IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
TYLER DIVISION

EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. APPLE INC., Defendant.	Case No. 6:12-cv-00783-MHS-CMC JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. SPRINT NEXTEL CORPORATION; SPRINT COMMUNICATIONS COMPANY L.P.; SPRINT SPECTRUM, L.P.; and SPRINT SOLUTIONS, INC., Defendants.	Case No. 6:12-cv-00791-MHS-CMC JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. FACEBOOK, INC., Defendant.	Case No. 6:12-cv-00784-MHS-CMC JURY TRIAL DEMANDED

EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. FOURSQUARE LABS, INC., Defendant.	Case No. 6:12-cv-00785-LED JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. GROUPON, INC., Defendant.	Case No. 6:12-cv-00787-MHS-CMC JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. LIVINGSOCIAL, INC., Defendant.	Case No. 6:12-cv-00789-MHS-CMC JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. MILLENNIAL MEDIA, INC., Defendant.	Case No. 6:12-cv-00790-MHS-CMC JURY TRIAL DEMANDED

EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. TWITTER, INC., Defendant.	Case No. 6:12-cv-00792-MHS-CMC JURY TRIAL DEMANDED
EVOLUTIONARY INTELLIGENCE, LLC, Plaintiff, v. YELP, INC., Defendant.	Case No. 6:12-cv-00794-MHS-CMC JURY TRIAL DEMANDED

Evolutionary Intelligence's P.R. 3-1 Disclosure of Asserted Claims and Infringement Contentions, and P.R. 3-2 Disclosures

Pursuant to Patent Rule 3-1, Plaintiff Evolutionary Intelligence, LLC ("Evolutionary Intelligence") provides its Disclosure of Asserted Claims and Preliminary Infringement Contentions. Evolutionary Intelligence's statements are based on publicly available materials regarding the accused infringing products and services of Defendants. Evolutionary Intelligence has not yet had access to any discovery of Defendants' materials, and therefore certain information is not yet available to Evolutionary Intelligence that is relevant to its infringement claims. Evolutionary Intelligence reserves the right to supplement and/or modify its disclosures herein based on additional information obtained through formal discovery or other means concerning the Defendants' products or services. Further, the patents' claims have not

yet been construed, and Evolutionary Intelligence's investigation of claim construction issues is continuing. Accordingly, Evolutionary Intelligence reserves its right to supplement and/or modify its disclosures pursuant to P.R. 3-6.

I. Evolutionary Intelligence's P.R. 3-1(a) Disclosures

A. Apple Inc. ("Apple")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Apple directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1–16; and U.S. Patent No. 7,702,682, claims 1–11 and 14–23. *See* Exhibits A and B.

B. Facebook, Inc. ("Facebook")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Facebook directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1–16; U.S. Patent No. 7,702,682, claims 1–11 and 14–23. *See* Exhibits C and D.

C. Twitter, Inc. ("Twitter")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Twitter directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-16; U.S. Patent No. 7,702,682, claims 1-11, 14-23. *See* Exhibits E and F.

D. Millennial Media, Inc. ("Millennial")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Millennial directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1–16; U.S. Patent No. 7,702,682, claims 1, 3–7, 10, 11, and 14–22. *See* Exhibits G and H.

E. Sprint Nextel Corporation, Spring Communications Company, L.P., Sprint Spectrum, L.P., and Sprint Solutions, Inc. (collectively, "Sprint")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Sprint directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-12, and 14-16; U.S. Patent No. 7,702,682, claims 1-11, and 14-23. *See* Exhibits I and J.

F. Groupon, Inc. ("Groupon")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Groupon directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-5, 7-16; U.S. Patent No. 7,702,682, claims 1, 3-7, 10-11, and 14-23. *See* Exhibits K and L.

G. Livingsocial, Inc. ("Livingsocial")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Livingsocial directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-4, 7-9, 11-16; U.S. Patent No. 7,702,682, claims 1, 3-7, 10-11, 14-16, 19, and 21. *See* Exhibits M and N.

H. Foursquare Labs, Inc. ("Foursquare")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Foursquare directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-16; U.S. Patent No. 7,702,682, claims 1, 3-7, 10-11, and 14-23. *See* Exhibits O and P.

I. Yelp, Inc. ("Yelp")

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(a), that Yelp directly, contributorily and/or by inducement infringes the following claims: U.S. Patent No. 7,010,536, claims 1-9, 11-

16; U.S. Patent No. 7,702,682, claims 1, 3–7, 10–11, 15–16, 19, and 21. *See* Exhibits Q and R.

II. Evolutionary Intelligence's P.R. 3-1(b) Disclosures

Subject to ongoing discovery and investigation, Evolutionary Intelligence hereby contends, pursuant to P.R. 3-1(b), that the asserted patent claims are infringed by the Defendants' Accused Instrumentalities as shown in the infringement charts attached as Exhibits A through R.

