The
Simple
Event
Notification
Service
Environment
Architecture

Revision 0.1
19 February 1996

Underscore™

Copyright © 1996 Underscore, Inc.
All Rights Reserved

This document may be distributed in any form, electronic or otherwise, provided that it is distributed in its entirety and that the copyright and this notice are included.

Underscore and the Underscore logo are trademarks of Underscore, Inc.

Author's Contact Information:

Comments, suggestions and inquiries regarding SENSE may be submitted via electronic mail to **JK Martin** at `jkm@underscore.com`.

Acknowledgments:

I would like to thank Binnur Al-Kazily of Hewlett-Packard and the many other members of the Printer Working Group their continued support of the SENSE Project.

Where this document can be found:

This document can be obtained from:

`ftp://ftp-out.external.hp.com/snmpmib/sense/arch001.ps`

Technical contributors:

Richard Landau, Digital Equipment Corporation

Jeff Schnitzer, Underscore, Inc.

Mike Timperman, Lexmark

Table of Contents

§

Overview

This document describes the “Simple Event Notification Service Environment,” abbreviated as “SENSE” throughout this and other related documents and correspondence.

What is SENSE?

SENSE is a component architecture designed to facilitate the collection and distribution of events within a heterogeneous network environment.

SENSE may be described as being:

- A facility used by an application to receive events from one or more sources
- An open framework into which components may be easily integrated to provide new sources of events and/or consume events to generate useful information
- A protocol used by cooperating components to interact for the purpose of sending and receiving events and descriptive information about each other

Brief History

The concept of SENSE was originated by JK Martin (Underscore), Rick Landau (Digital) and Mike Timperman (Lexmark) in October, 1995, in response to an identified need for the transmission of events from network printer devices to network management applications specifically designed to monitor such devices. As work progressed it became readily apparent that the need for such notification services are pervasive in today’s widespread distributed network environments.

Current Project Status

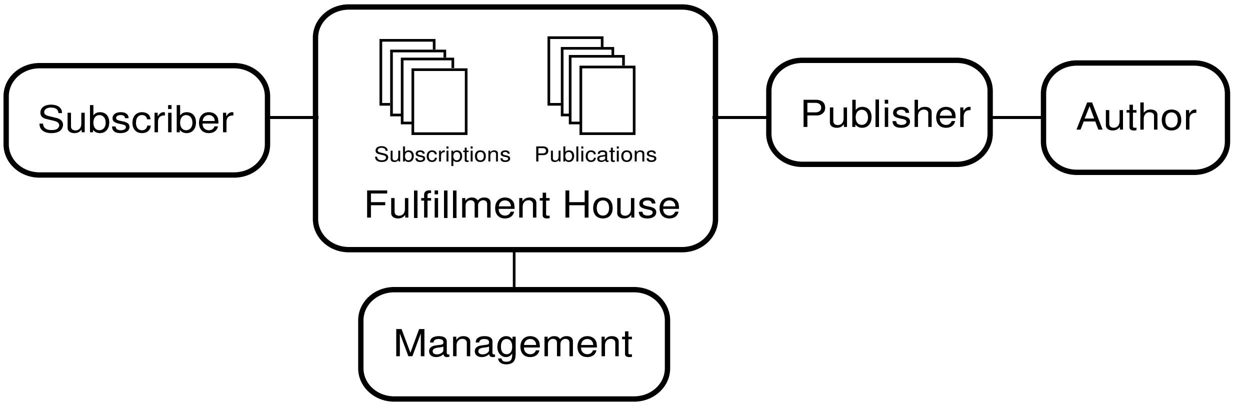
The design and implementation of a rapid prototype of SENSE was undertaken by Underscore in late 1995 to test the basic concepts described in this document. As of this writing the current schedule for this effort is:

Completed Specifications:	April 1996
Pre-Beta Release:	May 1996
Public Beta Release:	July 1996
Production Release:	October 1996

Underscore expects to arrange for delivery of pre-Beta code to interested parties in late March, 1996.

The Magazine Publishing Model

The architecture of SENSE is based on a simple model for magazine publishing as shown in the following diagram:



This model is referred to within this document as the "Magazine Publishing Model," or simply the "MPM." The key components of this model are:

Publication A collection of information of interest to one or more persons.

