Distributed Protocol Factory 2.0a Requirements Specification

1. Scope

1.1 Overview

The Distributed Protocol Factory provides a generic API for accessing messaging, as well as a default implementation for nodes on the same subnet. This enhancement to the component will provide a new implementation that allows for data to be compressed before transferring it to other nodes. This will allow nodes in a band width constrained environment to more efficiently communicate.

1.2 Logic Requirements

1.2.1 Pluggable Compression

The compression mechanism must be pluggable to allow for different compression implementation to be selected as they best fit the environment.

1.2.2 Default Compression Implementation

There must be a default compression provided with the component.

1.2.3 Self Identifying Compression

The compressed messages must retain some information so the receiving node can identify what compression is used and select the correct decompression method. If the compression method is unknown to the receiver it will take actions consistent with the error handling already in the Distributed Protocol Factory.

NB: The compression is considered part of the protocol, not an application layer on top of the protocol, so the protocol must deal with the unknown compression method error.

1.2.4 Default Implementation

Where appropriate and consistent with the enhancement requirements and other TopCoder guidelines, the default implementation may be reused in part or whole in any manner.

1.3 Required Algorithms

- How is the compression done?
- How is the compression algorithm identified by the receiver?

1.4 Example of the Software Usage

A client is running a large group of nodes that will be synching large caches using the Distributed Protocol Factory. Due to bandwidth constraints between the primary and secondary server groups, it is not feasible to transmit the data without compression. The client will use the compressing version of the Distributed Protocol Factory to enable them to synchronize their servers' caches in this environment.

1.5 Future Component Direction

None specified.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The design of this component will identify any necessary underlying protocols (such as tcp/ip or udp.) LGPL is discouraged, JINI and JXTA are allowed.

2.1.3 Environment Requirements

- Development language: Java 1.4
- Compile target: Java 1.4
- Multiple runtime environments
    - WebLogic
    - JBoss
    - JVM 1.4
- It is not guaranteed that the component will be running inside a J2EE container, but the J2EE jar will be accessible.

2.1.4 Package Structure

com.topcoder.network.synchronization.compression

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

None required.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

As determined by design.

3.2.2 TopCoder Software Component Dependencies:

Compression Utility 2.0 may be used.
(Do not use configuration manager, directly or indirectly.)

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.