SeeBeyond ICAN Suite

Working with Collaboration IDs

Release 5.0.5 for Schema Run-time Environment (SRE)

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050406091816.

Contents

Chapter 1

The Anatomy of a Collaboration-ID

A Collaboration-ID, as produced by the SeeBeyond Collaboration-ID editor, has two components:

1. An Event Type Definition
2. A set of rules that apply tests to the Event Type Definition

When an Event is passed to a Collaboration-ID, two evaluations are performed:

1. The Event is first tested against the Event Type Definition (an Event parse) to see if the Event conforms structurally to the Event Type Definition. This test also checks for any requirements defined by input tags definitions included in the Event Type Definition.
2. The associated rules are then evaluated against the Event's content.

If both the event parse and the associated rules complete successfully, the Collaboration-ID function returns a boolean true; but if either the event parse or any of the rules fail, the Collaboration-ID function returns a boolean false.

1.1 The ID Service in Workslices

The workslice component in the e*Ways and the BOBs is the executing thread that operates one or more Collaborations. Each Collaboration supports any of a number of Collaboration Services, including the SeeBeyond Technology Corporation™ (SeeBeyond™) default services Copy, Monk Collaboration, and Monk ID.

Collaborations in inbound e*Ways, which process Events from external sources, handle failures differently from those in BOBs or outbound e*Ways. In an inbound e*Way, if any of the collaborations fail, all collaborations in the workslice are considered to have failed, and the Event is NAKed. In BOB's and outgoing e*Ways, each defined Collaboration succeeds or fails individually.

In earlier SeeBeyond products such as e*Gate 3.5 or 3.6 (formerly known as DataGate), incoming messages were identified and labeled; this ID label was then used to drive the routing and translation process. In e*Gate, Events can be identified and processed before being forwarded to the queuing service for distribution. Both the Monk ID Service and the Monk Collaboration Service can perform the identification function.

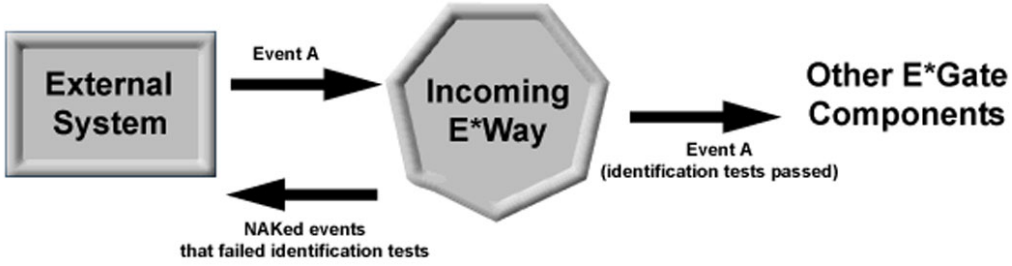
The Monk ID Service can be used in the workslices to include Collaboration-ID functions directly in the process flow. Collaboration-ID functions use "iq-put" function

calls to insert Events directly into queues. Using this feature, Collaboration-IDs can identify an Event, call one or more sub-collaborations to modify the Event content, and distribute the resulting Events to one or more queues. Alternatively, the same results can be achieved within the Monk Collaboration environment: after the initial event parse, the Collaboration can perform its own ID tests before processing and enqueueing any output events. By using the Monk Collaboration Service to perform identification functions in the inbound e*Way workslice, you can provide Collaboration-ID functionality without the restrictions of the Monk ID Service.

1.2 Collaboration-ID/Incoming e*Way Architecture

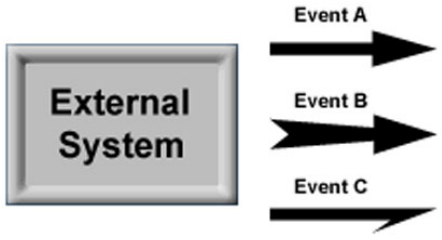
In an extremely simple system, an external source sends a single Event Type to an incoming e*Way. The incoming e*Way applies an identification test, forwarding Events that pass the test and returning events that fail to the external system.

