



HP JToolkit 2.0 for NonStop servers

Features at a glance

- Utilizes knowledge of familiar software products
- Enables Pathway server programs to be written in Java
- Combines the flexibility of Java with the scalability of NonStop servers

HP's JToolkit for NonStop servers provides you with application program interfaces (APIs) and tools that combine the advantages of Java technology with your knowledge of existing HP NonStop server products, such as Pathway and Enscribe software. Using these Java APIs allows programmers to take advantage of the productivity features of the Java language to design new applications that provide deep integration with existing software assets and leverage existing knowledge and skill sets. The JToolkit for NonStop servers includes the following software:

- *Java API for Pathway* allows you to write fully functional Pathway server applications using the Java language. The APIs provide support for writing both single-thread and multi-thread Pathway servers.
- *Java API for Enscribe* provides you with a simplified API for accessing Enscribe files from Java applications. It supports four types of Enscribe files: Unstructured, EntrySequence, KeySequence, and Relative.
- *Java API for Pathsend* enables Java applications running on NonStop servers to send request messages to, and receive replies from, existing Pathway servers.
- *ddl2java tool* is a command-line tool that generates Java classes from data descriptions defined in a Data Definition Language (DDL) dictionary file. The generated Java classes implement methods that can convert Java data types to HP NonStop Kernel operating system data types and vice versa. These classes can then be used with Java APIs for Pathway, Enscribe, and Pathsend to further simplify the programming logic required to implement your business logic.

Scalable TCP/IP (SIP) software provides a transparent way to give the NonStop server advantages of scalability and availability to a network server (SIP server) written in Java. Because SIP does not introduce any APIs, existing network servers written in Java and their clients can take advantage of SIP with no code changes.

These APIs are fully integrated with the Java Virtual Machine's (JVM's) threading engine. This allows you to write efficient multi-thread Java applications by leveraging Java's native threading facilities. Specifically, all I/O operations such as reading \$RECEIVE, reading or writing Enscribe records, and performing Pathsend's are synchronous with respect to a given Java thread. This means that while a thread is blocked waiting for the I/O operation to complete, all other threads within the application can be running.

Java API for Pathway

Until now, you could write Pathway server programs in any of the following languages: COBOL, C, C++, and pTAL. With Java API for Pathway, you can now write your Pathway server programs in Java, enabling you to combine the simplicity and the productivity features of the Java language with the unparalleled availability and scalability of the Pathway application server platform.

In simple terms, a Pathway server program services requests from external processes by reading requests off its \$RECEIVE, performing the necessary business logic to process the request and then replying to the request. In essence, Java API for Pathway provides Java APIs for reading from, and replying to, \$RECEIVE messages while making the APIs easy for Pathway programmers to understand.

The Java API for Pathway product enables you to perform the following functions:

- Open and close \$RECEIVE
- Perform a read operation on \$RECEIVE
- Reply to a message read from \$RECEIVE
- Perform multiple reads on \$RECEIVE before replying
- Cancel an outstanding read on \$RECEIVE
- Reply to messages in any order
- Set priority or first-in, first-out (FIFO) ordering of messages in the \$RECEIVE queue
- Enable receipt of user-selected system messages
- Enable optional notification, via a Java exception, when there are no more openers

Java API for Pathway comprises four major classes:

- *Receive*: This singleton class represents \$RECEIVE. The class provides methods to open and close \$RECEIVE, read and reply to \$RECEIVE messages, and cancel an outstanding read. In addition, various set methods are provided to set \$RECEIVE attributes such as "receive depth," "sync depth," and so on.
- *ReceiveInfo*: This class provides information about the last read request message and provides various get methods that return information about the attributes associated with the message.
- *ReceiveNoOpeners*: This is an exception class, an instance of which is thrown if a read is performed on \$RECEIVE when there are no more openers. A mechanism exists that enables you to disable this exception.
- *ProcessHandle*: This class provides methods to decompose a NonStop Kernel operating system process handle into its constituent parts. The methods provided in this class can be used to find detailed information on the process that sent the request.

You can use the methods associated with these classes to write both single-thread and multi-thread Pathway server programs.

