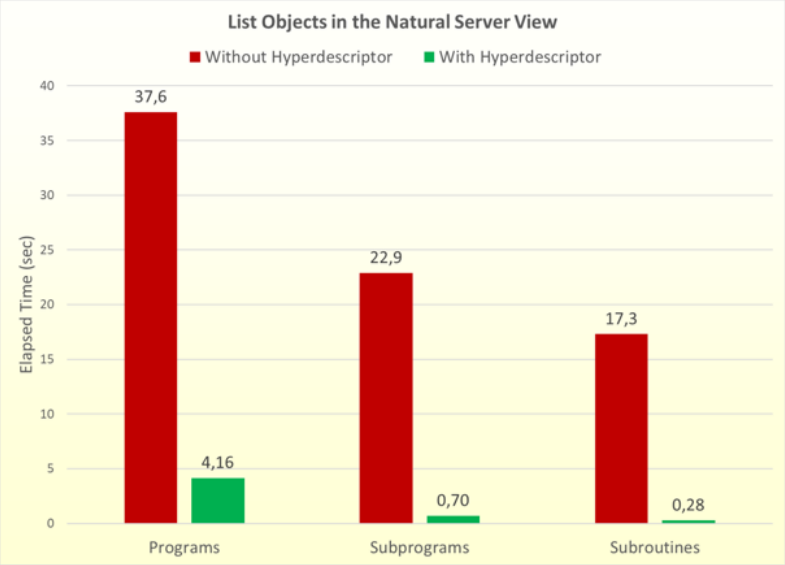
   where *fnr* is the file number of the Natural system file to be improved.

Repeat these steps for every Natural system file on which a hyperdescriptor should be used.

For further information regarding hyperdescriptors and the Adabas ADAINV utility, see the Adabas for Mainframes documentation.

Performance Example

In the following test, the program, subprogram and subroutine nodes in the **Natural Server** view are opened. The library contains in total about 15,000 objects, including 10,000 programs, 1,500 subprograms, and 100 subroutines. The image below shows the elapsed times of the NDV server component.



Checking the Availability of the Hyperdescriptors

The USR9203P example program in the SYSEXT library can be used to check whether the Natural hyperdescriptors are installed in the current FNAT or FUSER system file.

You can start the program from SYSEXT or directly in the Natural Tools and Utilities. To start the program in the Natural Tools and Utilities, invoke the context menu of a project in which you want to check the availability of the hyperdescriptors. Choose **NaturalONE** > **Tools and Utilities** > **Check Hyperdescriptor**.

Deleting a Hyperdescriptor

If a hyperdescriptor should not be used anymore, it can be deleted with the Adabas Online System (AOS). Proceed as follows:

1. Start **Natural Tools and Utilities** from NaturalONE and select AOS in the menu.

   Or start AOS on the mainframe server with the following Natural commands:

   ```
   LOGON SYSAOS
   MENU
   ```

2. Open the **File Maintenance**.

3. Select **Release Descriptor** with
   - the file number and DBID of the Natural system file in which the hyperdescriptor was defined, and
   - the descriptor name `H1` or `H2`.
4. Confirm the release (that is, deletion) of the specified descriptor.

If you want to use the hyperdescriptor again, you must rerun the Adabas `ADAINV` utility. If there is no Natural hyperdescriptor on the Natural system files anymore for a given database, you may remove the `ADARUN` cards for `NATHX15` from the Adabas startup job.

Maintaining the Natural System file

A Natural system file which contains the hyperdescriptor can be unloaded and reloaded with Adabas utilities. If you load the file, the hyperdescriptor exit must be known to the target database as described in the section *Installing the Hyperdescriptor Exit*. If the file is loaded and the `ADARUN` card is not defined, or the hyperdescriptor exit `NATHX15` is not found, you will receive the following message when modifying Natural objects or libraries:

```
NAT3079 Descriptor exit error occurred. DB/FNR/Subcode .../.../1.
```

In this case proceed as follows:

- If you need the hyperdescriptor in the target environment, follow the steps described in the section *Installing the Hyperdescriptor Exit*. A reallocation of the hyperdescriptor (Adabas `ADAINV` utility) is not required.
- If the hyperdescriptor is not wanted in the target file, you can delete it with the Adabas Online System as described in the section *Deleting the Hyperdescriptor*.