III. Evolutionary Intelligence's P.R. 3-1(c) Disclosures

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(c), that each element of each infringed claim is found within each Accused Instrumentality as shown in the infringement charts attached as Exhibits A through R.

IV. Evolutionary Intelligence's P.R. 3-1(d) Disclosures

Subject to ongoing discovery and investigation, Evolutionary Intelligence hereby contends, pursuant to P.R. 3-1(d), that each element of each asserted claim is literally present in each Accused Instrumentality as shown in Exhibits A through R.

At this time, Evolutionary Intelligence knows of no specific limitation of the asserted claims where infringement depends on the doctrine of equivalents. However, as indicated above, further discovery is required regarding literal infringement by the Defendants' Accused Instrumentalities, and the claims of the patents-in-suit have yet to be construed. Evolutionary Intelligence expressly reserves the right to augment and supplement its position on whether there is infringement under the doctrine of equivalents of any elements of any asserted claims after discovery from the Defendants and/or after the Court has construed the asserted claims.

V. Evolutionary Intelligence's P.R. 3-1(e) Disclosures

Pursuant to P.R. 3-1(e), Evolutionary Intelligence states that the '536 patent claims priority to Provisional Application No. 60/073,209, filed on January 30, 1998, and that the '682 patent is a continuation of U.S. Application No. 09/284,113, filed on April 7, 1999, and claims priority to Provisional Application No. 60/073,209, filed on January 30, 1998 and International Application No. PCT/US99/01988, filed on January 28, 1999.

VI. Evolutionary Intelligence's P.R. 3-1(f) Disclosures

Evolutionary Intelligence hereby discloses, pursuant to P.R. 3-1(f), that Evolutionary Intelligence does not have any apparatus, product, device, process, method, act, or other instrumentality that practices the claimed invention.

Evolutionary Intelligence reserves its right to argue that the claimed invention is practiced by any apparatus, product, device, process, method, act, or other instrumentality owned by any third party.

VII. Evolutionary Intelligence's P.R. 3-2 Disclosures

In accordance with P.R. 3-2, Evolutionary Intelligence makes the following disclosures:

Pursuant to P.R. 3-2(a), Evolutionary Intelligence identifies the following documents: EV0004059–4192.

Pursuant to P.R. 3-2(b), Evolutionary Intelligence identifies the following documents that were created on or before the date of application for the patents-in-suit and that evidence the conception, reduction to practice, design, and development of the claimed inventions: EV0003913–59.

There may be other documents relevant to conception, reduction to practice, design, development, and/or disclosure of the claimed invention pursuant to P.R. 3-2

that are protected by the attorney-client privilege and/or the attorney work product doctrine. A privilege log identifying those documents will be provided to the Defendants at the appropriate time.

Pursuant to P.R. 3-2(c), the following documents are copies of the file histories for the patents-in-suit: EV0000097–645, and EV0000646–1287.

Dated: May 22, 2013

Respectfully submitted,

/s/ Todd Kennedy /
Adam J. Gutride, Esq.
Seth A. Safier, Esq.
Todd Kennedy, Esq.
Anthony J. Patek, Esq.
Marie McCrary, Esq.
835 Douglass Street
San Francisco, California 94114
Telephone: (415) 789-6390
Facsimile: (415) 449-6469
adam@gutridesafier.com
seth@gutridesafier.com
todd@gutridesafier.com
anthony@gutridesafier.com
marie@gutridesafier.com

Charles Ainsworth
Parker Bunt & Ainsworth
100 E. Ferguson, Suite 1114
Tyler, Texas 75702
Telephone: (903) 531-3535
Facsimile: (903) 533-9687
charley@pbatyler.com

Attorneys for Plaintiff Evolutionary Intelligence LLC

CERTIFICATE OF SERVICE

The undersigned certifies that, on **May 22, 2013**, the foregoing document was served via email on all counsel of record who have consented to electronic service.

Dated: May 22, 2013

/s/ Todd Kennedy /
Todd Kennedy
Gutride Safier LLP
835 Douglass Street
San Francisco, California 94114
Telephone: (415) 789-6390
Facsimile: (415) 449-6469
todd@gutridesafier.com

Exhibit A

Infringement of U.S. Patent No. 7,010,536 by Apple Inc.

Subject to ongoing discovery and investigation, Evolutionary Intelligence identifies the following accused instrumentalities of Apple Inc.:

1. All Apple devices compatible with the iOS mobile operating system (collectively, the "iOS-Compatible Devices"). These devices include at least the iPhone, iPad, iPod Touch, and Apple TV. The accused instrumentalities shall also encompass the software applications designed to run on these Apple devices.

