CHAPTER

40

Using Agent Services

Introduction

Agent services is used with compute services and messaging services to provide client/server based task management for the nodes across your network. An *agent* is SAS source code that is launched by the DOMAIN server to support the execution of a task. After the task has finished executing, an agent may be designed to send a *completion notification* to a message queue for the client application, or the client can simply check the DOMAIN server to see if the agent has completed execution.

The source that composes an agent may be predefined and stored with the DOMAIN server, or the source may be defined dynamically with the request for the agent to execute.

Client/server-based agents provide the following types of services:

- distributed agent processing
- periodic agent processing
- conditional agent processing
- parallel agent processing.

These services can be used together to maximize the flexibility and functionality of a client/server application. For example, using agent services, an application can be designed to execute autonomously on a periodic basis. The application can make conditional decisions to satisfy its goals, which includes submitting other processes for execution on various hosts on the network.

Distributed Agent Processing

Using compute services and distributed messaging, agents can be used to distribute work across nodes on a network that is most efficient for a client/server application. This distributed service was designed for client/server applications which benefit from unattended, asynchronous execution.

Agents transparently use SAS/CONNECT to sign on to a designated location using a SAS spawner and a designated user identity, and then execute the agent's source code.

As agents are processed, the log and output windows are spooled until the agent has finished executing. The agent then signs off the designated node. Upon completion, a monitoring client may retrieve the spooled execution log and output, merge it with the local output or purge it.

In addition, agents may be designed to use the indirect-messaging facility to communicate information to client applications by using message queues. This allows agents to be constructed to generate alerts and/or dynamic reports as they execute. These conditions and information are then manifested as messages and written to queues, which are monitored by interested clients.

The initiating client SAS session is not required to remain active while the agent executes (unlike a conventional local or remote SAS/CONNECT session configuration).

Periodic Agent Processing

Periodic agent processing is a client/server-based implementation of a task scheduler. SAS has expanded the traditional task scheduling service to the client/server environment by allowing agents to run on remote hosts across a network. In general, other task management products are limited to running tasks only on the local host.

Agents can be defined so the DOMAIN server schedules it to run at a specific date and time or on a repetitive (periodic) run schedule. Periodic scheduling can be defined to start on a specific hour and minute on either a day-of-the-week schedule or a month-and-day schedule.

Periodic report generation agents can be defined to distribute electronic reports to clients in the form of message attachment packages that are delivered to queues monitored by desktop presentation applications.

For example, throughout the day, decision support transactions can be queued for off-hours execution. At a specified time, an application server agent can be scheduled to service the transaction queue and return results to the client that made the request. The results might be in the form of a response message that has the results delivered as attachments to the designated message queue.

Conditional Agent Processing

An agent-based application can be implemented to take advantage of conditional sequential-step logic. Having conditional capabilities allows an agent to spawn additional agents based on conditions that are encountered during execution.

For example, a filtering agent can be constructed, which processes operational data stores and searches for anomalous entries. Upon locating such an anomaly, a conditional agent may deliver an alert to an audit queue, which is monitored by a manager who has responsibility for the operational system being filtered. As well, other conditional logic may spawn an extraction agent to generate consolidation summaries.

After the summaries are ready, another agent may stage those summaries in the form of data set attachments to update messages that are delivered to a designated message queue. Replication agents might be used to monitor the queue. After the message is received, the replication agents execute on the decision support servers that provide daytime access, completing the replication process. If problems are encountered, alerts can be delivered accordingly.

Parallel Agent Processing

Given that agents can spawn other agents, independent tasks can be partitioned into subordinate agents. These subordinate agents can be launched by a controlling agent to execute on various nodes within the network. This way each agent can be processed at the same time in parallel with one another.

Using a completion notification message queue, the controlling agent could determine when each of the subordinate agents have finished executing. The controlling agent could also contain logic that synchronizes the work that is performed by the subordinate agents.

Client/server applications can use parallel processing to minimize overall elapsed time of a task's execution. By breaking the application into discrete encapsulated tasks, each independent task can be processed concurrently to reduce the execution time rather than running each task sequentially.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/CONNECT User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 537.

SAS/CONNECT User's Guide, Version 8

ISBN 1–58025–477–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, September 1999