Subscription A time-constrained association between a Publication and a Subscriber whereby the associated Publication is delivered to the associated Subscriber when the Publication is published during the life of the Subscription.

Author The person that generates the content that comprises a Publication; multiple Authors may contribute to the content of any given Publication.

Publisher The person responsible for coordinating the efforts of Authors associated with a Publication, and ensures the Publication is published as specified; a single Publisher may be responsible for multiple Publications.

Subscriber The person interested in receiving one or more Publications; interest is primarily based on content, but other factors can play a role, including geographic focus, language, etc.

Fulfillment House The “clearinghouse” to which Publishers submit Publications that are subsequently delivered to Subscribers.

Management Coordinates the operations of the Fulfillment House to ensure that all activities are performed in an efficient manner consistent with established policy; reserves the right to accept or cancel Subscriptions or Publications at will based on policy.

Requirements and Constraints

A primary motivation for developing SENSE is to improve on the delivery of critical event messages as compared to the SNMP TRAP model. In particular, SENSE should improve on these deficiencies in the SNMP TRAP mechanism:

- No standard method for adding or removing TRAP destination addresses, either statically or dynamically;
- All TRAP messages are directed to a fixed UDP port number, thereby forcing the use of a non-standard demultiplexing mechanism on hosts where multiple TRAP recipients operate;
- Only a single copy of the TRAP message is delivered to any given destination address using unreliable datagram technology; if the datagram gets lost, then the recipient is unable to determine that a TRAP message was ever sent.

SENSE must satisfy the following functional and operational requirements (not listed in any particular order):

R.1 Reasonably reliable receipt of Event Messages

A key requirement is for a Client to expect reasonably reliable receipt of Event Messages. The term "reasonably reliable" is used to denote the fact that a Server should make multiple attempts to deliver the message to the Client. It should be noted that absolute reliability is not considered practical, and thus, not considered as a requirement.

R.2 Datagrams are used at the transport layer

Since stream-oriented protocols are typically considered too "heavy" for lightweight services, datagrams should be used for all SENSE protocol implementations. While not called out as a requirement in the above list, it is expected that the SENSE facility should be implemented for use with at least the following datagram-oriented transports:

- IP/UDP
- NetWare IPX
- AppleTalk DDP

Other datagram-oriented transports are not necessarily precluded from implementation.

R.3 A well-known transport address is defined for common use

To facilitate interoperability, the Server should be able to operate using a standard, "well known" transport address.

R.4 A Server can operate using any transport address

The Server should be able to operate with any defined address within the target transport environment. This will, of course, require that Clients know of and use the non-standard transport address. This requirement is specified so as to allow a Server to operate in an environment in which the standard transport address is already in use, or when an unprivileged user wishes to operate a Server when the standard transport address requires privileges. This requirement should be considered optional for an implementation.

R.5 Highly portable design

The overall capabilities described by SENSE do not require complex nor significant resources. Hence, the SENSE specification should not require the use of facilities that are not readily and/or inexpensively available within the many desktop and server operating systems. Examples of facilities that would be considered not readily available are: DCE, ToolTalk, CORBA, Kerberos, and other such facilities that are either prohibitively expensive or unavailable on many platforms. Use of such facilities is considered optional.

R.6 Multiple sources of events can be managed by a single Server

A Server should be able to represent any number of Event Sources. No minimum or maximum number of supported Event Sources should be formally specified.

R.7 A Server can be queried to determine the set of event sources managed by the Server

A Client should be able to request the list of event sources supported by the Server.

R.8 A Server can be queried to determine its operational parameters

A Client should be able to request a list of operational parameters and their values from the Server.

R.9 A simple loopback capability to determine Server existence

A Client should be able to “ping” the Server to determine whether the Server is operating at the target transport address. This requirement could be reasonably satisfied through the implementation of Requirement R.8 above.

R.10 A client dynamically registers for receipt of events from multiple event sources

A Client should be able to dynamically request of events from a source represented on a Server. The period of time during which the Client continues to receive events is fixed; once this time period is exceeded, the Server automatically ceases transmission of the events without further action by the Client.

R.11 A client specifies the network address to which all Event Messages are directed

When the Client requests the receipt of events from a source, part of the request includes the destination transport address (network address and transport port number) to which all Event Messages are delivered.