Evolutionary Intelligence's identification of these accused instrumentalities is based on publicly available information. Evolutionary Intelligence has not yet had any discovery from Apple. Evolutionary Intelligence reserves its right to seek discovery regarding other products and services offered by Apple, and to identify additional accused instrumentalities based on that discovery and further investigation.

The discussion in the claim chart below with respect to some of the claim elements occasionally refers to and/or incorporates by reference discussion and evidence pertaining to other claim elements. Such incorporation by reference to shall not be read to imply that any two claim elements have the same claim scope. Rather, such incorporation by reference is used when two or more claim elements are sufficiently similar that repeating the relevant discussion and evidence would be unnecessarily cumbersome and redundant.

Subject to ongoing discovery and investigation, Evolutionary Intelligence contends, pursuant to P.R. 3-1(c), that each element of each infringed claim is found within each accused instrumentality as shown below:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
1A	1. An apparatus for transmitting, receiving and manipulating information on a computer system, the apparatus including	Each iOS-Compatible device is an apparatus for transmitting, receiving and manipulating information on a computer system. (*See* the claim elements below.) Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1B	a plurality of containers, each container being a logically defined data enclosure and comprising:	Each iOS-Compatible Device includes a plurality of containers, each container being a logically defined data enclosure. Examples of containers are "Event," "Reminder," and "Alarm" containers: **Creating and Editing Events** **Note:** If you're developing on iOS, you have the option of letting users modify event data with the event view controllers provided in the Event Kit UI framework. For information on how to use these event view controllers, see "Providing Interfaces for Events." Create a new event with the `eventWithEventStore:` method of the `EKEvent` class. You can edit the details of a new event or an event you previously fetched from the Calendar database by setting the event's corresponding properties. Some of the details you can edit include: • The event's title with the `title` property • The event's start and end dates with the `startDate` and `endDate` properties • The calendar with which the event is associated with the `calendar` property • The alarms associated with the event with the `alarms` property (see "Configuring Alarms" for more details) • The event's recurrence rule, if it is a repeating event, with the `recurrenceRules` property (see "Creating Recurring Events" for more details)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		(EV0001412.) ## Creating and Editing Reminders You can create reminders using the `reminderWithEventStore:` class method. The `title` and `calendar` properties are required. The calendar for a reminder is the list with which it is grouped. Like events, reminders can trigger time-based or location-based alarms to alert the user of a certain task. Read "Configuring Alarms" for more information on how to attach alarms to calendar items. To associate a start date or due date with a reminder, use the `startDateComponents` and `dueDateComponents` properties. To complete a reminder, set the `completed` property to `YES`, which automatically sets `completionDate` to the current date. (EV0001416.) ### alarmWithAbsoluteDate: Creates and returns an alarm with an absolute date. `+ (EKAlarm *)alarmWithAbsoluteDate:(NSDate *)`*date* **Parameters** *date* The date for the alarm. **Return Value** The created alarm. **Availability** Available in iOS 4.0 and later. **Declared In** `EKAlarm.h` (EV0001420.) Additional examples of containers are location containers such as the "CLLocation," "CLLocationCoordinate2D," "CLLocationManager," "CLPlacemark," and "CLRegion" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Overview A `CLLocation` object represents the location data generated by a `CLLocationManager` object. This object incorporates the geographical coordinates and altitude of the device's location along with values indicating the accuracy of the measurements and when those measurements were made. In iOS, this class also reports information about the speed and heading in which the device is moving. Typically, you use a `CLLocationManager` object to create instances of this class based on the last known location of the user's device. You can create instances yourself, however, if you want to cache custom location data or get the distance between two points. (EV0001512.) **CLLocationCoordinate2D** *A structure that contains a geographical coordinate using the WGS 84 reference frame.* `typedef struct { CLLocationDegrees latitude; CLLocationDegrees longitude; } CLLocationCoordinate2D;` (EV0001527.) **Overview** The `CLLocationManager` class defines the interface for configuring the delivery of location- and heading-related events to your application. You use an instance of this class to establish the parameters that determine when location and heading events should be delivered and to start and stop the actual delivery of those events. You can also use a location manager object to retrieve the most recent location and heading data. A location manager object provides support for the following location-related activities: • Tracking large or small changes in the user's current location with a configurable degree of accuracy. • Reporting heading changes from the onboard compass. (iOS only) • Monitoring distinct regions of interest and generating location events when the user enters or leaves those regions. • Deferring the delivery of location updates while the app is in the background. (iOS 6 and later only)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		(EV0001625.) Overview A `CLPlacemark` object stores placemark data for a given latitude and longitude. Placemark data includes information such as the country, state, city, and street address associated with the specified coordinate. It can also include points of interest and geographically related data. Placemark objects are typically generated by a `CLGeocoder` object, although you can also create them explicitly yourself. (EV0001661.) Overview The `CLRegion` class defines a geographical area that can be tracked. When an instance of this class is registered with a `CLLocationManager` object, the location manager generates an appropriate event whenever the user crosses the boundaries of the defined area. To use this class, create an instance of it and use the `startMonitoringForRegion:desiredAccuracy:` method of a `CLLocationManager` object to begin monitoring it. (EV0001686.) Publicly available information indicates that containers are used throughout the iOS operating system of the iOS-Compatible Devices. The containers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the containers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1C	an information	Each container of each iOS-Compatible Device comprises an information element having information.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

	element having information;	For example, information elements with information regarding events are associated with at least the following properties of "Event" containers:
		
		Properties
		
		allDay
		
		A Boolean value that indicates whether the event is an all-day event.
		
