Aphelion Java Toolkit Getting Started Guide

Release 1.1

DOC 0507-01

Contents

Preface

For More Information

For more information on the features of LynxOS, refer to the following printed and online documentation.

- *LynxOS Release Notes*

  This printed document contains late-breaking information about the current release.

- *LynxOS Installation Guide*

  This manual supports the initial installation and configuration of LynxOS and the X Windows System.

- *LynxOS User's Guide*

  This document contains information about basic system administration and kernel level specifics of LynxOS. It contains a "Quick Starting" chapter and covers a range of topics, including tuning system performance and creating kernel images for embedded applications.

- Online information

  Information about commands and utilities is provided online in text format through the **`man`** command. For example, a user wanting information about the GNU compiler would use the following syntax, where **`gcc`** is the argument for information about the GNU compiler:

  ```
  man gcc
  ```

More recent versions of the documentation listed here may also be found online.

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Kind of Text	**Examples**
Body text; *italicized* for emphasis, new terms, and book titles	Refer to the *LynxOS User's Guide.*

Environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data	`ls`
	`-l`
	`myprog.c`
	`/dev/null`

Commands that need to be highlighted within body text, or commands that must be typed as is by the user are **`bolded`**.	`login:` **`myname`**
	`#` **`cd /usr/home`**

Text that represents a variable, such as a file name or a value that must be entered by the user	`cat` *`filename`*
	`mv` *`file1 file2`*

Blocks of text that appear on the display screen after entering instructions or commands	`Loading file /tftpboot/shell.kdi`
	`into 0x4000`
	`.....................`
	`File loaded. Size is 1314816`
	`Copyright 2000 LynuxWorks, Inc.`
	`All rights reserved.`
	
	`LynxOS (ppc) created Mon Jul 17`
	`17:50:22 GMT 2000`
	`user name:`

Keyboard options, button names, and menu sequences	**Enter**, **Ctrl-C**

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

LynuxWorks Technical Support is available Monday through Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The LynuxWorks World Wide Web home page provides additional information about our products.

LynuxWorks U.S. Headquarters

Internet: `support@lnxw.com`
Phone: (408) 979-3940
Fax: (408) 979-3945

LynuxWorks Europe

Internet: `tech_europe@lnxw.com`
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

World Wide Web

`http://www.lynuxworks.com`

CHAPTER 1 Introduction

Overview

The Aphelion Java Toolkit for LynxOS provides developers the ability to create, debug, and compile Java and C/C++ applications for LynxOS embedded environments.

The Aphelion toolkit includes the following components:

- Aphelion Integrated Development Environment (IDE)
- Apogee's Java Compiler
- Apogee's Java Native Method Optimizer
- Java 2 Micro Edition Connected Device Configuration Java Virtual Machine (J2ME CDC JVM)

The Aphelion Integrated Development Environment (IDE) uses a similar look-and-feel of Microsoft Visual Studio. With Aphelion, users can create and manage projects, edit source code files, compile projects, and debug applications all from a Windows host system.

When a project is ready to be compiled, the Aphelion IDE uses the Apogee Java compilers Java into standard class files that can run on any Sun-compliant Java Virtual Machine. The IDE also uses the Apogee compiler to create a Java Virtual Machine.

Aphelion Debugger

The Aphelion integrated debugger uses the look-and-feel of Microsoft Visual Studio and allows users to debug Java applications across a network.

The Aphelion debugger also provides the ability to debug across the byte-code/native-code boundary. The transition between a byte-code method that calls a native code method is handled transparently by the debugger. For example, switching between the byte-code method `foo()`, and the native method `bar()` updates the disassembly display based on the implementation of the method (Java, or C/C++). The call stack displays the variables & disassembly information in byte-code context, or native-code context. Stepping into a native-code from a byte-code, or vice-versa automatically switches the debugger between diassembly displays.

CHAPTER 2 *Installing and Starting*

Before Installing

The Aphelion Java Toolkit for LynxOS includes components that must be installed on both the Windows host and Target system before users can develop Java applications.

Supported Hosts and Targets

The following cross development hosts and targets are supported:

Table 2-1: Supported Host and Target Systems

Supported Host Platform	Supported Target Systems
Windows 2000	LynxOS 4.0, Beta 2 -x86

System Requirements

The following system requirements must be met before installing Aphelion IDE:

Table 2-2: System Requirements

Requirement	Host	Target
Hard Disk	15 MB	5 MB
RAM	32 MB	20 MB for CDC J2ME

Installation Instructions

Aphelion must be installed on top of a pre-existing LynxOS Windows Cross Development Environment.

Install Aphelion by double-clicking on the `setup.exe` icon from the CD-ROM. Follow the InstallShield instructions to complete the installation.

NOTE: It is important to set the correct path to the LynxOS Cygwin tools during initial installation of Apogee's Aphelion. Failure to set the correct paths results in linking errors and incomplete builds.

After Installing

Several post-installation steps are required to configure both the host and target systems before using Aphelion. The following sections describe the required configurations.

NOTE: Aphelion must have direct access to the Java project files, source code, and project libraries on the LynxOS target system. Aphelion must be able to read and write to the LynxOS filesystem where these files are kept. Exporting the LynxOS filesystem via Samba is required. See "Configuring Samba for LynxOS" on page 7.

LynxOS Target System Configuration

The following sections describe the configuration steps required on the LynxOS target system.

Creating an Apogee Directory

Users should create a project directory for Apogee target components required by Aphelion. This is the directory that is exported to the Windows host. For example, `/usr/apogee/`. The project directory requires that the read, write, and execute bits are set. Use the following commands to create and set the permissions for a project directory:

```
# mkdir /usr/apogee
# chmod 775 /usr/apogee
```

Adding Required Java Components to the Target

Before building and debugging Java applications with Aphelion, users must install several Java system class files and utilities to the target system. These target files are available as tar archives in the `target` directory of the Apogee installation directory. The required target system tar archives are:

- `RunTime.tar`
- `LynxOSdebugger.tar`

Use the following instructions to properly setup the target system Java components:

1. Copy these tar archives to the Apogee directory on the LynxOS target system with FTP, Samba (see “Configuring Samba for LynxOS” on page 7), or another utility.
2. Change to the Apogee directory.

   ```
   # cd <apogee_dir>
   ```

   Where <*apogee_dir*> is the Apogee Directory (`/usr/apogee`, for example).
3. Untar the `RunTime.tar` archive.

