

---

# RESONATE | WHITE PAPER

## *Resonate Central Dispatch™ Data Stream Processing*

---

November 2001

Resonate Central Dispatch's data stream processing, a powerful, data peering capability that serves as the basis for directing traffic and maintaining high service levels within the e-business infrastructure.



# Table of Contents

<b>Central Dispatch Data Stream Processing.....</b>	<b>3</b>
Benefits and uses of data stream processing .....	3
<b>Lay the foundation for deep application insight .....</b>	<b>4</b>
Scenario: HTTP header parsing .....	4
Scenario: Selective IP address distribution .....	5
<b>Optimize transaction processing.....</b>	<b>5</b>
Scenario: Persistence based on J2EE session ID .....	6
<b>Simplify application programming .....</b>	<b>6</b>
Scenario: Persistence based on XML transactions.....	7
Scenario: Optimizing transactions based on SOAP HTTP requests .....	8
<b>Remove barriers, create opportunities.....</b>	<b>8</b>
Scenario: Scheduling based on application type .....	8
<b>Realize higher service levels.....</b>	<b>9</b>
Scenario: Redirecting the client request due to a reply error condition .....	10
Scenario: Redirecting the client request based on the SOAP HTTP response .....	11
Scenario: Change the server weight of the outbound server .....	13
Scenario: Disable the outbound server.....	14
<b>Summary .....</b>	<b>14</b>
Appendix A: HTTP header parsing scenario .....	15
Appendix B: XML scenario.....	16

## Central Dispatch Data Stream Processing

As e-businesses become more complex, enterprise IT groups look for ways to make it easier to operate their infrastructure, while trying to provide the high level of service their clients demand. They need ways to identify critical pieces of information that can increase the performance, reliability, and scalability of their e-services. This requires a deeper knowledge of how the e-business components – clients, services, applications, and message transports, work together.

Resonate Central Dispatch™ introduces a unique, data peering capability that serves as the basis for directing traffic and maintaining high service levels within the e-business infrastructure. This ability to look into all data from Layer 3 to 7 is called *data stream processing* (DSP). DSP enables users to deliver traffic to content servers, make changes to the Central Dispatch cluster to optimize traffic flow, and to respond to failures gracefully. Resonate Central Dispatch now provides a protocol-independent ability to provide crucial e-business functionality to streaming media, wireless telephony, voice over IP, XML, and a host of other critical technologies.

### Benefits and uses of data stream processing

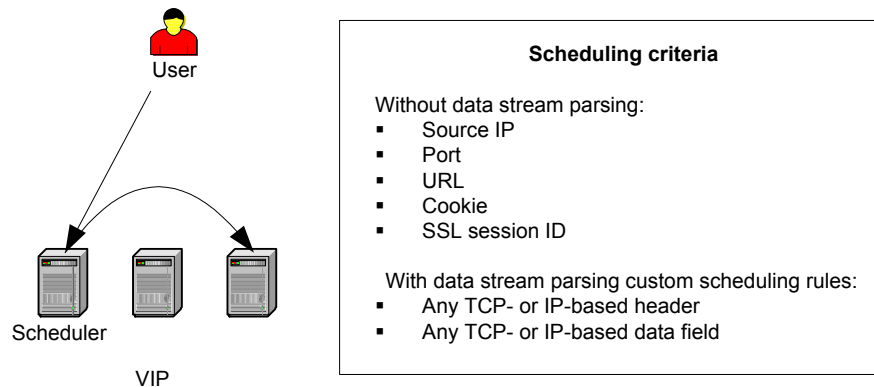
How many ways can data stream processing be used? There are no limits to the ideas and implementation methods of DSP. The following table shows how DSP adds value to managing e-services.