		`@property(nonatomic, getter=isAllDay) BOOL allDay`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		availability
		
		The availability setting for the event.
		
		`@property(nonatomic) EKEventAvailability availability`
		
		Discussion
		This setting is used by CalDAV and Exchange servers to indicate how the event should be treated for scheduling purposes.
		
		If the event's calendar does not support availability settings, this property's value is `EKEventAvailabilityNotSupported`.
		
		Availability
		Available in iOS 4.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		See Also
		`EKEventAvailability`
		Declared In
		`EKEvent.h`
		
		## birthdayPersonID
		
		The Address Book framework record identifier of the person for this birthday event. (read-only)
		
		`@property(nonatomic, readonly) NSInteger birthdayPersonID`
		
		Discussion
		This property is only set if this is a birthday event; otherwise the property is `nil`.
		
		Special Considerations
		
		Note: This property is equivalent to the `birthdayPersonUniqueID` property on OS X.
		
		Availability
		Available in iOS 5.0 and later.
		Declared In
		`EKEvent.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		endDate
		
		The end date for the event.
		
		`@property(nonatomic, copy) NSDate *endDate`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		eventIdentifier
		
		A unique identifier for the event. (read-only)
		
		`@property(nonatomic, readonly) NSString *eventIdentifier`
		
		Discussion
		You can use this identifier to look up an event with the EKEventStore method `eventWithIdentifier:`.
		
		If the calendar of an event changes, its identifier most likely changes as well.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		isDetached
		
		A Boolean value that indicates whether an event is a detached instance of a repeating event. (read-only)
		
		`@property(nonatomic, readonly) BOOL isDetached`
		
		Discussion
		This value is `YES` if and only if the event is part of a repeating event and one or more of its attributes have been modified from the repeating event's default attributes.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		organizer
		
		The organizer associated with the event. (read-only)
		
		`@property(nonatomic, readonly) EKParticipant *organizer`
		
		Discussion
		This property is `nil` if the event has no organizer.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDate
		
		The start date of the event.
		
		`@property(nonatomic, copy) NSDate *startDate`
		
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		status
		
		The status of the event. (read-only)
		
		`@property(nonatomic, readonly) EKEventStatus status`
		
		Discussion
		You should act based on an event's status only if the status is EKEventStatusCanceled, which indicates that the event has been canceled. Other statuses should be considered informational.
		
		Availability
		Available in iOS 4.0 and later.
		See Also
		EKEventStatus
		Declared In
		EKEvent.h
		
		(EV0001424-26.)
		
		In another example, information elements with information regarding reminders are associated with at least the following properties of "Reminder" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		completed
		
		A Boolean value determining whether or not the reminder is marked completed.
		
		`@property(nonatomic, getter=isCompleted) BOOL completed`
		
		Discussion
		Setting this property to `YES` will set `completionDate` to the current date; setting this property to `NO` will set `completionDate` to `nil`.
		
		Special Considerations
		If the reminder was completed using a different client, you may encounter the case where this property is `YES`, but `completionDate` is `nil`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Declared In
		`EKReminder.h`
		
		## dueDateComponents
		
		The date by which the reminder should be completed.
		
		`@property(nonatomic, copy) NSDateComponents *dueDateComponents`
		
		Discussion
		The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone.
		
		Special Considerations
		On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		(EV0001431-33.)
		
		In another example, information elements with information regarding alarms are associated with at least the following properties of "Alarm" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Properties
		
		absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKAlarm.h`
		
		proximity
		
		A value indicating how a location-based alarm is triggered.
		
		`@property EKAlarmProximity *proximity`
		
		Discussion
		Alarms can be set to trigger when entering or exiting a location specified by `structuredLocation`. By default, alarms are not affected by location.
		