   ```
   # tar xvf <path>RunTime.tar
   ```

   Where <*path*> is the location of the `RunTime.tar` file. The runtime components are installed in the current directory.
4. Change to the `/usr/bin` directory and untar the `LynxOSdebugger.tar` archive.

   ```
   # cd /usr/bin
   # tar xvf <path>LynxOSdebugger.tar
   ```

   Where <*path*> is the location of the `LynxOSdebugger.tar` file. The debugging components are installed in the `/usr/bin` directory.

The following files are copied to the target system:

Table 2-3: Target System Files

Tar Archive	Path	Filename
RunTime.tar	*<apogee_dir>/apgoee/*lib/	foundation.jar
	*<apogee_dir>/apogee/*lib/security	java.policy java.security
	<apogee_dir>/apogee/samples/src	Hello/HelloWorld.java Loops/Fibonacci.java Loops/FloatingPoint.java Loops/IntegerALU.java Loops/Loops.java Loops/TestCase.java Loops/Timer.java NativeArgs/CmdLine.java NativeArgs/native_code.c
	<apogee_dir>/apogee/samples/projects	readme.txt
LynxOSdebugger.tar	/usr/bin/	aphgdb remoteaphgdb

Adding .rhosts & /etc/hosts.equiv file

Users must add a `.rhosts` file to their LynxOS home directory, enabling Aphelion to issue remote commands to the target. A sample `.rhosts` file is shown below:

```
trustedhost                     trusteduser
windows1.lynuxworks.com         dan
windows1.lynuxworks.com         root
```

Figure 2-1: Example .rhosts file

Update the `hosts.equiv` to add the name of the Windows host. For example:

```
#
# $Date: 1996/12/09 23:41:16 $
# $Revision: 5.2 $
#
lynx
lynx10b
lynx386a
lynx386b
lynx386c
windows1
```

Figure 2-2: Example /etc/hosts.equiv file

Configuring Samba for LynxOS

The target system Java project files, Java libraries, and other project files must be accessible from a Windows host via an exported filesystem. The simplest way to do this is with Samba.

Install Samba on the LynxOS host (if required). Edit the `smb.conf` file to add the shared filesystem and start the samba daemons.

The following steps provide an example samba configuration:

1. If not already installed, run the `Install.samba` script to install the Samba components.

   ```
   # Install.samba
   ```

2. Create an `smb.conf` file to configure Samba for the network. This file must exist in the `lib/` directory of the Samba installation directory

(default is `/usr/samba/lib`). A sample `smb.conf` file is provided below.

```
[global]
   workgroup = lynuxworks
   wins server = pdc.lynuxworks.com
   wins support = no

[tmp]
   comment = Temporary file space
   path = /tmp
   read only = yes
   public = yes

[public]
   path = /usr/apogee
   public = yes
   only guest = yes
   writable = yes
   printable = no
   case sensitive = yes
   preserve case = yes
```

Figure 2-3: Sample smb.conf file

NOTE: The parameters `case sensitive` and `preserve case` included in `public` must be set to `yes`. If case sensitivity is disabled, errors may occur when accessing or writing files.

3. Start the Samba daemons `smbd`, and `nmbd`.

```
# /usr/samba/bin/smbd -D
# /usr/samba/bin/nmbd -D
```

See the *LynxOS Networking Guide* and the Samba man pages for additional configuration information.

Once Samba is configured on the LynxOS system, map a network drive on the Windows system to the LynxOS shared filesystem. For more information, see “Windows Configuration Instructions” on page 9.

Increasing the Default Data Size in /etc/starttab

The Aphelion debugger requires a larger data size than the default LynxOS configuration. Edit the `/etc/starttab` file and change the default data size as follows:

```
# Data, stack, and core file limits (in Kbytes)
16384
```

Change value to:

```
131072
```

After updating the value, reboot the LynxOS system.

Windows Configuration Instructions

From the Windows host where Apogee’s Aphelion is installed, map a network drive to the LynxOS shared filesystem.

1. Open Windows Explorer.
2. Select **Tools**->**Map a Network Drive**.
3. Select the Drive Letter and LynxOS target system directory.
4. Click **OK**.

Licensing

The Aphelion Java Toolkit for LynxOS must be licensed with the Globetrotter FLEXlm License Manager. For a detailed description of the FLEXlm licensing software, refer to the `http://www.globetrotter.com` web site.

License File

The license file `license.dat` contains a specific key required by FLEXlm to license the product. Customers must contact LynuxWorks Customer Support for the appropriate keys to be included in the `license.dat` file.

Before calling LynuxWorks Customer Support, users must have their host ID number available. See the next section for information on generating a unique host ID.

Launching the Host ID Generator

In order to obtain a license from LynuxWorks, users must obtain a unique host ID of their system. The `lmhostid.exe` executable generates a unique host ID number that is used to generate a license key for the `license.dat` file.

To obtain a host ID, use the following instructions:

1. Open a DOS prompt and change to the Apogee Installation `bin\` directory on the Windows host. For example,

```
C:\ cd Program Files\Apogee\Aphelion CDC for LynxOS x86\bin
```

2. Run `lmhostid.exe` with the Disk Serial Number generator option:

```
C:\ lmhostid.exe -vsn
```

```
C:\Program Files\Apogee\Aphelion CDC for LynxOS x86\bin>lmhostid.exe -vsn
lmhostid - Copyright(C) 1989-1999 Globetrotter Software, Inc.
The FLEXlm host ID of this machine is “DISK_SERIAL_NUM=30e93320”
```