6 The Local Natural Runtime

NaturalONE is a preconfigured development environment which is able to assist you during the entire development lifecycle. Out of the box, NaturalONE provides the necessary application development tools, testing capabilities, automated documentation generation capabilities (NATdoc) and a Natural execution environment. As soon as you start NaturalONE, the local Natural runtime is available to you. You can use it to execute and debug Natural applications, without interfering other developers working on the same application. The NaturalONE workspace keeps the sources of your application, and the local Natural runtime is used for cataloging and executing objects. The major benefit of the local Natural runtime is that you can start with your local development shortly after installing NaturalONE.

However, if you already have a Natural server environment in use, then you can connect to this Natural server environment using the **Map** command of the **Natural Server** view. After offloading the application to your local workspace, you can start to develop this application offline. After finishing the development, you can put the developed parts of your application back to the Natural server to make them available to other developers. See also *Deploying a Project*.

7 The Development Lifecycle: Step by Step

The preferred way for developing or maintaining Natural applications is working in local mode. This means that all of your Natural sources have been offloaded from the Natural server to the Eclipse workspace, and that - in the ideal case - your sources have been added to the repository of a version control system. This section briefly summarizes the required steps for this scenario.

1. The following steps should be done by a single person, and only once:
   a. Offload your application sources from the Natural server (for example, from a mainframe) and add them to one or more projects in your Eclipse workspace. See also *Downloading an Existing Library or Object from a Natural Server* in *Using NaturalONE*.
   b. Commit the projects with the downloaded sources to the repository of your preferred version control system. See also *Using a Version Control System* in the *Introduction*.

      For information on additional files and folders that you also need to keep in your version control repository, see *Committing the Objects to the Repository of the Version Control System* in *Using NaturalONE*.
   c. Inform all involved developers that the application sources have been moved to the version control repository. From now on, these sources should no longer be changed directly on the Natural server.
2. All involved developers check out the required sources from the version control repository into their own Eclipse workspaces.
3. All developers define private mode for the projects which have been checked out of the version control repository. See also the description of the **Steplibs** property page in *Changing the Project Properties*, which is part of *Using NaturalONE*.
4. All developers change the sources as required in their own Eclipse workspaces. See also *Using the Natural Editors* in *Using NaturalONE*.
5. To execute or to debug a program, a compiled (also called "cataloged") object is required on the Natural server. Therefore, it is required to "build" the Natural project. All new and modified sources are then uploaded to the associated Natural server and are stowed there. In private

mode, the cataloged objects are stored in private-mode user libraries on the Natural server. These libraries belong only to one user. See also the following topics in *Using NaturalONE*:

- *Modifying Objects in the Natural Environment or in the Repository*
- *Executing Objects*
- *Using the Debugger*

6. When the source changes have successfully been tested, each developer commits the changed sources from the Eclipse workspace to the version control repository.
7. When the most current sources have been committed to the version control repository, each developer can delete the contents of the used private-mode libraries. This is done using the consolidate functionality. See also *Consolidating Objects in Private-Mode Libraries* in *Using NaturalONE*.
8. When all developers have committed their final source changes to the version control repository and all development on a Natural application is thus complete, the Natural application is ready to be deployed. It can be deployed directly from the repository of the version control system to the Natural server on which it is to be made available. During the deployment, the objects are put into the "real" application libraries, not into the private-mode libraries. See also *Deploying Applications* in *Using NaturalONE*.

8 Performance Analysis of Natural Applications

The Challenge

A very common and regularly asked question in any company or organization is, whether and how an existing application can be tuned to further increase its performance. Giving an appropriate answer could be difficult, because any kind of business logic can be quite complex. During execution, a lot of objects are passed and it is hard to decide what the control flow looks like.

Performance optimization or an analysis of such logic usually requires a deep understanding of the application, and also detailed information about the amount of time spent in the various parts of the application. Modifying the code manually to measure the time means a lot of effort and is usually discarded. But how can you retrieve this information?

Software AG provides a tool, the Natural Profiler to fill this gap. The Natural Profiler can provide you with a very fast overview about the time-consuming parts of a Natural application. No code modification is required, and moreover, just a basic knowledge of the application is sufficient. The Natural Profiler collects the required information from the Natural execution environment on Linux, Windows or mainframe platforms and presents it in a clear and simple manner.