Benefit	Sample Use
Deeper application insight (lay the foundation)	<ul style="list-style-type: none"> <li>▪ Schedule on any TCP, UDP, or other TCP- or IP-based protocol data fields</li> <li>▪ Link scheduling decisions on all elements of the HTTP header</li> <li>▪ Exclude users' IP addresses during a Denial of Service attack</li> </ul>
Optimized transaction processing	<ul style="list-style-type: none"> <li>▪ Establish session persistence for J2EE-based applications</li> <li>▪ Prioritize scheduling decisions</li> <li>▪ Map persistence on cookies, and other attribute-value pairs</li> </ul>
Simplified application programming	<ul style="list-style-type: none"> <li>▪ Read XML, Voice over IP, wireless protocols</li> <li>▪ Link legacy data to e-business needs</li> <li>▪ Select the optimal processing resource based on content</li> </ul>
Create opportunities (remove barriers)	<ul style="list-style-type: none"> <li>▪ Determine the client application type</li> <li>▪ Schedule requests based on multi-byte character sets</li> <li>▪ Check binary content</li> </ul>
Higher service levels	<ul style="list-style-type: none"> <li>▪ Redirect requests to another server based on HTTP or application error codes</li> <li>▪ Lower or increase server weight based upon the outbound data response</li> <li>▪ Disable the server on catastrophic errors</li> </ul>

## Lay the foundation for deep application insight

Resonate Central Dispatch is the first product that provides deep access to any part of the data stream, whether inbound or outbound. Users can link scheduling to content, and not have to rely on cookies, or other web-browsing techniques. This is important in cases where users may disable their browser's feature to save cookies, which may affect traffic decision where cookies are expected to be used. In some cases, cookies are too limiting for the application to use, or the HTTP header does not have the right kind of information required for making application policy decisions.

**Figure 1. Resonate Central Dispatch scheduling criteria.**



Resonate Central Dispatch handles scheduling decisions based on origination source IP, source port, URL, cookie, or SSL session ID (see Figure 1). Data stream processing custom scheduling rules, for applications requiring scheduling based on other criteria. Resonate Central Dispatch will scan any inbound or outbound TCP- and IP-based data streams, for the requested data in any packet, from the first byte to the last byte.

These custom rules are easily created using Central Dispatch's built-in graphical user interface, Resonate CDMaster. Custom scheduling rules can be created and saved for different application needs.

String patterns can be up to 64 bytes in length, if longer strings are needed, 'AND' operators are used to link the patterns. Consider the following scenario, where HTTP header parsing is required.

### Scenario: HTTP header parsing

There are occasions where it is necessary to determine the type of Web browser that a client is making their request from. One such example exists when SSL session ID persistence is used. Certain Web browsers constantly renegotiate the SSL session ID in a short time window making it difficult to maintain session persistence to a given HTTPS server. This is true with the Microsoft Internet Explorer browsers.

To avoid this problem, the request can be sent to a Web server that will force a refresh of the Web page prior to the renegotiation timeout in order to maintain the same SSL session ID. In this scenario, every HTTP GET request that is

from a Microsoft Internet Explorer browser is served to specific application servers. (Even better, it would be desirable to also verify by browser version numbers.) Data stream processing is set to scan for the following items:

- HTTP method = "GET"
- HTTP header "User-Agent:" contains "MSIE"  
(for example, the precise string might read:  
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win  
9x 4.90)")

Requests that are received from Microsoft Internet Explorer browsers are sent to specific application servers via a set of receiving Web servers.

This scenario shows how easy it is to create custom scheduling rules, and simultaneously work around problems that normally would be difficult to implement without data stream processing.

*(To refer to a code example, please see "Appendix A: HTTP Header Parsing Scenario.")*

## **Scenario: Selective IP address distribution**

Selective IP address distribution is used when the user wants to show more or less favor to the request depending on the IP address. For example, the user wants to show more favor if they are serving a client with a known set of IP addresses. In this case, the user may want to send the client's traffic to the fastest set of servers.

The user may want to show less favor if the client request is a low priority. The client may not have paid their subscription or might be the source of a Denial of Service attack. The user would then filter or send the client to a less powerful or "sorry" server. Data stream parsing can also be used to exclude IP addresses.

In either case, the user can simply schedule requests based on specifying the *source address* as the key element of the inbound data scheduling rule. This insight into the data provides the key decision making capabilities that help e-businesses operate at their best.

## ***Optimize transaction processing***

Many session decisions are based on simple criteria, such as: 'Who is the user?' and 'what kind of a transaction is this?' This kind of information is not present in the HTTP header; it is necessary to look into the content to determine this. These elements may also be part of a shopping cart transaction. In addition, the shopping cart information is stored on the same server. This is called *session persistence*, without which all of the shopping cart requests will be scattered to many servers.