Basic structure of a single-thread server program

The simplest form of a Pathway server is a single-thread server that services one request at a time. Such servers can generally be written in the following sequence:

1. Obtain an instance of the Receive class using `Receive.getInstance()`
2. Set any \$RECEIVE attributes for which the supplied default does not fit your needs. Some attributes, such as receive depth and sync depth, must be set before opening \$RECEIVE, while other attributes can be set after opening \$RECEIVE.
3. Open \$RECEIVE using `Receive.open()`
4. Read a message from \$RECEIVE using `Receive.read()`
5. Perform whatever business logic is associated with the message that was just read
6. Reply to the message using `Receive.reply()`
7. Continue reading and replying to messages until no more openers of the server exist, which will be signaled when a `ReceiveNoOpeners` exception is thrown

Considerations for multi-thread server programs

Multi-thread servers have the ability to service multiple request messages concurrently. Because these APIs are “thread aware,” you can leverage Java’s native threading facilities to significantly simplify your multi-thread server programs.

A multi-thread server can use two different models to read and process requests. The first model has one thread performing all the read operations on \$RECEIVE and handing off the requests to other worker threads for processing. The second model has multiple threads performing read operations. Each thread is responsible for processing the request that it read. In the second model, you must ensure that the read on \$RECEIVE is through a synchronized method so that only one thread at a time calls the read method in the Receive class. This is necessary because only one read operation can be outstanding on \$RECEIVE at any given time.

In summary, Java API for Pathway allows you to leverage the power of the Java language to write fully functional Pathway server programs.

Java API for Enscribe

Like Java API for Pathway, Java API for Enscribe provides Java wrappers around the Enscribe procedure calls that give Java programs access to Enscribe disk files while being easy for Enscribe programmers to use and understand.

Java API for Enscribe provides access to the following types of Enscribe files: key sequenced, relative, entry sequenced, and unstructured. Also, the APIs support both Format 1 and Format 2 files. (Format 2 files support a larger file and partition size.)

The Java API for Enscribe enables you to perform the following functions:

- Create a new file
- Open an existing file
- Alter the characteristics of an existing file
- Rename an existing file
- Purge an existing file

- Sequentially read a file
- Position to a particular record within a file
- Lock a file or a record within a file
- Unlock a file or a record within a file
- Write records to a file
- Obtain information about an existing file
- Close a file

The following is an overview of the eight major classes that are comprised by Java API for Enscribe:

- *EnscribeFile*: This class provides methods that allow you to perform standard operations on Enscribe files.
- *EnscribeOpenOptions*: This class provides methods to set the various options needed to open a file. An instance of this class is set with the required options and passed as a parameter to the *EnscribeFile.open()* method. The open method of *EnscribeFile* class is also overloaded to open a file by specifying only the access and exclusion mode.
- *EnscribeFileAttributes*: This class provides get methods to obtain information about the attributes of an existing file and set methods that can be used to alter the attributes of an existing file.
- *EnscribeCreationAttributes*: This class is an abstract class with subclasses that provide methods to set and get file-creation attributes. Subclasses such as *StructuredCreationAttributes* (for relative and entry-sequenced files), *KeySequencedCreationAttributes* (for key-sequenced files), and *UnstructuredCreationAttributes* (for unstructured files) exist to set and get file-creation attributes for a particular file type.
- *EnscribeLastCallStatus*: This class provides methods to retrieve transient attributes of a file after performing an operation on the file.
- *AlternateKeyDescriptor*: This class provides get methods to obtain the attributes of a specific alternate key and also provides set methods to change those attributes.
- *EnscribeKeyPositionOptions*: This class provides set and get methods to position within a file according to the value of the primary or alternate key.
- *PartitionDescriptor*: This class provides get methods to obtain the attributes of a specific partition and also provides set methods to change those attributes.

For operations on audited files, you must use either the Java Transaction API (JTA) or other methods provided by `com.tandem.tmf` classes to perform transaction management.

In summary, Java API for Enscribe provides a simple Java API that hides the complexity behind the "raw" Enscribe procedure calls. In addition, because these APIs are thread aware, you can write multi-thread programs by leveraging Java's native threading facilities. All I/O operations are synchronous with respect to a given Java thread. However, while a thread is blocked waiting for the I/O operation to complete, all other threads within the application can be running.

Java API for Pathsend

Java API for Pathsend allows a Java application to send requests to, and receive replies from, Pathway servers without requiring any modifications to the server. For example, the Java application could be a Java servlet that, in essence, provides a Web interface to the Pathway server.