R.12 A client can cancel receipt of events at any time

A Client is free to cancel receipt of events before the assigned time period expires.

R.13 Events are asynchronously transmitted by the Server to each registered client as the events are received by the Server

The Server should send Event Messages to the network/transport address specified by the Client at event request time as such events occur. The Server will continue to periodically retransmit an Event Message until either the Server-defined retransmit count expires, or until the Client acknowledges receipt of the Event Message.

R.14 Clients acknowledge receipt of events

A Client must acknowledge receipt of an Event Message so that the Server will cease retransmission of the Event Message.

R.15 The content and format of an Event Message can have an opaque data component that has no relationship to the underlying SENSE protocol

An Event Message can optionally contain a data component that is not related to nor relevant to the SENSE system; instead, the receiving Client is expected to be familiar with the format of such messages based on the associated event source.

R.16 A Server must be able to control resource consumption

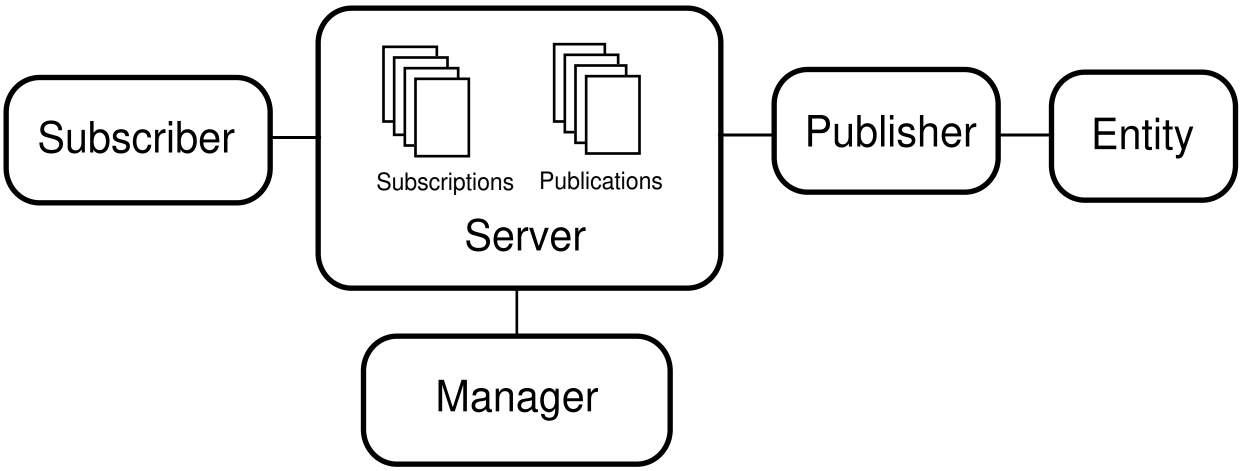
A key aspect of the SENSE facility is to be highly "Server-oriented" with respect to implementation and performance. In particular, the Server should be allowed to arbitrarily implement the values for such parameters as:

- Maximum number of clients
- Maximum registration period
- Maximum number of retries for delivery of event messages

It is expected that the values of these parameters (and probably many others) will be part of the response to a request for a Server's operational parameters as described in R.8 above.

Architectural Model

The SENSE Architecture Model closely resembles the previously described Magazine Publication Model, as shown in the following diagram:



Overview

The SENSE architecture revolves around two collections of definitions that are closely related, namely, *Components* and *Objects*, each of which are organized into loosely coupled classes.

The concept of a *Component* within SENSE is very important in that it represents the smallest unit of interoperability within the SENSE framework. The SENSE business model (not described in this document) revolves around the ability for customers and vendors to easily create, integrate and operate Components in a "plug-and-play" manner within the overall SENSE framework.

A fundamental aspect of a Component is that it is usually perceived as a relatively *active* element, such as an application or similar program unit. While a Component may have associated data, it is the executable aspect that distinguishes a Component from an Object.

An *Object* within SENSE refers to a collection of data used by Components to effect the capabilities of SENSE. Unlike Components, Objects are relatively *passive* in nature; there are no executable aspects surrounding any given Object type in SENSE. Objects are created, accessed and destroyed by Components.