To guarantee that the client is sent to the same server, it is required to link the client's IP address, cookie data, or any other data element to the appropriate

server each time a request is made. DSP allows users the ability to select any of those elements and link the inbound data stream to a specific server, essentially making the session 'sticky', or persistent. In DSP, session persistence can be based on a defined attribute-value pattern, for example, 'Session=123, 'USERID=ABC12', etc. Session persistence can also be defined by TCP header information such as source address, source port, or type of service.

## Scenario: Persistence based on J2EE session ID

This scenario shows how users can derive persistence from mechanisms that go beyond what Web switches and other traffic management devices offer. J2EE session IDs are similar to HTTP cookies, but they differ in two ways. First, these session IDs use URL characters that are not part of the HTTP standard, for example, the semicolon (;) character. As a result, normal out-of-the-box URL parsing rules will not work. Second, J2EE is used extensively at the application server level. The expected input request may look like the following example:

```
http://test42/catalog/index.html;jsessionid=1234;?
```

The inbound request will be scanned for ';jsessionid='. Once found, the next twelve bytes are expected to be the 'jsessionid' value. This will be used for persisting to the cluster of designated servers.

The resultant expressions would appear as follows.

```
Expression Type: ASCII
Start Offset: 0
End Offset: 100
Pattern Length: 112
Delimiter Count: 2
Maximum Value Length: 12
Pattern: ;jsessionid=
Delimiters: ;?
Name: J2EESession
```

Data stream processing provides the ability to extend scheduling capabilities to non-standard characters and values that are limited to standard ASCII characters, and multi-byte characters.

## *Simplify application programming*

Resonate Central Dispatch's data stream processing is like having an API into your application; it provides full access to the underlying programming implementation. This reduces the amount of code required to be written for new applications. Traffic decisions can simply be linked to the message content of the service. Dynamic information need not be stored in configuration files, saving application developers the burden of parsing.

Data stream processing is able to schedule for non-persistent or persistent application requirements based on key areas within the data stream, called *data stream resources*. These data stream resources are listed in the table below.

Data stream resources	Non-persistent	Persistent
Source Address	✓	✓
Destination Address	✓	
Source Port	✓	✓
Destination Port	✓	
Type of Service	✓	✓
Pattern	✓	✓
Pattern Mask	✓	
Pattern Value	✓	✓

Whether the application uses Voice over IP, XML, DHTML, WML, wireless protocols, or any other protocols, data stream processing surfaces the data content for the application.

### Scenario: Persistence based on XML transactions

This scenario involves an e-commerce client putting forth a bid transaction. The network administrator wants to grant priorities to their largest and best customers. The company, Anystore Corp., is one of the biggest clients. The company does not want to implement cookies, but has decided instead to use data stream processing to determine persistence based on a list of companies that match their criteria. When issuing a transaction, the resultant XML may look like the following example.

**Figure 2. Client's transaction in XML.**

```
<COMPANY>ANYSTORE CORP</COMPANY>
<CONTACT>JOHN DOE</CONTACT>
<TEL>8001112222<TEL>
<EMAIL>JDOE@XYZ.COM</EMAIL>
...
<ITEMNO>101</ITEMNO>
  <PARTNUM>12345-T1</PARTNUM>
  <QUANTITY>2000</QUANTITY>
  <PRICE>499.95</PRICE>
...
```

This scenario illustrates how data stream processing is ideal for legacy environments, where users can not alter applications. This is a commonplace scenario. Most users are not privileged to have a new e-business architecture built from the ground up utilizing the latest technological advances. Resonate has taken this into consideration with data stream processing. Old applications can continue to be used; their known quirks can be worked around with DSP.