The Java API for Pathsend provides the following functionality:

- Marshals a Java object representing a request to a Pathway server class so that the instance variables of the object are converted to NonStop Kernel system data types, which are then placed in the Pathsend buffer as a request message
- Unmarshals a response Pathsend buffer so that the NonStop Kernel system data types can be converted to the instance variables of a Java object representing the response from a Pathway server class
- Performs a Pathsend request to a Pathway server class

The `TsmpServer` class is the major class of Java API for Pathsend. This class is used to communicate with the Pathway server. In the constructor for this class, you provide the name of the target server class as well as the Pathmon under which the target server class runs. The class also provides several overloaded service methods that are used to perform the actual Pathsend.

Usage scenario

To use the Java API for Pathsend, you will typically use the following sequence:

1. Use the `ddl2java` tool (described later) to generate Java classes representing the request and reply data structures defined in a DDL dictionary file.
2. Use the various set methods of the generated request class to set the request data.
3. Instantiate an object of type `TsmpServer`.
4. Use the service method to send the request and receive a response. This method will automatically marshal the request object before sending the request and unmarshal the response to the response object.
5. Use the various get methods of the response object to retrieve the reply data.

In summary, Java API for Pathsend provides an easy mechanism for Java applications to invoke Pathway services. When used with the `ddl2java` tool, you do not have to worry about data conversions between Java data types and NonStop Kernel system data types.

`ddl2java` tool

The `ddl2java` tool is a command-line tool that can generate Java classes from data descriptions, such as data definitions or data records defined in a DDL dictionary file. These classes can then be used to

- Read and reply to messages read from `$RECEIVE` using Java API for Pathway
- Read or write data to an Enscribe file using Java API for Enscribe
- Send or receive data from a Pathway server using Java API for Pathsend

Using the `ddl2java` tool simplifies the programming required to use Java API for Enscribe and Java API for Pathsend. For example, if you were to deal with a byte array

for performing Pathsends, you would have to write code to marshal your request object (that is, convert Java data types to NonStop Kernel data types expected by the Pathway server) and unmarshal the reply to a response object (that is, convert the server reply from NonStop Kernel data types to Java data types). Using the ddl2java tool simplifies your code because it will generate classes that can do the marshaling and unmarshaling for you.

Depending on the options passed to the ddl2java tool, the tool generates classes that implement one or both of the following interfaces: GuardianOutput and GuardianInput. Objects of type GuardianOutput can be used to reply to messages read from \$RECEIVE, write records to Enscribe files, and send requests to Pathway servers. Objects of type GuardianInput can be used to read messages from \$RECEIVE, read records from Enscribe files, and receive replies from Pathway servers. Classes that implement the GuardianOutput interface are known as output classes, classes that implement the GuardianInput interface are known as input classes, and classes that implement both interfaces are known as I/O classes.

Instances of output classes are used as "out" parameters to certain methods. For example, the object representing the request in the service method of the TsmServer class can be an output object. Similarly, instances of input classes are used as "in" parameters to certain methods. For example, the various read methods of the EnscribeFile class accept an input object as a parameter. Instances of I/O classes can be used wherever an output object or an input object is acceptable.

For each class that the ddl2java tool generates from a DDL data description, the ddl2java tool generates the following:

- *Java class names:* The ddl2java tool forms the generated class name by shifting up the first character of the DDL description name and adding the prefix "O" for output classes, the prefix "I" for input classes, and the prefix "IO" for I/O classes.
- *Instance variable names:* The ddl2java tool maps DDL field names to Java instance variable names with no changes, except in cases where the name conflicts with Java syntax rules (for example, replacing a hyphen with an underscore).
- *Instance variable data types:* The data types of instance variables are similar to the DDL data types. For example, the DDL binary 16 data type is mapped to the Java data type short, while the float 64 data type is mapped to Java data type double.
- *Set and clear methods:* The ddl2java tool generates set and clear methods for output and I/O classes. The ddl2java tool creates a set method name from a Java instance variable name by shifting up the first letter of the Java instance variable name and adding the prefix "set." The ddl2java tool also generates clear methods that can be used to clear values previously set. Clear methods set instance variables that are declared as "short," "int," "long," "float," or "double" to zero and set instance variables that are declared as "string" to null.
- *Get methods:* The ddl2java tool generates get methods for input and I/O classes. The ddl2java tool creates a get method name from a Java instance variable name by shifting up the first letter of the Java instance variable name and adding the prefix "get."
- *Calls to conversion methods:* The ddl2java tool generates calls to conversion methods that convert and move the data to and from NonStop Kernel system data types.
 - *Marshal:* For output and I/O classes, the ddl2java tool generates a marshal method that converts the values of the instance variables to NonStop Kernel system data types and places these values in a byte array. This method is typically called before writing data out. Methods in the Java APIs for Pathway,

Enscribe, and Pathsend that accept an output (or I/O) object automatically call this method for you.