The concept of a *Publication* is critically important within SENSE. A Publication is not categorized as an official Component, but rather as one of the defined Object types, since a Publication does not satisfy the fundamental Component criteria of being an *active* element, such as a program.

Objects, including Publications, are described in more detail later in this document, however a brief definition of the term "Publication" is necessary in order to properly introduce the Component classes within SENSE, since Publications are essentially the central focus and common thread among all Components.

Briefly, a Publication is the representation of the entity that generates events; that is, where ever the term "event source" is used, imagine that event source as a "Publication" within SENSE.

For example, when SENSE is applied to the problem domain of printer management, a Publication would represent a single printer. A party interested in receiving events from a specific printer would register for receipt of events from the Publication that represents the printer. To determine which Publication to select for event reception, a party would also request descriptive information about one or more of the available Publications.

Components

A *Component* is an "active" element within SENSE in that a Component is typically implemented as some kind of program unit, such as an application, or library used within an application. A Component may reside on any network host capable of communications with compatible SENSE Components.

Component Classes

There are two primary classes of Components within SENSE:

- Server
- Client

All communications activities within SENSE are conducted either to or from the Server; that is, no direct communications are conducted between any two Client Components.

Server Class

The "Server" class contains one member: the SENSE Server. The SENSE Server is essentially the "center of the SENSE universe" in that all interactions between Components exist between one of the Client classes and the Server.

The Server performs these functions:

- Manages Client Sessions
- Provides a modest level of Directory Services
- Receives Event Messages from Clients that manage Publications on the Server
- Propagates Event Messages to interested Clients
- Publishes its own Publication (set of events)

An important aspect of SENSE communications is that Servers do not communicate between each other. No attempt is made for one Server to "know about" other Servers and the Publications that might be registered on those Servers.

Client Class

The "Client" class is further subdivided into four subclasses, each performing a specific role.

Publisher	Registers Publications with the Server; collects and submits to the Server events from the event source represented by the Publication.
Subscriber	Dynamically registers with the Server to receive events from one or more event sources, and queries the Server for the list of available Publications, and properties of those Publications.
Manager	Performs management activities on the Server in a privileged manner; very little work has been done to date for this Client class.
Transient	A special kind of Client that only exists to submit queries (informational requests) to the Server; until a Client registers as either a Subscriber or Publisher, it is considered a "Transient" Client.

Objects

An *Object* is a "passive" element within SENSE in that it has no executable aspects whatsoever. Objects exist only through creation by Components. A SENSE Object is usually ephemeral in that it does not typically exist beyond the life of the Component responsible for its existence.

Object Classes

Object classes may be viewed as two collections of classes, based on the entity or concept represented by the class.

Active Objects

An *active* Object is one that is relatively long lived, and where one or more of its Properties typically change value during its lifetime.

An active Object is represented as a collection of defined *Properties*. A Property is a name-value pair of character strings that closely resemble an "environment variable" found in many operating systems. A collection of Properties is called a *PropertyList*.

A critically important aspect of active Objects is that while each defined Object has a known set of defined Properties, the Object may also have other Properties attached to it by a Component. When Objects are exchanged between Components, the exchange is handled such that the receiving Component accepts the complete PropertyList that currently describes the Object without knowing that the Object's PropertyList contains one or more Properties unknown to the receiving Component.

The ability of a Component to “attach” arbitrary Properties to an active Object without having to coordinate the handling of such Properties between interested Components is a fundamental aspect of the extensibility of the SENSE architecture.

The set of active Objects includes:

Publisher	The PropertyList describing a single Publisher Component
Subscriber	The PropertyList describing a single Subscriber Component
Manager	The PropertyList describing a single Manager Component
Publication	The PropertyList describing a single Publication

Passive Objects

A *passive* Object is one that is relatively short lived, and typically none of its Properties change during its lifetime.

Passive Objects are a diverse lot, ranging from singular “blobs” of information to assorted lists.

The set of passive Objects includes:

Property	A named string structured as a name-value pair of strings
PropertyList	An unordered list of Properties
Name	A simple string used as a key in a specific context
NameList	An unordered list of Names
Message	The basic protocol data unit (PDU) used in SENSE communications
EventData	A Publication-specific collection of data associated with a specific event

Sessions

A fundamental SENSE concept is that of the Client *Session*. A Session is used to maintain a relationship between a specific Client and a Server. There are three types of Sessions managed by a Server, one for each of three types of Clients:

- Publisher
- Subscriber
- Manager

These three session types are quite similar in design and function; they exist as individually defined types only to allow the Server flexibility in assigning administrative policies with regard to capacity and required identification parameters.