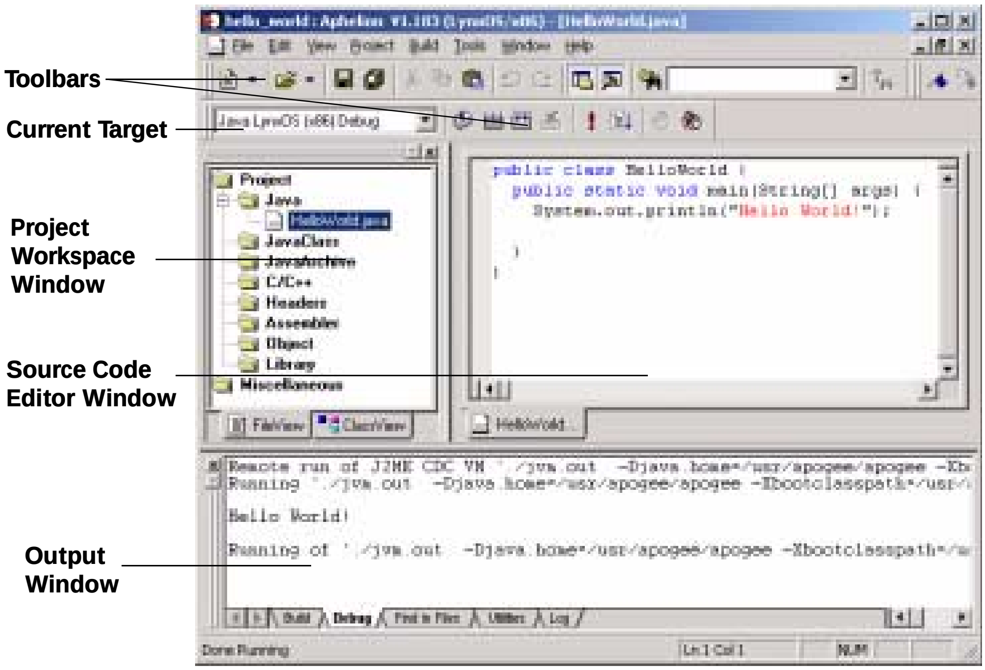
Figure 2-4: lmhostid Output

When contacting LynuxWorks for the license keys for the `license.dat` file, use the host ID number displayed.

Starting Apogee Aphelion

Start Apogee Aphelion by double-clicking on the **Apogee Aphelion CDC** icon on the desktop, or click **Start -> Programs -> Apogee Aphelion -> Aphelion CDC for LynxOS x86**.

The following figure details the main window of the Aphelion IDE:



```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");

  }
}
```

```
Remote run of J2ME CDC VM './jvm.out  -Djava.home=/usr/apogee/apogee -Xb
Running './jvm.out  -Djava.home=/usr/apogee/apogee -Xbootclasspath=/usr/

Hello World!

Running of './jvm.out  -Djava.home=/usr/apogee/apogee -Xbootclasspath=/u
```

Figure 2-5: Aphelion IDE Interface

Toolbars	Provides quick access to many IDE functions, including file options, build options, and debugging options.
Current Target	Displays the current configuration for the project (Debug/Release) to be used when building.
Source Code Editor Window	Displays the source or class file information selected in the Project Files Window. Here, users can edit source code and update files.
Project Workspace Window	Displays the source code files included in the project. Also displays the Java classes.
Output Window	displays system messages on compilation status, debugging information, and error logs.

Toolbar Buttons

Some of the Toolbar buttons are described below:

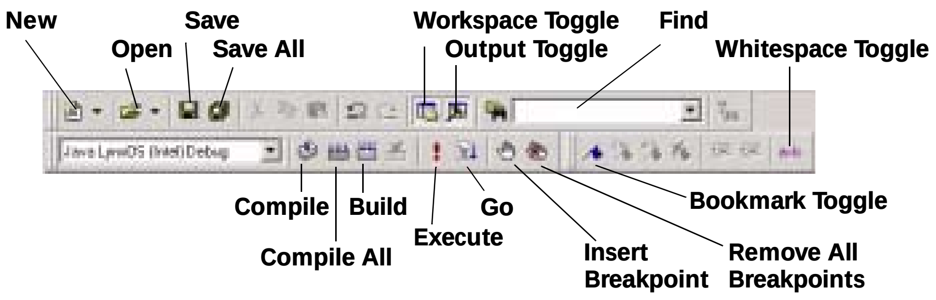


Figure 2-6: Toolbar Buttons

New	New Document or Project
Open	Opens Document or Project
Save	Saves Document or Project
Save All	Saves all Documents and Projects
Workspace Toggle	Toggles Workspace pane on or off
Output Toggle	Toggles Output pane on or off
Find	Find keyword in a file or directory
Compile	Compiles current file only
Compile All	Compiles all files
Build	Rebuilds project
Execute	Rebuilds (if needed) and executes project on the target
Go	Start Debugging
Insert/ Remove Breakpoint	Inserts or removes breakpoints to/from source
Remove All Breakpoints	Removes all breakpoints
Bookmark Toggle	Toggles Bookmark insertion at cursor point
Whitespace Toggle	Toggles indent symbols in Source Code Editor Window

Debug Toolbar Buttons

The Debugging Toolbar buttons are described below:

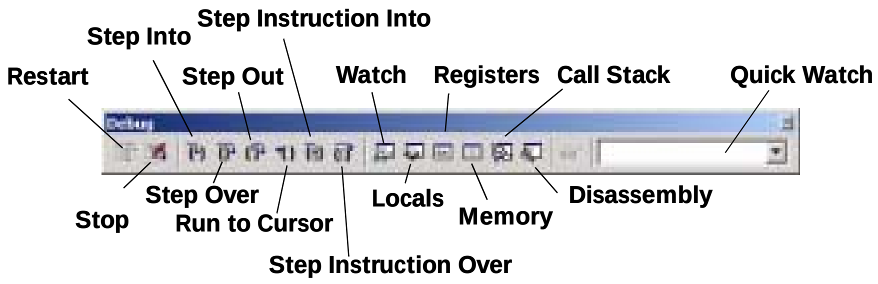


Figure 2-7: Debugging Toolbar

Restart	Restart Debugging
Stop	Stop Debugging
Step Into	Step into function
Step Over	Step over function
Step Out	Step out of function
Run to Cursor	Run program to cursor point
Step Instruction Into	Step into assembly or byte-code line instruction
Step Instruction Over	Step over assembly or byte-code line instruction
Watch	Open Watch Window
Locals	Open Local Variable Window
Registers	Open Register Window
Memory	Open Memory Window
Call Stack	Open Call Stack Window
Disassembly	Open Disassembly Window
Quick Watch	Open Quick Watch Window

CHAPTER 3

Creating Java Projects

Creating a Project

There are two types of projects that users can create in the Aphelion Java Toolkit for LynxOS:

- **Java** --The Java LynxOS option creates Java class files and a JVM from Java source files and C/C++ native methods.
- **C-C++**--The C-C++ LynxOS option is specific to creating C or C++ projects *only.* This option is used to create non-Java projects with Aphelion.

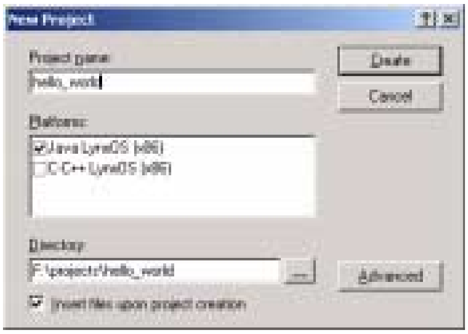
The following section provides a step by step example of creating and compiling a Java project with Aphelion.