		Availability
		Available in iOS 6.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		See Also EKAlarmProximity **Declared In** EKAlarm.h relativeOffset The offset from the start of an event, at which the alarm fires. @property NSTimeInterval relativeOffset **Discussion** If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm. **Availability** Available in iOS 4.0 and later. **Declared In** EKAlarm.h structuredLocation The location to trigger an alarm. @property EKStructuredLocation *structuredLocation **Discussion** This property is used in conjunction with proximity to perform geofence-based triggering of reminders. **Availability** Available in iOS 6.0 and later. **Declared In** EKAlarm.h (EV0001418-20.) In another example, information elements with information regarding location are associated with at least the following properties of the "CLLocation" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		`coordinate` (page 5) *property* The geographical coordinate information. (read-only) `altitude` (page 5) *property* The altitude measured in meters. (read-only) `horizontalAccuracy` (page 6) *property* The radius of uncertainty for the location, measured in meters. (read-only) `verticalAccuracy` (page 8) *property* The accuracy of the altitude value in meters. (read-only) `timestamp` (page 7) *property* The time at which this location was determined. (read-only) – `description` (page 9) Returns the location data in a formatted text string. (EV0001513.) In another example, information elements with information regarding location are associated with at least the following properties of the "CLLocationManager" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		activityType
		
		The type of user activity associated with the location updates.
		
		`@property(assign, nonatomic) CLActivityType activityType`
		
		Discussion
		The location manager uses the information in this property as a cue to determine when location updates may be automatically paused. Pausing updates gives the system the opportunity to save power in situations where the user's location is not likely to be changing. For example, if the activity type is `CLActivityTypeAutomotiveNavigation` (page 29) and no location changes have occurred recently, the radios might be powered down until movement is detected again.
		
		Availability
		Available in iOS 6.0 and later.
		
		Declared in
		`CLLocationManager.h`
		
		delegate
		
		The delegate object to receive update events.
		
		`@property(assign, nonatomic) id<CLLocationManagerDelegate> delegate`
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		Teslameter
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		desiredAccuracy
		
		The accuracy of the location data.
		
		`@property(assign, nonatomic) CLLocationAccuracy desiredAccuracy`
		
		Discussion
		The receiver does its best to achieve the requested accuracy; however, the actual accuracy is not guaranteed.
		
		You should assign a value to this property that is appropriate for your usage scenario. In other words, if you need the current location only within a few kilometers, you should not specify `kCLLocationAccuracyBest` for the accuracy. Determining a location with greater accuracy requires more time and more power.
		
		When requesting high-accuracy location data, the initial event delivered by the location service may not have the accuracy you requested. The location service delivers the initial event as quickly as possible. It then continues to determine the location with the accuracy you requested and delivers additional events, as necessary, when that data is available.
		
		The default value of this property is `kCLLocationAccuracyBest`.
		
		This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		
		Declared in
		`CLLocationManager.h`
		
		distanceFilter
		
		The minimum distance (measured in meters) a device must move horizontally before an update event is generated.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		`@property(assign, nonatomic) CLLocationDistance distanceFilter`
		
		Discussion
		This distance is measured relative to the previously delivered location. Use the value `kCLDistanceFilterNone` to be notified of all movements. The default value of this property is `kCLDistanceFilterNone`.
		
		This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		heading
		
		The most recently reported heading. (read-only)
		
		`@property(readonly, nonatomic) CLHeading *heading`
		
		Discussion
		The value of this property is `nil` if heading updates have never been initiated.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`
		
		headingFilter
		
		The minimum angular change (measured in degrees) required to generate new heading events.
		`@property(assign, nonatomic) CLLocationDegrees headingFilter`
		
		Discussion
		The angular distance is measured relative to the last delivered heading event. Use the value `kCLHeadingFilterNone` to be notified of all movements. The default value of this property is `1` degree.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		headingOrientation
		
		The device orientation to use when computing heading values.
		
		`@property(assign, nonatomic) CLDeviceOrientation headingOrientation`
		
		Discussion
		When computing heading values, the location manager assumes that the top of the device in portrait mode represents due north (0 degrees) by default. For applications that run in other orientations, this may not always be the most convenient orientation. This property allows you to specify which device orientation you want the location manager to use as the reference point for due north.
		
		Although you can set the value of this property to `CLDeviceOrientationUnknown`, `CLDeviceOrientationFaceUp`, or `CLDeviceOrientationFaceDown`, doing so has no effect on the orientation reference point. The original reference point is retained instead.
		
		Changing the value in this property affects only those heading values reported after the change is made.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`
		
		location
		
		The most recently retrieved user location. (read-only)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		`@property(readonly, nonatomic) CLLocation *location`
		
		Discussion
		The value of this property is `nil` if no location data has ever been retrieved.
		
		In iOS 4.0 and later, this property may contain a more recent location object at launch time. Specifically, if significant location updates are running and your application is terminated, this property is updated with the most recent location data when your application is relaunched (and you create a new location manager object). This location data may be more recent than the last location event processed by your application.
		
		It is always a good idea to check the timestamp of the location stored in this property. If the receiver is currently gathering location data, but the minimum distance filter is large, the returned location might be relatively old. If it is, you can stop the receiver and start it again to force an update.
		
		Availability
		Available in iOS 2.0 and later.
		