- *Unmarshal*: For input and I/O classes, the ddl2java tool generates an unmarshal method that takes NonStop Kernel system data in a byte array, converts it into Java data types, and moves it to individual fields in the input or I/O object. This method is typically called after reading data in. Methods in the Java APIs for Pathway, Enscribe, and Pathsend that accept an input (or I/O) object automatically call this method for you.

In summary, the ddl2java tool further simplifies the programming effort required to use the APIs for Pathway, Enscribe, and Pathsend software by generating classes that can marshal and unmarshal the data as well as providing set and get methods to set and retrieve instance variables.

Scalable TCP/IP (SIP) software

Scalable TCP/IP (SIP) software provides a transparent way to provide the system advantages of NonStop servers, such as scalability and availability, to a network server (SIP server) written in the Java language. Existing servers written in Java and their clients can take advantage of SIP with no code changes.

SIP provides a method for distributing client requests among SIP servers. The distributor and SIP servers run as Pathway server classes. Running under the Pathway environment gives SIP the manageability, load balancing, and scalability benefits of NonStop servers.

For example, Pathway system processes take responsibility for

- Starting and managing the SIP server classes
- Distributing the SIP server processes across processors in NonStop systems
- Load balancing the client requests

SIP servers

A SIP server is any Java program that uses the accept method in the `java.net.ServerSocket` class. A SIP server usually has code that does the following:

- Constructs a `ServerSocket` to listen for connections on a specified port
- Invokes the accept method on the `ServerSocket` to accept the connection request

To configure a Java network server as a SIP server, you must configure the server as a Pathway server class. A Java network server can be configured as a SIP server if

- The SIP server uses the accept method in the `java.net.ServerSocket` class.
- The intended use of the SIP server is to communicate with a client until the client's request is fully processed. After the communication has terminated, the SIP server closes the socket that was used to communicate with the server. (Note: You should not configure a Java application as a SIP server if the application retains the client state after the client disconnects from the server, because the SIP distributor cannot guarantee that the client will reconnect to the same SIP server process.)

A program can have several types of SIP servers, with each type performing a different service. In this case, each type of SIP server runs in a different server class. SIP servers that perform different tasks can be configured in the same Pathway application.

SIP distributor

The SIP distributor runs as a single server in a Pathway server class. If a program has more than one SIP server (and therefore has more than one server class), a SIP distributor is created for each server class. Because a SIP distributor runs under the Pathway environment, Pathway software restarts the distributor if it fails (for example, due to a processor failure).

You can use the configuration tool to configure SIP servers in a Pathway application. The tool automatically configures the SIP distributor(s).

During its initialization routine, a SIP distributor binds itself to one or more specified ports (this is called a *TCP/IP socket bind*). The SIP distributor acts as a proxy for the SIP server, listening in on any ports that the SIP server would use if it had no proxy.

SIP tools

The following SIP tools are provided:

- *Configuration tool*, which configures a new SIP application
- *Reconfiguration tool*, which reconfigures an existing SIP application
- *Management tool*, which manages and views the attributes of an existing SIP application

In summary, SIP is an excellent choice for providing higher levels of availability and scalability to a basic Java network server without requiring any additional code.

Ordering information

<i>Product ID</i>	<i>Description</i>
SJ76V2	HP JToolkit for NonStop servers

Specifications

System requirements

Hardware	Any NonStop S-series server running the NonStop Kernel operating system, Release Version Update G06.20 or later
Software	HP NonStop Server for Java 4.0, based on Java 2 SDK, Standard Edition, Version 1.4.1 (SJ96V4)

Notes

JToolkit 1.0 and 2.0 for NonStop servers are functionally identical. There are no API/functionality changes in Version 2.0, except that SIP is not included in the first release of Version 2.0. Customers that use SIP will not be able to move to NonStop Server for Java 4.0 until a subsequent release of JToolkit for NonStop servers includes the SIP APIs.

JToolkit 1.0 for NonStop servers is compatible with HP NonStop Server for Java 2.1 and 3.1 software.

JToolkit 2.0 for NonStop servers is compatible with NonStop Server for Java 4.0 software.

© Copyright 2003 Hewlett-Packard Development Company, L.P.
Java is a US trademark of Sun Microsystems, Inc. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

10/2003

For more information, go to www.hp.com/go/nonstop.