Creating a Simple Java Project -- Hello World

This section describes the steps required to create a simple Hello World application in Java.

1. Start Aphelion by double-clicking on the Aphelion Icon on the desktop, or clicking **Start -> Programs -> Apogee Aphelion -> Apogee Aphelion CDC for LynxOS x86**.
2. In the Aphelion interface, click **File**-> **New Project.**

The following window appears:



Project name:
hello_world

Platforms:
☑ Java LynxOS (x86)
☐ C/C++ LynxOS (x86)

Directory:
F:\projects\hello_world

☑ Insert files upon project creation

Figure 3-1: New Project Window

Type in a name for the project in the **Project name** field (hello_world, for example), and select the platform of the project. For the purposes of this example, select Java LynxOS (x86).

Also, in the **Directory** field type in the name of the LynxOS mounted filesystem.

3. Click the **Create** button when finished.
4. The user is prompted to add Java source files to the project:

   Users can Click the **Cancel** button, as this example demonstrates creating Java source within the IDE.

The Aphelion IDE main window appears with the default settings for the hello_world project:

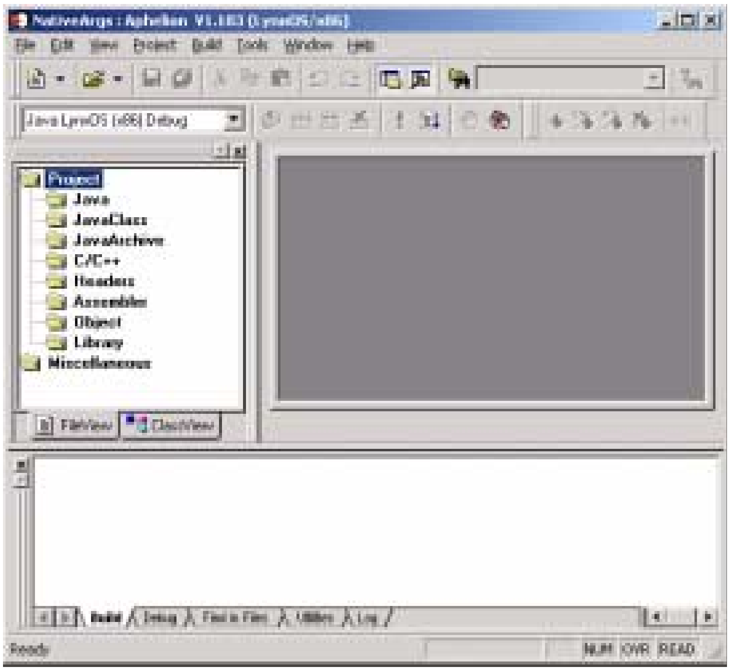


Figure 3-2: Aphelion IDE Main Window

Adding and Editing Source Files

To create and add a Java source file to the project, use the following instructions:

1. Click **File**-> **New** to create a new file. An empty file called **Untitled** appears in the Source Code Editor Window.
2. Type in the following code in the Source Code Editor Window:

```
public class HelloWorld {
     public static void main(String args[]) {
          System.out.println("Hello World!");
     }
}
```

Figure 3-3: Java Hello World Application

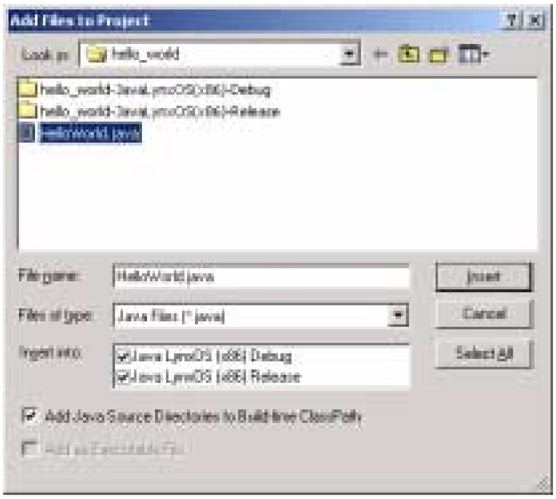
The source for this Hello World application is included with the Apogee product in the `samples/src/Hello` directory in the Apogee Installation directory.

3. Save the project by clicking **File** -> **Save as...** Save the file in the `hello_world` directory as `HelloWorld.java`.

NOTE: It is important to note that the Java source code filename match the Java Class name.

---

4. Add the `HelloWorld.java` file to the Java source code directory by clicking **Project**->**Add Files**.



Look in: hello_world

hello_world-Java(LynxOS(x86))-Debug
hello_world-Java(LynxOS(x86))-Release
HelloWorld.java

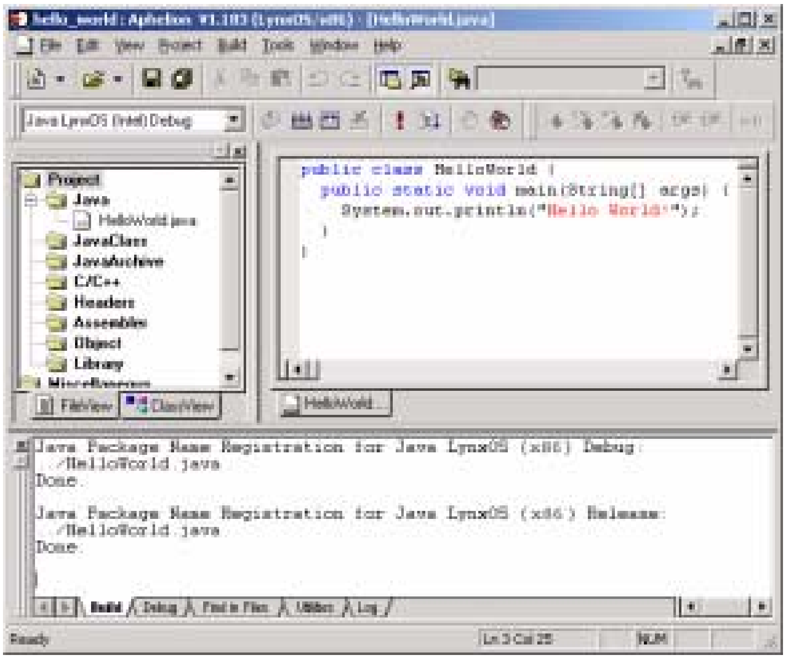
File name: HelloWorld.java
Files of type: Java Files (*.java)
Insert into: ☑ Java LynxOS (x86) Debug
☑ Java LynxOS (x86) Release

☑ Add Java Source Directories to Build-time ClassPath

Figure 3-4: Add Files to Project Window

5. Select the `HelloWorld.java` file and click **Insert**.

6. The `HelloWorld.java` source code is added to the project:



```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

```
Java Package Name Registration for Java LynxOS (x86) Debug:
  /HelloWorld.java
Done.