		See Also
		– `startUpdatingLocation` (page 25)
		
		Related Sample Code
		AVMovieExporter
		Locations
		PhotoLocations
		TaggedLocations
		
		Declared in
		`CLLocationManager.h`
		maximumRegionMonitoringDistance
		
		The largest boundary distance that can be assigned to a region. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDistance maximumRegionMonitoringDistance`
		
		Discussion
		This property defines the largest boundary distance allowed from a region's center point. Attempting to monitor a region with a distance larger than this value causes the location manager to send a `kCLErrorRegionMonitoringFailure` error to the delegate.
		
		If region monitoring is unavailable or not supported, the value in this property is `-1`.
		
		Availability
		Available in iOS 4.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Declared in
		`CLLocationManager.h`
		
		monitoredRegions
		
		The set of shared regions monitored by all location manager objects. (read-only)
		
		`@property(readonly, nonatomic) NSSet *monitoredRegions`
		
		Discussion
		You cannot add regions to this property directly. Instead, you must register regions by calling the `startMonitoringForRegion:desiredAccuracy:` (page 34) method. The regions in this property are shared by all instances of the `CLLocationManager` class in your application.
		
		The objects in this set may not necessarily be the same objects you specified at registration time. Only the region data itself is maintained by the system. Therefore, the only way to uniquely identify a registered region is using its `identifier` property.
		
		The location manager persists region data between launches of your application. If your application is terminated and then relaunched, the contents of this property are repopulated with region objects that contain the previously registered data.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		pausesLocationUpdatesAutomatically
		
		A Boolean value indicating whether the location manager object may pause location updates.
		
		`@property(assign, nonatomic) BOOL pausesLocationUpdatesAutomatically`
		
		Discussion
		Allowing the location manager to pause updates can improve battery life on the target device without sacrificing location data. When this property is set to `YES`, the location manager pauses updates (and powers down the appropriate hardware) at times when the location data is unlikely to change. For example, if the user stops for food while using a navigation app, the location manager might pause updates for a period of time. You can help the determination of when to pause location updates by assigning a value to the `activityType` property.
		
		The default value of this property is `YES`.
		
		Availability
		Available in iOS 6.0 and later.
		
		See Also
		`@property activityType` (page 11)
		
		Declared in
		`CLLocationManager.h`
		
		(EV0001632-38.)
		
		In another example, information elements with information regarding location are associated with at least the following properties of the "CLPlacemark" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		`name` (page 8) *property* The name of the placemark. (read-only) `addressDictionary` (page 5) *property* A dictionary containing the Address Book keys and values for the placemark. (read-only) `ISOcountryCode` (page 7) *property* The abbreviated country name. (read-only) `country` (page 6) *property* The name of the country associated with the placemark. (read-only) `postalCode` (page 9) *property* The postal code associated with the placemark. (read-only) `administrativeArea` (page 6) *property* The state or province associated with the placemark. (read-only) `subAdministrativeArea` (page 10) *property* Additional administrative area information for the placemark. (read-only) `locality` (page 7) *property* The city associated with the placemark. (read-only) `subLocality` (page 10) *property* Additional city-level information for the placemark. (read-only) `thoroughfare` (page 11) *property* The street address associated with the placemark. (read-only) `subThoroughfare` (page 10) *property* Additional street-level information for the placemark. (read-only) `region` (page 9) *property* The geographic region associated with the placemark. (read-only) `inlandWater` (page 7) *property* The name of the inland water body associated with the placemark. (read-only) `ocean` (page 9) *property* The name of the ocean associated with the placemark. (read-only) `areasOfInterest` (page 6) *property* The relevant areas of interest associated with the placemark. (read-only) (EV0001662-63.) In another example, information elements with information regarding location are associated with at least the following properties of the "CLRegion" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		center
		The center point of the region. (read-only)
		`@property(readonly, nonatomic) CLLocationCoordinate2D center`
		Availability
		Available in iOS 4.0 and later.
		Related Sample Code
		Regions
		Declared in
		`CLRegion.h`
		identifier
		The identifier for the region object. (read-only)
		`@property(readonly, nonatomic) NSString *identifier`
		Discussion
		This is a value that you specify and can use to identify this region inside your application.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLRegion.h`
		
		radius
		The radius (measured in meters) that defines the region's outer boundary. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDistance radius`
		
		Availability
		Available in iOS 4.0 and later.
		
		Related Sample Code
		Regions
		
		Declared in
		`CLRegion.h`
		
		(EV0001688.)
		
		In another example, information elements with information regarding location are associated with at least the following properties of the "CLHeading" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		headingAccuracy
		
		The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDirection headingAccuracy`
		
		Discussion
		A positive value in this property represents the potential error between the value reported by the `magneticHeading` (page 5) property and the actual direction of magnetic north. Thus, the lower the value of this property, the more accurate the heading. A negative value means that the reported heading is invalid, which can occur when the device is uncalibrated or there is strong interference from local magnetic fields.
		