How a performance analysis could be achieved is described in the following sections.

The Natural Profiler Approach

During a Natural session, different kinds of events may occur (for example "starting a program", "before a database call", etc.). The Natural Profiler - using an integrated routine in the Natural environment - intercepts the event and collects data specific to the event in a trace record. When a Natural application is profiled under NaturalONE, the event data is immediately interpreted by NaturalONE and visualized. When a Natural application is profiled outside NaturalONE (for example in Natural Batch), the event data is written to a Natural resource file which can be read and visualized in NaturalONE at any time later. See *Displaying Natural Profiler Resource Data* in the *Using NaturalONE* documentation for more information.

The resulting performance data is presented on the **Hot Spots** page. It shows how the CPU and elapsed time is distributed over the programs, statements and even program lines of the application, and how often a statement was executed. In the **Event Trace** page all events which have occurred during the Natural session, are listed.

Required Steps

The most important questions when tuning an application are: where are the most time-critical parts in my business logic? And, how do I get the information?

First of all, start profiling while you execute the application as usual. Within NaturalONE, you will see where the hotspots (the time-consuming parts of your application) are. Use this information to take the next iteration steps. Take a closer look into the time-consuming parts of the logic using the statement trace. If the collected information is still not sufficient, start logging the event trace. This is, of course, just an example - the steps may vary depending on the complexity of the investigated business logic.

How to Start Profiling

The easiest way to start profiling is offered by NaturalONE for programs available in the local mode. Use the context menu function **Profile As > Natural Application** which executes the program as usual and simultaneously starts the profiling. This is described in *Starting a Profiler Session* in the *Using NaturalONE* documentation.

How you start profiling outside NaturalONE depends on the platform on which the application is running:

- On Linux and Windows, profiling is started by specifying the `PROFILER` parameter in the parameter file or as a dynamic parameter. The `PROFILER` parameter is described in detail in the *Parameter Reference* section of the Natural documentation corresponding to your platform.
- On mainframes, profiling of batch applications is started by invoking the `INIT` and the `START` function of the Profiler utility in batch mode. For details, see the section *Profiler Utility - Batch Mode* in the *Utilities* section of the *Natural for Mainframes* documentation.
- Mainframe online applications can be profiled with the Profiler utility in online mode. The Profiler utility offers functions to view the event data online. The data can be saved as resource file and interpreted by NaturalONE. For details, see the section *Profiler Utility - Online Mode* in the *Utilities* section of the *Natural for Mainframes* documentation.

Interpreting the Profiler Results

When an application is profiled in NaturalONE or when a Profiler resource is opened, the Profiler displays the output on two general pages:

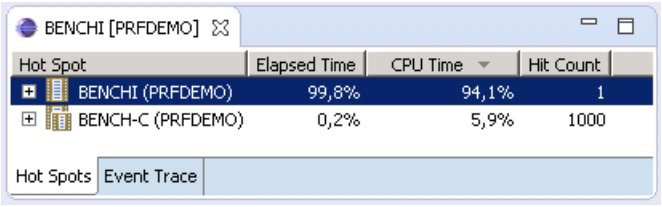
- **Hot Spots**
- **Event Trace**

The **Event Trace** page shows all events which have occurred during the monitored Natural session. This page is only displayed if the event trace option has been selected in the Natural preferences or if an *nprf* resource file is opened.

For a performance analysis, the **Hot Spots** page is of most interest. It shows the distribution of the elapsed time and CPU time of the objects executed. The elapsed time is the overall time elapsed in the real world while executing the profiled application. The CPU time is the time the processor requires to execute the application. In general, the CPU time is the better performance indicator because the elapsed time depends on varying factors such as on the machine load, I/O wait times, database calls, execution times, etc.

If you sort the **Hot Spots** page by the CPU time (click on the **CPU Time** entry in the header line), the objects which required the most CPU time are listed first. These are the objects which should be analyzed further for performance optimization.