Java Package Name Registration for Java LynxOS (x86) Release:
  /HelloWorld.java
Done.
```

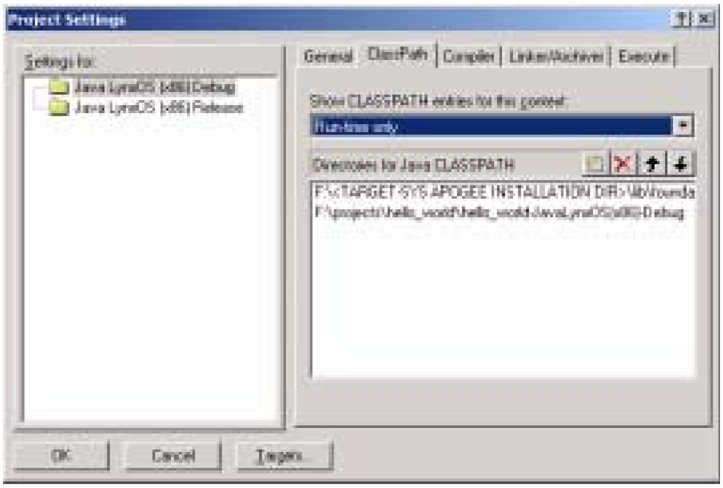
Figure 3-5: Adding Source Code to a Project

Changing the Project Settings

Before compiling the project, users must configure the project settings to specify the Java environment variable CLASSPATH, as well as define user settings on the target system.

1. Click **Project->Settings**.
2. Click on the **Classpath** tab.
3. In the **Classpath Entries** drop-down list, select **Run-Time**.
4. Replace <TARGET-SYS APOGEE INSTALLATION DIR> with the Windows network drive letter and path to the LynxOS Target. For

example `F:\`. This is the location where the Apogee Java `foundation.jar` support file is located.



Show CLASSPATH entries for this context: Run-time only

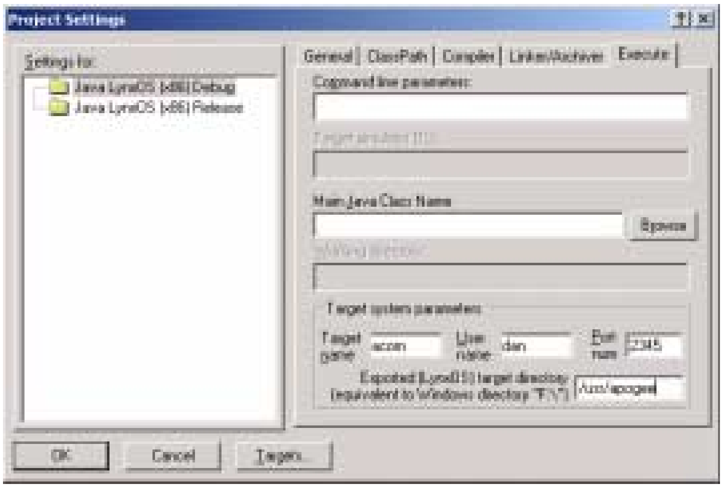
Directories for Java CLASSPATH

```
F:\<TARGET SYS APOGEE INSTALLATION DIR>\lib\founda
F:\projects\hello_world\hello_world\JavaLynxOS(x86)Debug
```

Figure 3-6: CLASSPATH Project Settings

5. Click on the **Execute** tab.
6. In the **Target System Parameters** pane, type in the target system name and user name to use when running projects. Also, type in the target system directory that is shared on the windows host (for example, `/usr/apogee`)

7. Enter the Java Class Name in the **Main Java Class Name** field. Click the **Browse** button to browse the target.



Java LynxOS (x86) Debug
Java LynxOS (x86) Release

Command line parameters:

Main Java Class Name

Target system parameters

Target name: acorn
User name: dan
Port num: 2345
Exported (LynxOS) target directory (equivalent to Windows directory "F:\"): /usr/apogee

Figure 3-7: Execute Project Settings

Compiling the Project

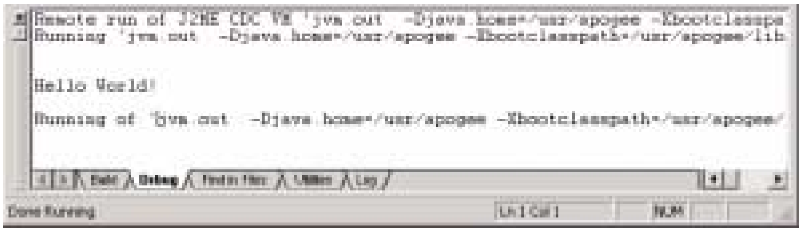
Building the project creates the Java class files, JVM and other support files. Once compiled, the application can be executed from the Aphelion interface, or run manually on the target. For more information on running the application manually, see “Running the Project from the LynxOS Command Line” on page 22.

To compile the project, use the following instructions:

1. Select either Debug (compile with debugging information) or Release (compile without debugging information) in the **Current Target** field.
2. Click the **Build** button. All of the project files are compiled, linked and saved in the project directory, in either the Debug or Release folder.

Running the Project

To run the project from the Apogee’s Aphelion interface, click on the **Execute Program** button. The project runs in the **Output Window** of the Aphelion interface.



```
Remote run of J2ME CDC VM 'jvm.out  -Djava.home=/usr/apogee -Xbootclasspa
Running 'jvm.out  -Djava.home=/usr/apogee -Xbootclasspath=/usr/apogee/lib

Hello World!