*(To refer to a code example, please see "Appendix B: XML Scenario.")*

## Scenario: Optimizing transactions based on SOAP HTTP requests

Data stream processing is ideal for handling new Internet communication methods that facilitate complex computing environments, such as SOAP (Simple Object Access Protocol). SOAP can help to determine what kind of Web services are requested by the client, which results in selecting the processing power appropriate for the transaction. In this scenario, a client makes a request for a stock quote. The actual quotes request is indicated in the SOAPAction HTTP request header. This is a request that must be processed as quickly as possible. The data stream processing rule lists the servers that are multiprocessor machines or have the most resources available, thereby scheduling the request to the machine with the highest processing power available.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/quotes"

<env:Envelope xmlns:env="http://www.xyz.com/soap-envelope" >
  <env:Body>
    <m:GetLastTradePrice
      env:encodingStyle="http://www.xyz.com/soap-encoding"
      xmlns:m="http://example.org/quotes">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </env:Body>
</env:Envelope>
```

Fine granularity of scheduling request can be attained by examining fields that indicate specific application procedures, or methods. Decisions can be made to send requests that are CPU-bound to multiprocessor machines; requests that are I/O-bound can be sent to machines with high-end storage and bandwidth facilities.

## *Remove barriers, create opportunities*

There are many barriers that keep applications from delivering on requests. In some cases, these barriers are due to application age factors, new business application environments, and performance limitations. Data stream processing lowers these barriers to keep applications performing, simultaneously creating new opportunities for the applications to move to.

## Scenario: Scheduling based on application type

With data stream processing, users can serve content based on the type of media that the client is accessing, by creating scheduling rules. This is similar

to the way users can define a style sheet in HTML 4.0, detailing the device to be used for serving content like print or audio.

Examine the following HTML 4.0 definitions:

```
<LINK rel="stylesheet" media="aural" href="corporate-aural.css"
type="text/css">
<LINK rel="stylesheet" media="screen" href="corporate-screen.css"
type="text/css">
<LINK rel="stylesheet" media="print" href="corporate-print.css"
type="text/css">
```

Data stream processing directs the inbound requests to servers that can support screen and print information, since that is what the application was originally designed for. However, if a request for speech media is made, the request may be redirected to a "sorry server" to play a canned message indicating that the service is unavailable. Here, the company recognizes the application type, and rather than sending an error message or inappropriate content, is able to answer the request appropriately.

## *Realize higher service levels*

E-businesses try to maintain service levels and protection against application failure by ensuring the availability of the Web server, the application server, and the storage device. Many often forget that a critical step in the e-business transaction is when the data is delivered to the client, for display on the client's browser. Does the returned data affect service levels? If the data displays an error condition instead of the actual data, then the answer is a resounding 'Yes!'

The active service level management aspects of real-time *monitoring* and policy-based *control* is suitably manifested in the data stream processing handling of outbound data replies. Any field within the data packet can be monitored for the expected content, or for the unexpected error condition. Upon meeting specific criteria, possible actions can be taken to ensure delivery of content. Those actions include:

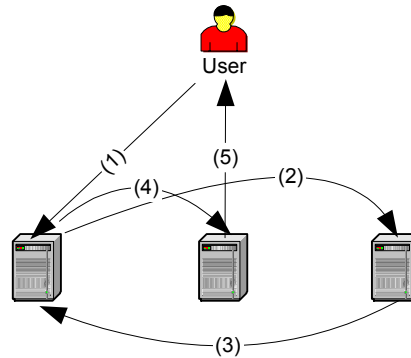
- Redirect the original inbound data request to another server;
- Change the server weight of the outbound server; or,
- Remove the outbound server from the Central Dispatch cluster.

These actions are possible on every node where Resonate Central Dispatch is installed. Central Dispatch's kernel-based agent, RXP, takes advantage of its location above the NIC interface. The RXP is able to check all outbound data replies *before* the data is sent to the client. This is the best way of ensuring high service levels: By checking for expected data patterns, error responses, or any binary value or string that requires an immediate action on the content server.

Redirection is most often used in situations where it is desirable to have another server retry the request rather than forcing the user to send their

request again. It is an elegant way to hide the problem from the client while allowing time for corrections to be made. The following illustration shows how Resonate Central Dispatch handles redirection based on outbound scheduling rules.

**Figure 2. Redirection based upon outbound rules.**



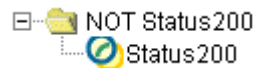
#### **Outbound parsing: Redirect Action**

- (1) Client sends a request to the VIP (Scheduler).
- (2) Scheduler forwards the request to a content server.
- (3) The application handles the request, sends the reply. RXP is checking the packet for the pattern specified in the outbound rule. The pattern (possibly an error string) matches, and sends a message to the Scheduler indicating that the pattern matched.
- (4) The Scheduler marks the node, and schedules the original request to one of the remaining servers.
- (5) The application handles the request, sends the reply. RXP checks the packet for the pattern, does not find a match, and let's it go to the client.

### **Scenario: Redirecting the client request due to a reply error condition**

In many cases, it is desirable to redirect the original request to another server. Instead of letting the user do it, Resonate Central Dispatch can redirect the request automatically. Consider a situation where the client must be ensured a valid page of data. In this scenario, the client is requesting airline flight information. Data stream processing can monitor the outbound data reply and respond on the presence of any error, i.e., any HTTP request not returning status code 200. This process is accomplished in two easy steps.

First, an outbound data reply rule is created:



```
Expression Type: ASCII
Start Offset: 0
End Offset: 100
Pattern Length: 12
Pattern: HTTP/1.1 200
Name: Status200
```

Second, this is applied as an HTTP service with an action command of *redirect*. Resonate Central Dispatch's data stream processing will automatically redirect the client to the desired server.

### **Scenario: Redirecting the client request based on the SOAP HTTP response**

SOAP provides a method to indicate where specific problems exist during the request transaction. If an error occurs using SOAP, an HTTP 500 "Internal Server Error" is required to be returned. SOAP provides the detail through fault codes. If a request is not processed for any reason, SOAP requires that a fault must be generated. But, this error information is too vague to base decisions on. What the user needs is specific error detail in order to address the specific problems.

In this scenario, a request will be redirected to another server if the outbound server has been deemed to be incapable of handling the request through the existence of Server class errors. Other fault codes, such as Client class of errors, would not require request redirection; only Server class of errors, which refer to the specific server that is processing the request. Data stream processing rules can be created to determine the following sequence:

1. If HTTP message "500 Internal Server Error" is present (an error has occurred), and;
2. If "<faultcode>env:Server</faultcode>" is present (some server error), then;
3. Redirect the original request to another server.

Another more complex situation would be when the user wants to check application errors to determine why the request could not be processed. In the example below, the <errorcode> field indicates application reasons that could further determine whether the condition is permanent, and the server should be removed from the cluster.

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

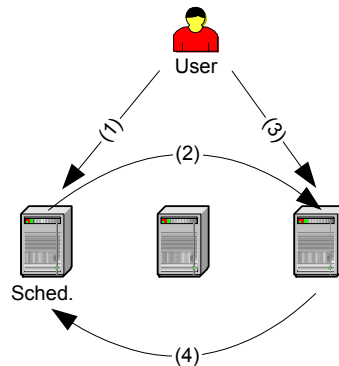
<env:Envelope xmlns:env="http://www.xyz.com/soap-
envelope" >
  <env:Body>
    <env:Fault>
      <faultcode>env:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>
        <e:myfaultdetails
xmlns:e="http://example.org/faults" >
          <message>My application didn't work</message>
          <errorcode>1001</errorcode>
        </e:myfaultdetails>
      </detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Sometimes redirection is not enough, and it is desired to reduce traffic to a content server to allow the server time to recover. This is where the ability to decrement, or *downgrade*, the server weight can be automatically accorded to a server's response. Once the server is deemed healthy, the rules can automatically increment, or *upgrade*, the server weight to ensure that traffic scheduling increases to that server.

In rare, but vital situations, the need may arise where the server response warrants the node to be removed until the problem can be resolved. The outbound rule can select the *disable server* action to ensure that the node is removed from further scheduling activities.

The figure below shows how these actions are accomplished by Resonate Central Dispatch.

**Figure 3. Upgrade / downgrade / disable action on outbound rule.**



#### **Outbound parsing: Upgrade / Downgrade / Disable Action**

- (1) Client sends a request to the VIP (Scheduler).
- (2) Scheduler forwards the request to a content server.
- (3) The application handles the request, sends the reply. RXP is checking the packet for a pattern specified in the outbound rule. The pattern (possibly an error string) matches, and sends a message to the Scheduler indicating that the pattern matched.
- (4) A message is sent to the Scheduler to take one of the following actions:
  - Increment the server weight by one (upgrade)
  - Decrement the server weight by one (downgrade)
  - Disable the server

The Scheduler takes one of the actions on the server.

#### **Scenario: Change the server weight of the outbound server**

If the content server is not responding properly or in a timely fashion, an automatic action can be taken to change the server weight of the affected server. If an outbound rule is defined with an automatic action to *downgrade server*, the server weight will automatically be reduced by 1 each time the rule meets the required conditions. Conversely, if the rule is defined with an automatic action to *upgrade server*, the server weight will automatically be increased by 1.

## **Scenario: Disable the outbound server**

If a content server response is considered serious enough, the server can be removed from the cluster. By *disabling* the server, the automatic action via the outbound rule ensures that no traffic is scheduled to the malfunctioning content server. The incident is logged and an automatic e-mail alert is sent to the administrator to work on the affected server at a later time.

Alternative products that do outbound data parsing are not as comprehensive as Resonate Central Dispatch, which has data stream processing capabilities. Hardware Web switches, for example, can look at the outbound data packet, but are limited to checking a fixed portion of the HTTP header; they look only for error codes in a specific location in the data stream. Resonate Central Dispatch is able to look at the HTTP header, and the *entire* data stream.

## ***Summary***

With Resonate Central Dispatch's data stream processing, e-businesses can ensure service levels and adapt quickly to changes brought on by the natural evolution of the e-business application infrastructure. This is the one solution that can provide the custom traffic management capability that will ensure the high service levels all clients expect.

Data stream processing is available in Resonate Central Dispatch version 3.3 and up.

---

## Appendix A: HTTP header parsing scenario

This section describes the HTTP header parsing example in greater detail.


To describe the parsing rules in pseudo-code, the rules would appear as:

```
if string[0:499] contains "GET /" AND
    if string[0:499] contains "User-Agent:" AND
        string[0:499] contains "MSIE" then
    {
        Schedule request to web servers
    }
```


The resultant expressions created in the CDMaster interface would resemble the following syntactical expressions.



```
if string[0:499] contains "GET /" AND
    if string[0:499] contains "User-Agent:" AND
        string[0:499] contains "MSIE" then
    {
        Schedule request to web servers
    }
```

The resultant expressions created in the CDMaster interface would resemble the following syntactical expressions.

 GetRequest

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 5
Pattern: GET /
Name: GetRequest
```

 AND (UserAgent AND MSIE)

-  UserAgent
-  MSIE

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 11
Pattern: User-Agent:
Name: UserAgent
```

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 4
Pattern: MSIE
Name: MSIE
```

---

## Appendix B: XML scenario

This section describes the XML parsing scenario in greater detail.

The inbound request rules may appear as follows:

```
if string[0:499] contains "<COMPANY>ANYSTORE
CORP</COMPANY>" OR
    string[0:499] contains "<COMPANY>BCD CORP</COMPANY>"
OR
    string[0:499] contains "<COMPANY>XYZ INC</COMPANY>"
```

In addition, the application needs to ensure that only data delivered to destination ports 5000 to 6000, inclusive, are scheduled to a server.


An additional inbound request rule would appear as:

```
...OR (if destPort >= 5000 AND destPort <= 6000)
{
    Schedule request to web servers
}
```

The resultant expressions created in the CDMaster interface would resemble the following syntactical expressions.

 AND  


```
Start Port: 5000
End Port: 6000
Name: PortRange-1
```

 OR (AnyStore\_Corp OR (BCD\_Corp OR XYZ\_Inc))

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 32
Pattern: <COMPANY>ANYSTORE CORP</COMPANY>
Name: AnyStore_Corp
```

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 27
Pattern: <COMPANY>BCD CORP</COMPANY>
Name: BCD_Corp
```

```
Expression Type: ASCII
Start Offset: 0
End Offset: 500
Pattern Length: 26
Pattern: <COMPANY>XYZ INC</COMPANY>
Name: XYZ_Inc
```

Resonate is a registered trademark, the Resonate logo, Keeping E-Business Open for Business, Resonate Central Dispatch, is a trademark of Resonate, Inc. All other trademarks are the property of their respective owners. Copyright © 2002 Resonate, Inc. 1/02 TWP014.