Since the CommonSENSE protocol is based on datagrams—and since the standard datagram protocol in IP networks is the unreliable UDP transport service—a SENSE Server must be able to handle the likely scenario of a Client unexpectedly terminating its Session without first contacting the Server, thereby not allowing the Server to recapture its resources in a timely manner.

Registration

The way this scenario is handled in SENSE is that all Clients (except "Transient" Clients) must *register* with the Server as either a Subscriber, Publisher or Manager. Every Session registration involves a *registration period* that is "negotiated" between the Client and Server.

The negotiation of a registration period is quite simple and is designed to be "Server-centric" in that the Client "requests" a particular registration period in the registration request message, and the Server "declares" the actual registration period in the corresponding response message. This approach gives a Server a great deal of flexibility in managing its resources by being able to enforce short registration periods, if necessary, when it appears that too many Clients fail to properly deregister when they no longer have need for SENSE services.

Renewal

When the registration period for a Client is about to expire, the Client must continue the Session by sending the Server a *renewal* request. Similar to the registration request described above, the Client again proposes a registration period, and the Server responds with the value that is actually used.

Expiration Notices

When a Client Session is about to expire, the Server may optionally assist the Client in managing this situation by sending the Client an *expiration notice* to "nudge" the Client in sending a renewal request. This optional Server capability alleviates Client application developers from having to deal with potentially difficult timer facilities on the local platform.

Directory Services

A SENSE Server provides a modest level of Directory Services to its Clients to minimize the dependency on external facilities that may not be available in all platform or network environments.

The services needed by most Clients fall into two general categories of requests:

GetObjectIdList	Acquire a list of identifiers for a given Object class
GetObjectProperties	Acquire one or more Properties for the specified Object

Each of these requests can be made of one of the following Object types:

- Publication
- Publisher
- Subscriber
- Manager

A response to a GetObjectIdList request is a NameList containing the IDs of all currently registered Objects on the Server.

A response to a GetObjectProperties request is a PropertyList containing either all currently defined Properties, or only those Properties specified in the request. *(Note: should we dub this the "<u>Powerful GetObjectProperties</u>" message to better fit into the "Simple..." universe? ;--)*

A Server may elect to not support these services for particular Object types due to security concerns. For example, it may not be desirable within a given environment to allow arbitrary Clients to have access to identification information for Subscribers currently registered on the Server.

Events

Events form the backbone of existence for SENSE. The simple goal for SENSE in this respect is to allow the passage of opaque event data from a Publisher (on behalf of a registered Publication), through the Server, and on to all registered Subscribers. In a sense (pun intended), the protocol and services provided by a SENSE Server provide for a sort of "wrapper protocol" that allows an arbitrary Publisher to propagate any kind of data—even binary data—through a Server and on to any number of Subscribers (only limited by the resources of the Server).

Perhaps more importantly are the facilities provided by SENSE to maximize the likelihood of successful receipt of events by Subscribers. The delivery mechanism used by a Server includes multiple retries in sending any given Event Message to a target Subscriber; the Server will repeatedly send the Event Message until either the Client acknowledges receipt of the message, or the retry count is exceeded. (The maximum retry count is one of the properties "negotiated" between the Client and the Server when the Client registers or renews a Session with the Server.)

Event Protocols

One of the defining characteristics of a Publication is which Event Protocols are supported by the Publication. The term *Event Protocol* is used here to denote the set of Properties contained within an Event Message, as well as the content and format of an optional opaque data component attached to the message.

Note that using the term "Event Protocol" is really a misnomer here, since there is no real protocol exchange between the Publisher and its (anonymous) Subscribers. The term is used, however, as it is expected that most Publishers will use SENSE to encapsulate the contents of certain PDUs for distribution to interested Subscribers, since SENSE offers significantly improved event distribution services when compared with the native services associated with such PDUs.