		Availability
		Available in iOS 3.0 and later.
		
		Declared in
		`CLHeading.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		magneticHeading *The heading (measured in degrees) relative to magnetic north. (read-only)* `@property(readonly, nonatomic) CLLocationDirection magneticHeading` **Discussion** The value in this property represents the heading relative to the magnetic North Pole, which is different from the geographic North Pole. The value `0` means the device is pointed toward magnetic north, `90` means it is pointed east, `180` means it is pointed south, and so on. The value in this property should always be valid. In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` property in *CLLocationManager Class Reference*. If the `headingAccuracy` property contains a negative value, the value in this property should be considered unreliable. **Availability** Available in iOS 3.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		See Also
		`@property headingAccuracy` (page 5)
		`@property trueHeading` (page 6)
		
		Declared in
		`CLHeading.h`
		
		timestamp
		
		The time at which this heading was determined. (read-only)
		
		`@property(readonly, nonatomic) NSDate *timestamp`
		
		Availability
		Available in iOS 3.0 and later.
		
		Declared in
		`CLHeading.h`
		
		trueHeading
		
		The heading (measured in degrees) relative to true north. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDirection trueHeading`
		
		Discussion
		The value in this property represents the heading relative to the geographic North Pole. The value `0` means the device is pointed toward true north, `90` means it is pointed due east, `180` means it is pointed due south, and so on. A negative value indicates that the heading could not be determined.
		
		In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` property in *CLLocationManager Class Reference*.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Important: This property contains a valid value only if location updates are also enabled for the corresponding location manager object. Because the position of true north is different from the position of magnetic north on the Earth's surface, Core Location needs the current location of the device to compute the value of this property.
		
		Availability
		Available in iOS 3.0 and later.
		
		See Also
		`@property magneticHeading` (page 5)
		
		Declared in
		`CLHeading.h`
		x
		
		The geomagnetic data (measured in microteslas) for the x-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue x`
		
		Discussion
		This value represents the x-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		y
		
		The geomagnetic data (measured in microteslas) for the y-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue y`
		
		Discussion
		This value represents the y-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		
		z
		
		The geomagnetic data (measured in microteslas) for the z-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue z`
		
		Discussion
		This value represents the z-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		
		Publicly available information indicates that information elements are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the information elements are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1D	a plurality of registers, the plurality of registers forming part of the container and including	Each container of each iOS-Compatible Device comprises a plurality of registers, the plurality of registers forming part of the container. Exemplar registers of the "Event" containers include at least "allDay", "availability", "birthdayPersonID", "endDate", "eventIdentifier", "isDetached", "organizer", "startDate", and "status". Exemplar registers of the "Reminder" containers include at least "completed", "completionDate", "dueDateComponent", and "startDateComponents". Exemplar registers of the "Alarm" containers include at least "absoluteDate", "proximity", "relativeOffset", and "structuredLocation". Exemplar registers of the "CLLocation" containers include at least "altitude", "coordinate", "course", "horizontalAccuracy", "speed", and "verticalAccuracy." Exemplar registers of the "CLLocationManager" containers include at least "activityType", "delegate", "desiredAccuracy", "distanceFilter", "heading", "headingFilter", "headingOrientation", "location", "maximumRegionMonitoringDistance", "monitoredRegions", and "pausesLocationUpdatesAutomatically". Exemplar registers of the "CLPlacemark" containers include at least "name", "addressDictionary", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", and "areasOfInterest". Exemplar registers of the "CLRegion" containers include "center", "identifier", and "radius". Exemplar registers of the "CLHeading" containers include "headingAccuracy", "magneticHeading", trueHeading", "y", and "z". (*See* claim element 1C for a more detailed discussion regarding the above registers.) Publicly available information indicates that registers are used throughout the iOS operating system of

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the iOS-Compatible Devices. The registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1E	a first register for storing a unique container identification value,	The plurality of registers includes a first register for storing a unique container identification value. On information and belief, all containers in each iOS device have first registers for storing a unique container identification value: Managed Object IDs and URIs An NSManagedObjectID object is a universal identifier for a managed object, and provides basis for uniquing in the Core Data Framework. A managed object ID uniquely identifies the same managed object both between managed object contexts in a single application, and in multiple applications (as in distributed systems). Like the primary key in the database, an identifier contains the information needed to exactly describe an object in a persistent store, although the detailed information is not exposed. The framework completely encapsulates the "external" information and presents a clean object oriented interface. `NSManagedObjectID *moID = [managedObject objectID];` There are two forms of an object ID. When a managed object is first created, Core Data assigns it a temporary ID; only if it is saved to a persistent store does Core Data assign a managed object a permanent ID. (EV0001881.) For example, the "eventIdentifier" registers store the unique container identification value of each of the "Event" containers:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		eventIdentifier
		A unique identifier for the event. (read-only)
		`@property(nonatomic, readonly) NSString *eventIdentifier`
		Discussion
		You can use this identifier to look up an event with the EKEventStore method `eventWithIdentifier:`.
		If the calendar of an event changes, its identifier most likely changes as well.
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		(EV0001425.)
		