Running of 'jvm.out  -Djava.home=/usr/apogee -Xbootclasspath=/usr/apogee/
```

Figure 3-8: Running the Project

The Aphelion interface does not allow for user input when running applications. Java applications that require user interaction must be run from the LynxOS command line.

Running the Project from the LynxOS Command Line

Aphelion creates a script that allows users to quickly run their Java projects. The script `_runit.aph` runs the `jvm.out` file with all of the Java classpath, Java home, and other variable definitions set. To run the Java project, type:

```
# ./_runit.aph
```

Running jvm.out Manually for J2ME

Though the jvm.out file can be run from the _runit.aph script, in some cases, users may want to run the jvm.out manually. To do so, certain environment variables, classpaths, and other options must be set.

The syntax used when running jvm.out is as follows:

```
# <VM_NAME> -Djava.home=<JAVAHOME> \
-Xbootclasspath=<APP_SPECIFIC_CLASSPATH> \
-Djava.class.path=<PATH><MAINCLASS> <OPTIONAL ARGS>
```

The following table provides descriptions of the arguments to the Java Virtual Machine.

Table 3-1: J2ME JVM Arguments & Descriptions

Option	**Description**
<VM_NAME>	This is the name of the Java Virtual Machine that is produced by Aphelion. By default, the filename is called `jvm.out`. Check the project settings to verify the JVM filename.
`-Djava.home=`*<JAVAHOME>*	This is the Java classpath for the application to run.
`-Xbootclasspath=` *<APP_SPECIFIC_CLASSPATH>*	This is the pathname that contains the `foundation.jar` file on the target. For example, `/usr/apogee/lib`.
`-Djava.class.path=` *<PATH><MAINCLASS>*	This is the full pathname and name of the Java class containing the `main()` method. For example, the `main()` method is contained in the class `HelloWorld`:.
<OPTIONAL_ARGS>	Any additional arguments.

For example, to run the HelloWorld class with the build JVM, type in the following command:

```
bash-2.02# jvm.out -Djava.home=/usr/apogee \
-Xbootclasspath=/usr/apogee/lib/foundation.jar \
-Djava.class.path=. HelloWorld
```

CHAPTER 4

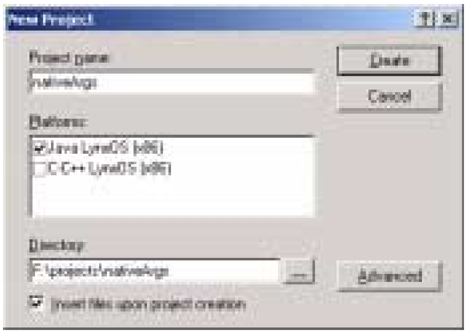
Debugging a Project

This chapter details creating and debugging a Java project with both Java and native code.

Creating the Project

Create an empty Java project.

1. Select **New -> Project** from the Aphelion Interface.
2. The New Project Window appears. Type in a project name (`nativeArgs` in this example), select the Java LynxOS Platform, and specify the project directory (if necessary).



New Project

Project name: nativeArgs

Platforms:
[✓] Java LynxOS (x86)
[ ] C/C++ LynxOS (x86)

Directory: F:\projects\nativeArgs

[✓] Insert files upon project creation

Figure 4-1: Creating the nativeArgs Project

3. The **Add Files to Project** Window appears. Several Java and Native Method source files are included with Aphelion. These example source files are located in the `samples/src` directory of the Aphelion

Installation directory on the Windows host. In the `nativeArgs` example directory, add the files `CmdLine.c`, `native_code.c` to the project.

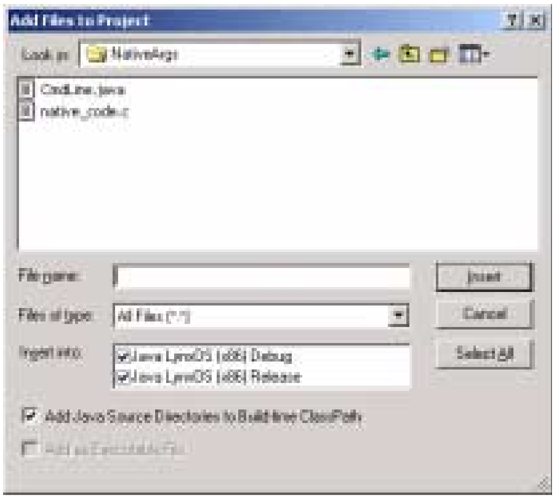
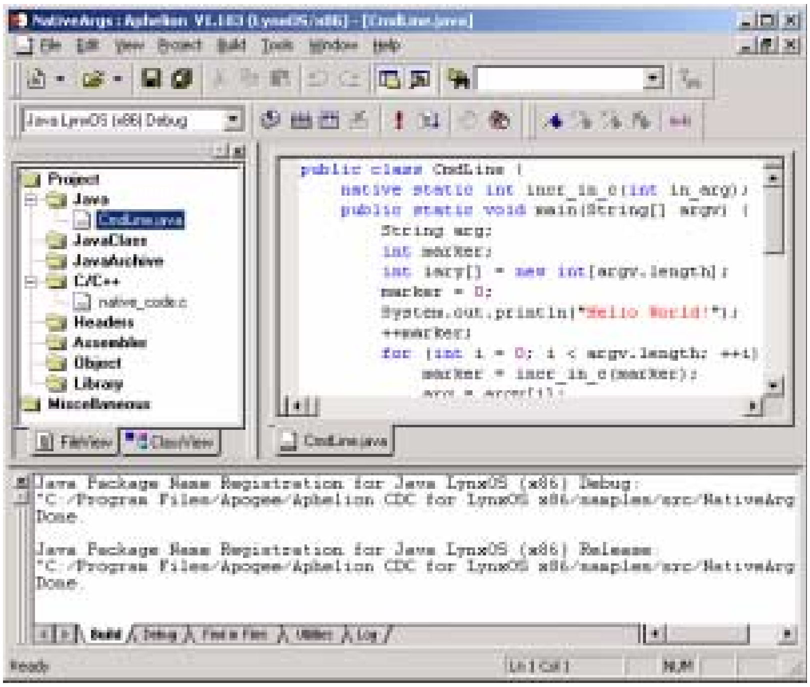


Figure 4-2: Adding nativeArgs source files to Project

The source files are added to the project. The source files are organized in the **Project Workspace Window** according to their file type.



```
public class CmdLine {
    native static int incr_in_c(int in_arg);
    public static void main(String[] argv) {
        String arg;
        int marker;
        int iary[] = new int[argv.length];
        marker = 0;
        System.out.println("Hello World!");
        ++marker;
        for (int i = 0; i < argv.length; ++i)
            marker = incr_in_c(marker);
            arg = argv[i];
```

```
Java Package Name Registration for Java LynxOS (x86) Debug:
"C:/Program Files/Apogee/Aphelion CDC for LynxOS x86/samples/src/NativeArg
Done.