SNMP is a prime example of this scenario, whereby TRAP messages are sent in such a manner that interested Client applications may *dynamically* commence receiving TRAPs from a number of SNMP agents, then disassociate themselves from reception, all without the SNMP agents being involved in the process, particularly in the process of defining and redefining destination TRAP addresses.

All that must be done to provide this kind of capability is to write a SENSE Publisher that communicates with a SNMP agent (as a type of "proxy" agent) and publishes a corresponding Publication on a SENSE Server. Alternatively, SENSE Publisher capabilities could be directly integrated within the SNMP agent.

One of the challenges of SENSE is to enumerate or otherwise register the set of names and/or identifiers that specify particular Event Protocols, such as SNMP.

Management Services

As of this writing, the topic of SENSE Management Services has not yet received much attention due to other, more pressing requirements to establish the basic foundation and infrastructure. However, it is expected that a modest level of services in this area will be developed to provide for these types of management activities:

Set Server Properties	Set one or more operational controls or identifying properties of a Server
Client Termination	Terminate a Client for any number of reasons, ranging from security to resource recapture (due to an errant Client application) to Component updating activities
Publication Termination	Similar to Client Termination, but targeted at Publications
Shutdown/Restart Server	Control the orderly, graceful shutdown of a Server so that all currently registered Clients may properly handle the interruption of services

Messaging Model

The Messaging Model used in SENSE is based on datagram technology. This choice was made in response to the requirement that SENSE be highly scaleable.

Clients communicate with Servers using a simple message format based on the Common Printer Access Protocol (CPAP) designed by Brian K. Reid and Christopher Kent while at Digital Equipment Corporation. The message format is very simple in that it is based on character strings and integers encoded as character strings. This approach largely eliminates the need to deal with the usual “Little Endian/Big Endian” problem of byte and bit order, making it an ideal candidate when multi-platform deployment and interoperability is critical.

Given the simple communications requirements of SENSE, it seems natural to base the general protocol used between SENSE Components on CPAP. This decision is so natural that the name of the variation of CPAP used in SENSE is called the “**CommonSENSE**” protocol.

An interesting aspect of CPAP of which the CommonSENSE protocol borrows heavily is the fact that, unlike many other protocols, there are only two messages used to convey a response to any given request; one message indicates success, and the other indicates failure.

The use of only two response messages is made possible by the fact that every message contains a message ID number that is unique to the *sender* of the message; when the receiver responds to the message, one of the two response messages types is returned in which the message ID of the response is the same as the original request message. It is up to the sender to maintain its own set of message ID values, which usually requires the management of a single “stationary pad-like” integer value that increases monotonically with each use.

The use of a message ID in this model allows the sender to manage what might be termed a “sub-session” with its communications peer. That is, if the sender wishes to perform rapid dispatching of message handling functions, then the message ID can be quickly and efficiently used to discriminate among received messages without having to manage potentially large relationships between message opcodes and corresponding message handler functions.

SENSE messages are roughly divided into six categories:

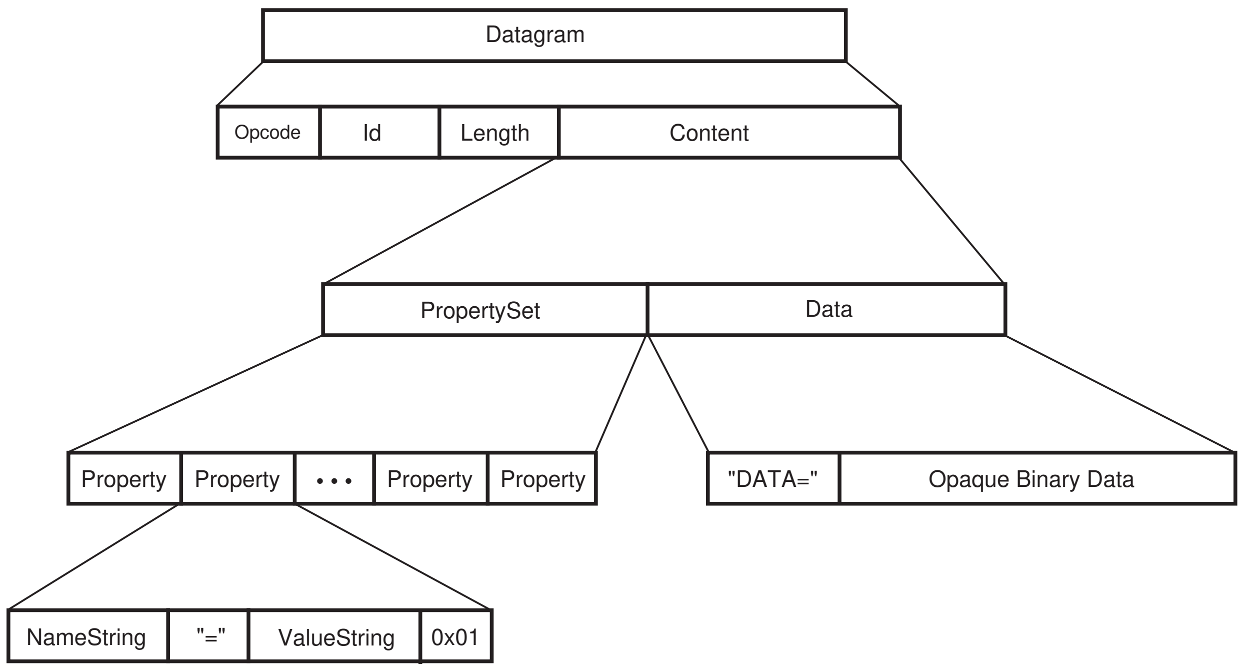
Queries	Requests from Clients to which the Server responds
Registration	Messages that pertain to the registration, renewal and deregistration of Client sessions on the Server
Events	Asynchronous messages pertaining to the delivery of events associated with Publications
Management	Messages that control the operations and identification of the Server
Responses	Messages sent in response to queries, or to acknowledge receipt of certain types of messages
Maintenance	Miscellaneous messages used to test or assist in communications connectivity

The currently defined set of messages is described in the following table.

Message Name	Description
Event Messages	
PublishEvent	Submit an Event to the Server
Event	Receive an Event from a Server
Registration Messages	
Cancel	Cancel a subscription to one or more Publications
RegisterManager	Start a Management Session
RegisterPublication	Start a new Publication
RegisterPublisher	Start a Publisher Session
RegisterSubscriber	Start a Subscriber Session
RenewManager	Continue a Management Session
RenewPublisher	Continue a Publisher Session
RenewSubscriber	Continue a Subscriber Session
Subscribe	Register for receipt of Events from one or more Publications
UnRegisterManager	Terminate a Manager Session
UnRegisterPublication	Terminate a Publication
UnRegisterPublisher	Terminate a Publisher Session
UnRegisterSubscriber	Terminate a Subscriber Session
UpdateProperties	Update the Properties of an Object
Query Messages	
GetManagerIdList	Obtain the list of IDs for all registered Managers
GetManagerProperties	Obtain one, multiple or all Properties of a specified Manager
GetPublicationIdList	Obtain the list of IDs for all registered Publications
GetPublicationProperties	Obtain one, multiple or all Properties of a specified Publication
GetPublisherIdList	Obtain the list of IDs for all registered Publishers
GetPublisherProperties	Obtain one, multiple or all Properties of a specified Publisher
GetSubscriberIdList	Obtain the list of IDs for all registered Subscribers
GetSubscriberProperties	Obtain one, multiple or all Properties of a specified Subscriber
Management Messages	
SetServerProperties	Set one or more operational or identifying Properties of a Server
ShutdownServer	Shutdown a Server
TerminateManager	Terminate one or more Manager Sessions
TerminatePublication	Terminate one or more Publications
TerminatePublisher	Terminate one or more Publisher Sessions
TerminateSubscriber	Terminate one or more Subscriber Sessions
Maintenance Messages	
KeepAlive	Indicate that the specified Session remains intact
Loopback	Determine basic connectivity
Response Messages	
Reply	Positive acknowledgment or returned results from an associated message
Nak	Negative response

Message Structure

The structure of a SENSE message is shown in the following illustration:



Glossary of Terms

This section contains concise definitions of SENSE terminology.

It is recognized that the following list is neither complete nor entirely consistent...

Alert Condition Condition of a Publication in which the Publication is unable to perform its intended mission until the cause of the alert condition is eliminated.

Alert Message An Event Message declaring that the Publication has entered the Alert Condition, or that a new Alert Event has occurred while the Publication is in the Alert Condition.