Figure 1 A Simple ID Test



In a more complex system, however, the simple model that NAKs unidentifiable Events may be the wrong approach. Consider the following example:

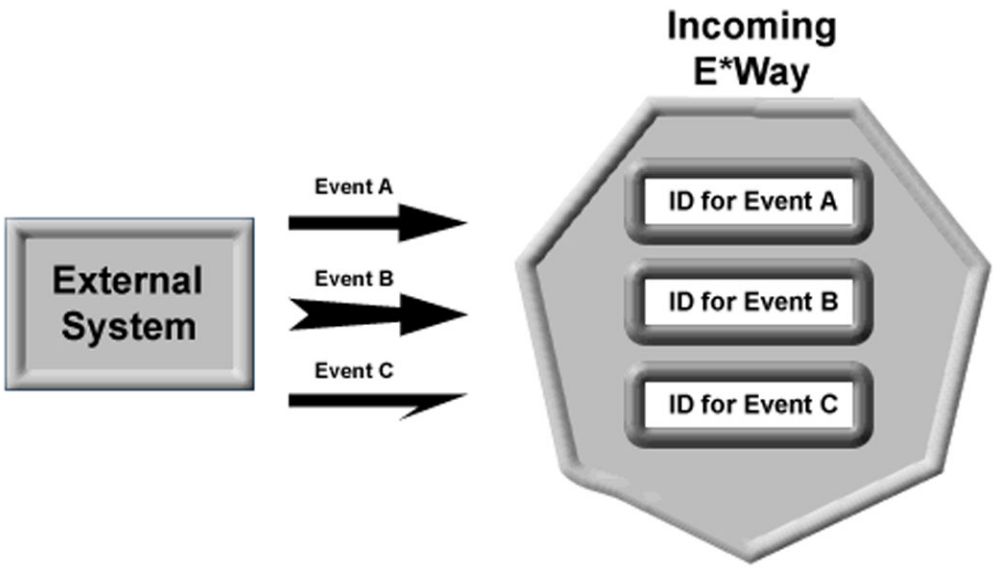
Figure 2 Multiple Event Types from a Single Source



In this example, an external system sends the e*Gate system any of three types of Events (one at a time). Each one of the Events is acceptable to the e*Gate system, but the Events are dissimilar and must be processed separately.

If we use the scheme discussed in Figure 1, where unidentified Events are NAKed and returned to the server, the configuration would look like the following:

Figure 3 Applying Multiple Collaboration-IDs

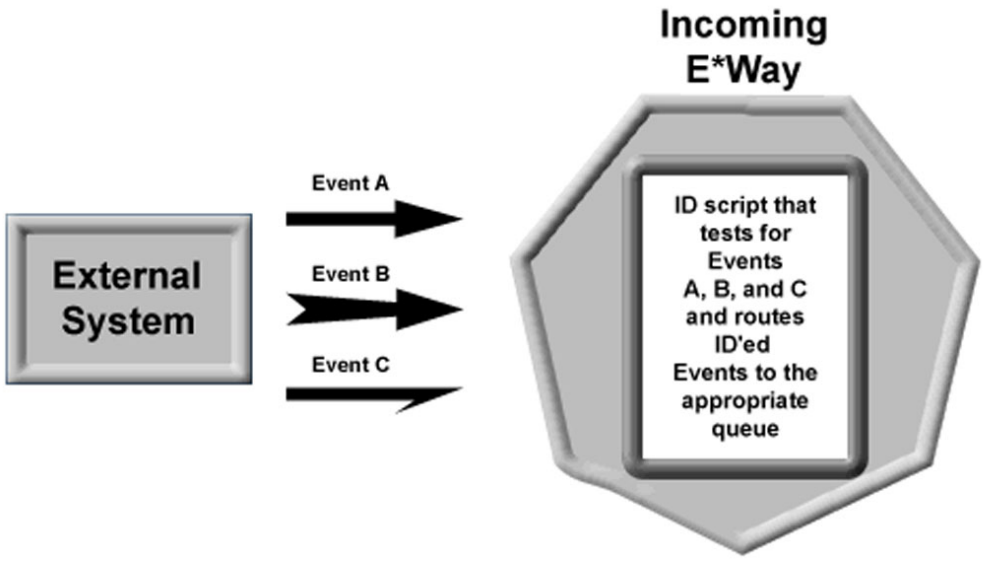


This scheme has two major drawbacks:

1. Each ID parses the Event separately. This means that a single Event would be parsed three times (once per ID), slowing system performance.
2. Although a given Event may pass the ID for its type (Event A passes the test for ID A, for example), the Event will fail the ID for the other two Event Types. Therefore, any given Event will be NAKed and sent back to the external system even if it is acceptable to one of the IDs-effectively, this means that no Event will pass the ID e*Way to enter the e*Gate system.

A better scheme is shown in Figure 4:

Figure 4 Single Collaboration Rules Script for Multiple IDs



In this scheme, a single ID script parses each incoming Event. Events that match ID A are sent directly to Queue A via the "iq-put" function call. Events that match IDs for B and C are similarly put directly into appropriate queues. Once in those queues, other e*Gate components will continue processing the Events.

In this scheme, only Events that fail all the tests within the ID script will be NAKed. Optionally, you could provide an error-handling routine in the ID script that forwards Events that match no IDs to a fourth "bad Event" queue, perhaps for further processing or later examination. This refinement moves all error processing to the e*Gate system and never returns NAKed Events to their originating system.

This scheme has several advantages:

1. It keeps all ID functions within the same script and within the same workslice, improving system performance.
2. It enables you to process both successfully identified Events and problem Events within the e*Gate system.
3. Most importantly, it will perform its intended function-the three-ID alternative will not.

See the sample code at the end of the next section for an example of how this scheme could be implemented.

Chapter 2

Upgrading e*Gate 3.5/3.6 Environments to e*Gate 4 or Higher

If you need to migrate existing e*Gate 3.5/3.6 implementations to e*Gate, use the following technique to encapsulate existing e*Gate 3.5/3.6 ID and Translation functions within an e*Gate Monk Collaboration in a workslice.

Using the example discussed in the section **"Collaboration-ID/Incoming e*Way Architecture" on page 5**: In e*Gate 3.5/3.6, an inbound message is passed through three different IDs, and if an ID is successful, the related translation is called. The result of the translation is then sent on toward a destination system. Pictorially, the situation looks like

```
Input message ->
         ID1  -> (if true) -> xlate1 -> (if successful) -> route
         ID2  -> (if true) -> xlate2 -> (if successful) -> route
         ID3  -> (if true) -> xlate3 -> (if successful) -> route
```

In e*Gate, this can be implemented in the Monk Collaboration service as follows:

```
Input Event ->     Event
```

parsed into a single collaboration, using a single-node Event Type Definition as the input Event, passing all data in the "~input%msg.data" Event Type Definition node.

The body of the function looks like:

```
(if (ID1 ~input%msg.data)
  (begin
    (set! xlate_result (xlate1 ~input%msg.data))
    (iq-put event_to_send_to xlate_result ... )
  )
  (begin
    (display "Not id'ed with ID1")
  )
)
(if (ID2 ~input%msg.data)
  (begin
    (set! xlate_result (xlate2 ~input%msg.data))
    (iq-put event_to_send_to xlate_result ... )
  )
  (begin
    (display "Not id'ed with ID2")
  )
)
(if (ID3 ~input%msg.data)
  (begin
    (set! xlate_result (xlate3 ~input%msg.data))
    (iq-put event_to_send_to xlate_result ... )
```

```
        )
        (begin
          (display "Not id'ed with ID3")
        )
      )
```

where "ID1" through "ID3" are whatever identification tests you wish to perform to identify the Event.