Java Package Name Registration for Java LynxOS (x86) Release:
"C:/Program Files/Apogee/Aphelion CDC for LynxOS x86/samples/src/NativeArg
Done.
```

Figure 4-3: nativeArgs Project Window

Generating Header Files for Native Methods

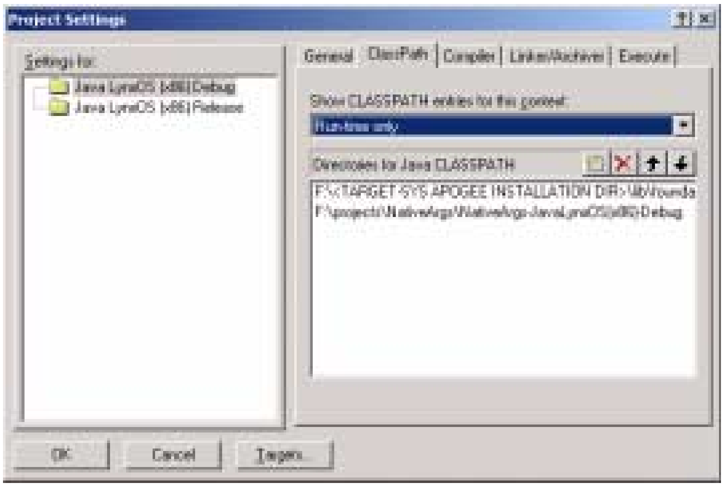
When using header files for Native Methods in a Java project, the .h file is generated within Aphelion with the Apogee `apjavah` tool. This is similar to the Sun `javah` tool that is specific to Apogee's compiler.

Changing the Project Settings

Before compiling the project, users must configure the project settings to specify the Java environment variable CLASSPATH, as well as define user settings on the target system.

1. Click **Project->Settings**.

2. Click on the **Classpath** tab
3. In the **Classpath Entries** drop-down list, select **Run-Time**.
4. Replace <TARGET-SYS APOGEE INSTALLATION DIR> with the Windows network drive letter and path to the LynxOS Target. For example `F:\lib`. This is the location where the Apogee Java `foundation.jar` support file is located.



Show CLASSPATH entries for this context: Run-time only

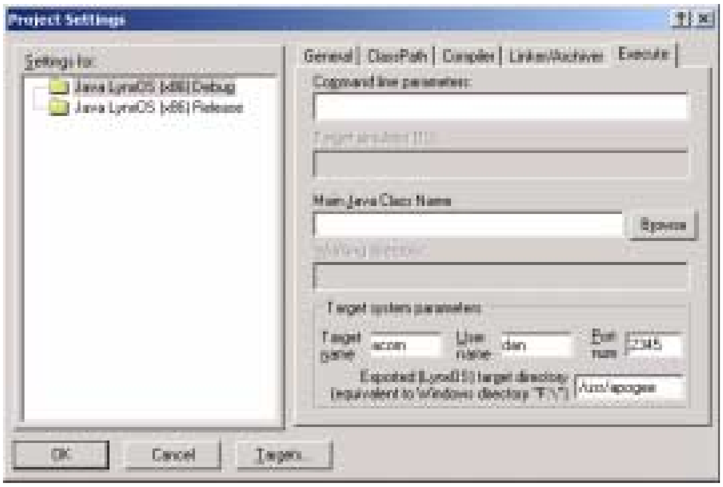
Directories for Java CLASSPATH

F:\<TARGET-SYS APOGEE INSTALLATION DIR>\lib\founda
F:\projects\NativeArgs\NativeArgs-JavaLynxOS(x86)-Debug

Figure 4-4: CLASSPATH Project Settings

5. Click on the **Execute** tab.
6. In the **Target System Parameters** pane, type in the target system name and user name to use when running projects. Also, type in the target system directory that is shared on the windows host (for example, `/usr/apogee`).

7. Enter the Java Class Name in the **Main Java Class Name** field. Click the **Browse** button to browse the target.



Command line parameters:

Main Java Class Name

Target system parameters

Target name: acorn

User name: dan

Port num: 2345

Exported (LynxOS) target directory (equivalent to Windows directory "F:\"): /usr/apogee

Figure 4-5: Execute Project Settings

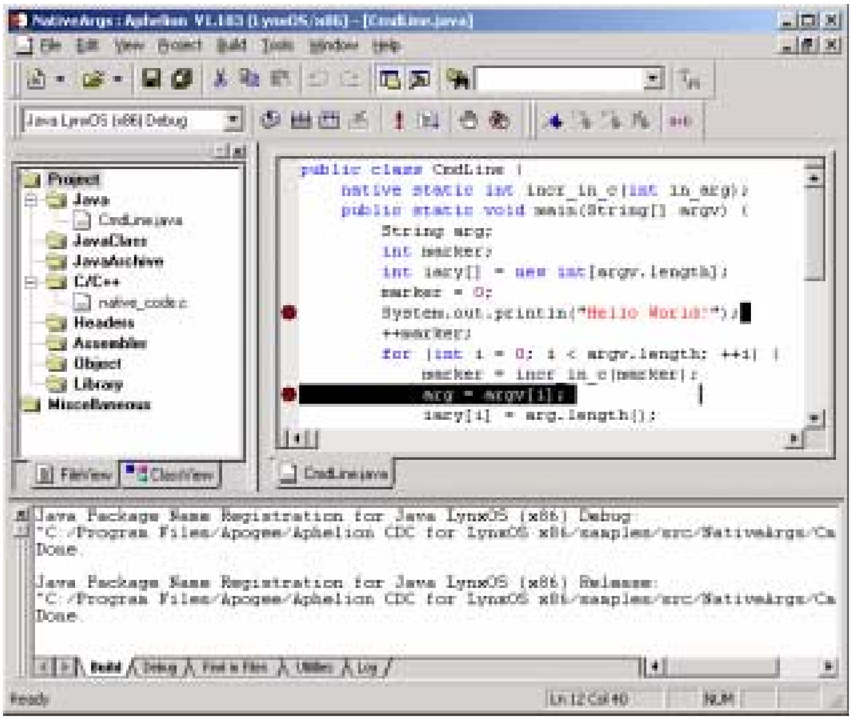
Building the Source

Build the source by clicking on the **Build** button. This compiles all of the sources, creating the `jvm.out` file, and any Java support files. Be sure to select **Java LynxOS x86 Debug** in the **Current Target** pulldown menu.

Debugging the Project

The following instructions provide details on setting breakpoints, starting the debugger, and stepping through source code.

1. Set breakpoints in the source code by selecting a line and clicking the **Insert/Remove Breakpoint** button.