In the example below, the program BENCHI consumed the most CPU time:



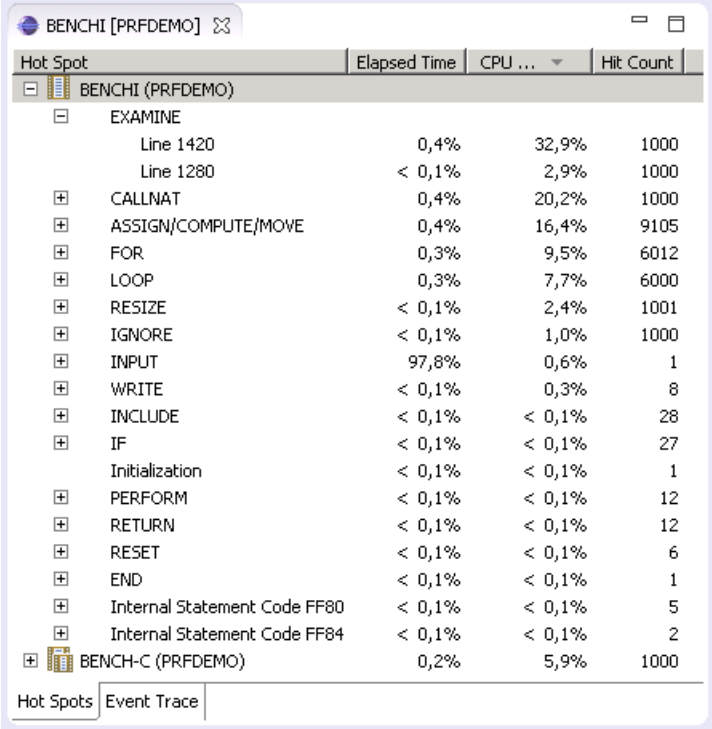
The Natural Profiler enables you to look closer into the profiled objects. If you expand the node of an object, it shows how the time consumed by the object is distributed over the statement types. If you further expand the node of a statement type, the Natural Profiler shows the elapsed and CPU time of individual lines in the source.

Notes:

1. Statement types and source lines are displayed in the **Hot Spots** page if the option **Profile statements** is enabled in the Natural preferences (see *Hot Spots* under *Profiler* in *Setting the Preferences*) and the compiler option `SYMGEN=ON` was set (for Windows and Linux objects). If the event data is read from a resource file, it is additionally required that statement events have been collected.

2. On Linux and Windows, the collection of statement events is triggered if the NS event type has been selected in the EVENT subparameter of the PROFILER parameter. See the *Parameter Reference* section of the Natural documentation corresponding to your platform.
3. On mainframes, the collection of statement events is triggered if STATEMENT=ON or STATEMENT=COUNT has been selected for the event filter. See the section *Event Filter* under *Profiler Utility - Batch Mode* in the *Utilities* section of the *Natural for Mainframes* documentation.

In the above example, when expanding the node of the program BENCHI you can see that the most CPU time (35.8%) was required by the EXAMINE statements. If you further expand the **EXAMINE** node, you can finally find that line 1420 had the highest CPU time consumption:



If the application was profiled with NaturalONE, you can double-click on a line entry. NaturalONE will then open the corresponding source and navigate directly to the selected line. In the example, if you double-click on line 1420, NaturalONE opens the program BENCHI and positions to the line with the time-consuming EXAMINE statement:

```
VALUE 'Xarray100_Examine'
  FOR #I = 1 TO LOOPS
    #A := #I
    EXAMINE #XARRAY(*) FOR #A GIVING INDEX IN #J
  END-FOR
```

Notes:

1. The line numbers in the **Hot Spots** page refer to the Natural line numbers. The direct navigation works similar to the **Go to Natural Line** command described in *Going to a Specific Natural Line Number* under *Using the Source Editor* in the *Using NaturalONE* documentation.
2. The direct navigation is not supported for event data read from Profiler resource files.

Profiling Long-Running Applications

Profiling a long-running application can produce huge amounts of data, especially when Natural statements are monitored. The more data the Profiler generates, the more time is required to transport the data from the server to the NaturalONE client. This section describes how to minimize the amount of data while still monitoring what matters. The following strategies are covered:

- Server-Side Data Consolidation
- Sampling
- Event Selection
- Start and Pause Profiling
- Combining Strategies

Server-Side Data Consolidation

When a Natural application is profiled, the Natural nucleus collects one record for each event. These are the records displayed in the **Event Trace** page. The **Hot Spots** page shows the data in a consolidated (aggregated) format. NaturalONE summarizes the event data so that, for example, instead of 1000 entries of the same line, only one hot-spot entry is displayed. This consolidated entry shows a hit count of 1000 and reflects the total elapsed and CPU time consumed by all 1000 traced events combined.