API Application Programming Interface; commonly used to reference either the specification or set of library modules used by an application to communicate with a Server.

Basic Edition The standard, fundamental Edition for a Publication; a Publication must have this Edition registered.

Busy Condition Condition of a Publication in which the Publication is currently performing useful work in its intended mission.

Cancel Terminate subscription to the specified Publications.

Capability A specific feature performed by a Server, Publisher or Subscriber; a capability is indicated by the presence of a specific property, the value of which specifies the level of capability. For example, if a Server is able to persistently store Publications, then it would define the property "`PersistentPublications`" and set the value to "`Yes`".

Client An application consuming the services of a Server.

Client Context An API element used within a Client application to manage a single session with a Server.

Condition The overall operational status of a Publication, one of a small, finite set of enumerated values.

Client Context An API element used within a Client application to manage a single session with a Server.

Edition A specific form of a Publication; an Edition typically describes the binding between a Publisher and an Event Protocol, but may also describe specific subsets of Event information for the Publication represented by the Publisher. An Edition may be comprised of any combination of a number of dimensional aspects of the Publication.

Entity The abstract source of events represented by a Publication.

Event A collection of information submitted by a Publisher on behalf of an associated Publication. A Subscriber receives Events after it has subscribed to the Publication.

Event Message A message used to convey the occurrence and content of an Event. A Publisher sends Event Messages to a Server on behalf of the associated Publication, whereupon the Server sends the Event Message (with enhancements) to all current Subscribers of the Publication.

Event Protocol The format and content of a series of messages that convey event information between two parties; examples might include SNMP, IEEE 1284.1 (TIPSI), CPAP, etc.

Healthy Condition A condition indicating that the Publication is operationally consistent and able to perform its intended mission.

Idle Condition A condition in which the Publication is operationally consistent, but currently not performing any useful work in its intended mission.

Info Message A message containing arbitrary informative data, such as debugging information, etc.

Message A protocol element exchanged between a Client and a Server.

Message Id	An unsigned 32-bit integer used by the message sender to uniquely identify the message.
Nak Message	A message indicating that the previously sent message can not be processed as desired; the reason for failure is contained in the message.
Name	An string used to denote a specific Value.
NameList	An unordered list of Names.
Negotiation	The mechanism used to resolve registration properties when a Server and a Client may have different desires or policies; a Client submits the desired value, and the Server responds with the value that must be used during the session.
Operator	A person designated to perform certain types of operational tasks for an Entity.
Persistent	Denotes a state that will not change unless further action is taken.
Property	An object attribute represented by a Name/Value pair.
PropertyList	An unordered list of Properties.
Publication	The representation of an Entity as managed by a Publisher.
Publication Id	The unique Server-assigned identifier for a single Publication.
Publisher	The agent acting on behalf of an Entity to publish Events that occur within the Entity.
Publisher Id	The unique Server-assigned identifier for a specific Publisher.
Publisher API	The API used by a Publisher to communicate with a Server.
Registration Period	The time during which a Client Session is active; once the period expires, the Session is no longer exists and all communications with the Client ceases.

Renewal The mechanism used by a Client to extend the life of a Session.

Reply Message A message used to respond to a previously received request. All Reply messages have as a Message Id the same Id specified in the original request message.

Server The agent providing SENSE services to Clients.

Session The time during which a Client is registered with a Server; a Session commences upon successful registration with the Server, and ends when the Client either deregisters with the Server, or the registration period expires.

State Message A message describing the fact that a Publication has entered a particular state.

Subscribe The mechanism used by a Subscriber to begin receiving Event Messages from a Publication.

Subscriber A Client interested in Events from a Publication.

Subscriber API The API used by a Subscriber to communicate with a Server.

Subscription The mechanism used to manage the relationship between one or more Publications and a single Subscriber.

Subscription Id The unique Server-assigned identifier for a single Subscription.

Technician A person assigned certain operational tasks that require a relatively high level of training.

Transient A situation that is very short-lived, one that does not require further action to change.

Unknown Condition A condition indicating that the current status of an Publication is not currently known.

Value A string representing context-specific information.

Warning Condition A condition indicating that the Publication currently has one or more state situations which, if not corrected in the relatively near future, will probably cause the Publication to enter the Alert Condition.