```
public class CmdLine {
    native static int incr_in_c(int in_arg);
    public static void main(String[] argv) {
        String arg;
        int marker;
        int iary[] = new int[argv.length];
        marker = 0;
        System.out.println("Hello World!");
        ++marker;
        for (int i = 0; i < argv.length; ++i) {
            marker = incr_in_c(marker);
            arg = argv[i];
            iary[i] = arg.length();
```

```
Java Package Name Registration for Java LynxOS (x86) Debug
"C:/Program Files/Apogee/Aphelion CDC for LynxOS x86/samples/src/NativeArgs/Cm
Done

Java Package Name Registration for Java LynxOS (x86) Release:
"C:/Program Files/Apogee/Aphelion CDC for LynxOS x86/samples/src/NativeArgs/Cm
Done
```

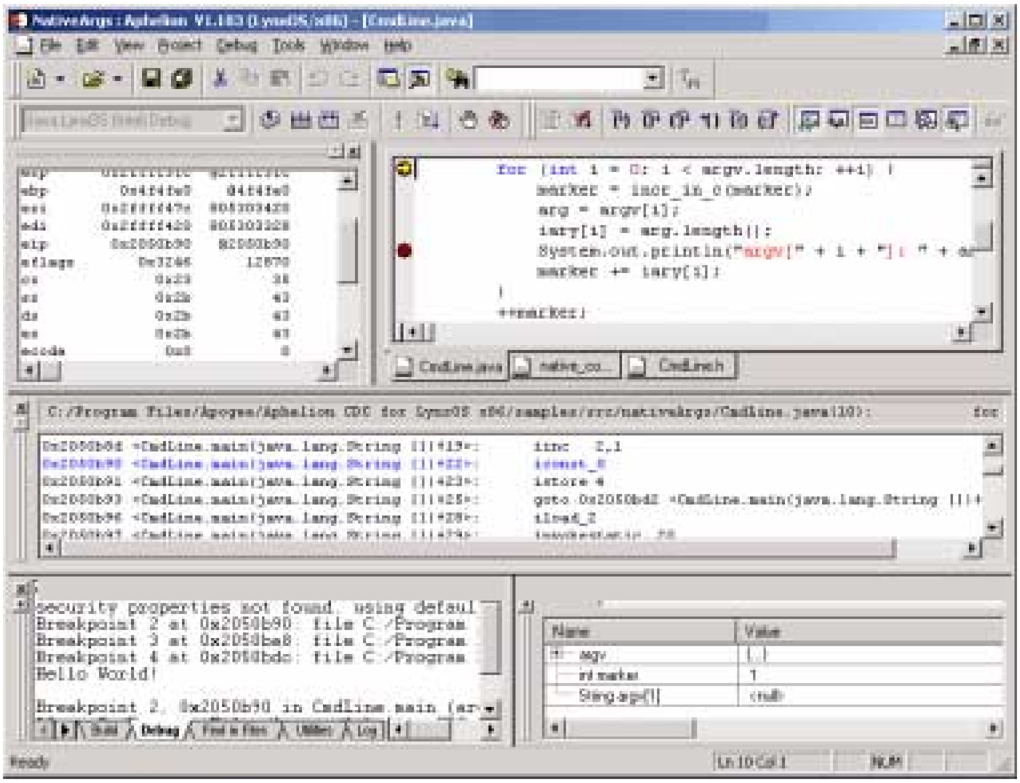
Figure 4-6: Setting Breakpoints

2. Once all breakpoints are set, start the debugger by clicking on the **Go** button.

NOTE: Be sure that the default data size is updated in `/etc/startab` on the LynxOS target. See the section "Increasing the Default Data Size in /etc/starttab" on page 9.

---

3. The program runs until it reaches a breakpoint. Then the debugging toolbar appears. From this toolbar, users can select a variety of windows to view, including Call Stack, Disassembly, and Watchpoints.


NativeArgs : Aphelion V1.103 (LynxOS/x86) - [CmdLine.java]
File Edit View Project Debug Tools Window Help
CmdLine.java
native_co...
CmdLine.h
Build
Debug
Find in Files
Utilities
Log
Ready
Ln 10 Col 1
NUM

```
ebp        0x4f4f4fe0    4f4f4fe0
esi        0x2ffff47c    805303420
edi        0x2ffff420    805303328
eip        0x2050b90     82050b90
eflags     0x3246        12870
cs         0x23          35
ss         0x2b          43
ds         0x2b          43
es         0x2b          43
ecode      0x0           0
```

```
        for (int i = 0; i < argv.length; ++i) {
            marker = inor_in_0(marker);
            arg = argv[i];
            iary[i] = arg.length();
            System.out.println("argv[" + i + "]: " + ar
            marker += iary[i];
        }
        ++marker;
```

```
C:/Program Files/Apogee/Aphelion CDC for LynxOS x86/samples/src/nativeargs/CmdLine.java(10):   for
0x2050b8d <CmdLine.main(java.lang.String [])+19>:      iinc   2,1
0x2050b90 <CmdLine.main(java.lang.String [])+22>:      iconst_0
0x2050b91 <CmdLine.main(java.lang.String [])+23>:      istore 4
0x2050b93 <CmdLine.main(java.lang.String [])+25>:      goto 0x2050bd2 <CmdLine.main(java.lang.String [])+
0x2050b96 <CmdLine.main(java.lang.String [])+28>:      iload_2
```

```
security properties not found. using defaul
Breakpoint 2 at 0x2050b90: file C:/Program
Breakpoint 3 at 0x2050ba8: file C:/Program
Breakpoint 4 at 0x2050bdc: file C:/Program
Hello World!

Breakpoint 2, 0x2050b90 in CmdLine.main (ar
```

Name	Value
argv	{...}
int marker	1
String arg[1]	<null>

Figure 4-7: Debugging Windows

Users can step through functions, edit values, and observe source from the Aphelion Debugger.

The Aphelion Debugger moves between Java bytecode source to native methods transparently. Windows are automatically updated with the appropriate values, depending on the type of source currently being debugged.

After stepping through the applications, the debugger quits, and users are returned to the project window.

CHAPTER 5

Known Problems and Limitations

This section covers known problems and limitations of Aphelion.

Debugging Limitations

Updating Display Windows

Changing a value in a debugging window does not automatically update the value in other debugging windows. To refresh the values in a window, close and reopen the window. For example, if a variable is updated in the **Local** window, close and reopen the **Watchpoint** window to update the variable in the **Watchpoint** window.

Stepping through a program refreshes all windows.

Index