If no event trace is required, the data can already be consolidated on the server side which heavily increases the flow-rate.

- For a server-side data consolidation of a Linux or Windows application profiled under NaturalONE, unselect the **Enable event trace** option in the Natural Event Trace preferences as described in *Event Trace* under *Profiler* in the section *Setting the Preferences*.
- For a server-side data consolidation of a Linux or Windows application profiled outside NaturalONE, switch off the `EVENTTRACE` subparameter of the `PROFILER` parameter. See the *Parameter Reference* section of the Natural documentation corresponding to your platform.
- There is no server-side data consolidation of mainframe applications profiled under NaturalONE.
- For a server-side data consolidation of a mainframe application profiled outside NaturalONE, switch on the `CONSOLIDATE` keyword of the Profiler utility `INIT` function. See *Initializing Profiling* in the *Natural Profiler* section of the *Utilities* documentation, which is part of the *Natural for Mainframes* documentation.

- If you have profiled an application outside NaturalONE without the server-side data consolidation, you can consolidated the data later with the Profiler utility `CONSOLIDATE` function which is available on all platforms.

Sampling

Natural Profiler sampling is a statistical approach that significantly reduces the number of events. For every sampling CPU-time interval, the Natural Profiler collects only one event (the last one) and discards all others. The sampling provides a good estimation of the consumed CPU time. It does not give other estimations such as hit counts, elapsed times, Adabas times, etc.

- For applications profiled under NaturalONE, select the **Enable sampling** option in the Natural Hot Spots preferences as described in *Hot Spots* under *Profiler* in the section *Setting the Preferences*.
- For Linux or Windows applications profiled outside NaturalONE, switch on the `SAMPLING` subparameter of the `PROFILER` parameter. See the *Parameter Reference* section of the Natural documentation corresponding to your platform.
- For mainframe batch applications, specify `SAMPLING=ON` for `INIT` function of the Profiler utility. For detailed information about sampling see the section *Sampling* under *Profiler Utility - Batch Mode* in the *Utilities* section of the *Natural for Mainframes* documentation.

Event Selection

Statement events have the biggest impact on the performance and quantity of the Profiler data. In comparison to statement events, all other events have only a low impact on the performance but they can also enlarge the quantity. It is, therefore, recommended to monitor statement events only if you really need them. Of the other events, you should only monitor those which you want to analyze.

- For applications profiled under NaturalONE, unselect the **Profile statements** option in the Natural Hot Spots preferences and the **Natural Statement** option in the Natural Event Trace preferences as described in *Hot Spots* and *Event Trace* under *Profiler* in the section *Setting the Preferences*.
- For Linux or Windows applications profiled outside NaturalONE, unselect the `NS` event type in the `EVENT` subparameter of the `PROFILER` parameter. See the *Parameter Reference* section of the Natural documentation corresponding to your platform.
- For mainframe batch applications, specify `STATEMENT=OFF` for the event filter. See the section *Event Filter* under *Profiler Utility - Batch Mode* in the *Utilities* section of the *Natural for Mainframes* documentation.

Start and Pause Profiling

In general, you can pause and restart the Profiler data collection from the profiled Natural application by calling the application programming interface (API) `USR8210N`. See the *Application Programming Interface* section in the *Using NaturalONE* documentation.

For applications profiled outside NaturalONE, you can use the Natural programs delivered in the SYSPRFLR library to pause and restart the data collection. See *Using Profiler Delivered Programs* in *Starting and Pausing Data Collection* under *Profiler Utility - Batch Mode* in the *Utilities* section of the *Natural for Mainframes* documentation. The programs in the SYSPRFLR library are available on all platforms.

Combining Strategies

For monitoring long-running applications, it can be helpful to combine some of the above strategies. You should keep in mind that the consolidation works best for program loops, whereas sampling works in any situation but should only be used for CPU analysis.

- In a first approach statements are not monitored. The server-side consolidation is used if available. On the **Hot Spots** page you can see which program has consumed the most time and the program hit counts are also valid.
- In a second profiler run, you can monitor the statements and use the start and pause functions to collect only data of those objects with the highest CPU times. Use sampling and if available, the server-side consolidation. On the **Hot Spots** page you can see the program lines with the highest CPU times.