		In another example, "Reminder" containers also have unique identifiers:
		
		Using Unique Identifiers
		If you know a specific reminder's unique identifier from previously fetching it with a predicate, you can call the `calendarItemWithIdentifier:` instance method. `calendarItemWithIdentifier:` can fetch any calendar item (reminders and events), whereas `eventWithIdentifier:` fetches only events.
		
		(EV0001416.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		calendarItemWithIdentifier: Returns either the event's first occurrence or the reminder with the specified identifier. - (EKCalendarItem *)calendarItemWithIdentifier:(NSString *)*identifier* **Parameters** *identifier* The calendar item's unique identifier. **Return Value** The reminder or the first occurrence of an event with the specified identifier. **Availability** Available in iOS 6.0 and later. **Declared In** EKEventStore.h (EV0001463.) calendarItemsWithExternalIdentifier: Returns either the event's first occurrences or the reminders with the specified external identifier. - (NSArray *)calendarItemsWithExternalIdentifier:(NSString *)*externalIdentifier* **Parameters** *externalIdentifier* The calendar item's external identifier. **Return Value** An array of calendar items with the specified identifier. **Discussion** The external identifier can be obtained from the calendarItemExternalIdentifier property. There may be more than one matching calendar item due to reasons discussed in calendarItemExternalIdentifier. **Availability** Available in iOS 6.0 and later. **Declared In** EKEventStore.h (EV0001463.) In addition, on information and belief, "Alarm" containers also have registers with unique identifiers.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		In another example, unique identifiers are associated with "Region" conainers:
		
		Defining a Region to Be Monitored
		To begin monitoring a region, you must define the region and register it with the system. You define regions using the `CLRegion` class, which currently supports only the creation of circular regions. Each region you create must include both the data that defines the desired geographic area and a unique identifier string. (The identifier string is required and is the only guaranteed way for your app to identify regions later.) To register a region, you call the `startMonitoringForRegion:desiredAccuracy:` method of your `CLLocationManager` object.
		
		Listing 1-4 shows a sample method that creates a new region based on a circular overlay region. The overlay's center point and radius form the boundary for the region, although if the radius is too large to be monitored, it is reduced automatically. You do not need to save strong references to the regions you create but might want to store the region's identifier if you plan to access the region information later.
		
		(EV0001549.)
		
		identifier
		The identifier for the region object. (read-only)
		`@property(readonly, nonatomic) NSString *identifier`
		Discussion
		This is a value that you specify and can use to identify this region inside your application.
		Availability
		Available in iOS 4.0 and later.
		Declared in
		`CLRegion.h`
		
		(EV0001687-88.)
		
		Publicly available information indicates that first registers are used throughout the iOS operating system of the iOS-Compatible Devices. The first registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the first registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

1F	a second register having a representation designating time and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus event time,	The plurality of registers includes a second register having a representation designating time and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus event time.
		
		For example, the "allDay", "endDate", and "startDate" registers of the "Event" containers designate time:
		
		allDay
		
		A Boolean value that indicates whether the event is an all-day event.
		
		`@property(nonatomic, getter=isAllDay) BOOL allDay`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		. . . .
		
		endDate
		
		The end date for the event.
		
		`@property(nonatomic, copy) NSDate *endDate`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDate
		
		The start date of the event.
		
		`@property(nonatomic, copy) NSDate *startDate`
		
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		(EV0001425-26.)
		
		In another example, the "completionDate", "dueDateComponents", and "startDateComponents" registers of the "Reminders" containers designate time:
		
		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		dueDateComponents The date by which the reminder should be completed. `@property(nonatomic, copy) NSDateComponents *dueDateComponents` **Discussion** The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised. This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone. **Special Considerations** On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X. **Availability** Available in iOS 6.0 and later. **Declared In**

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		EKReminder.h
		
		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		EKReminder.h
		
		(EV0001432-33.)
		
		In another example, the "absoluteDate" and "relativeOffset" registers of the "Alarms" containers designate time:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		. . . .
		relativeOffset
		
		The offset from the start of an event, at which the alarm fires.
		
		`@property NSTimeInterval relativeOffset`
		
		Discussion
		If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		(EV0001419.)
		
		Further, the second register governs interactions of the container with other containers, systems or processes according to utility of the time information relative to real-world time. In particular, the time information described above is used to govern many, if not most, aspects of the iOS Events, Reminders, and Alarms features (including, for example, reading and writing calendar events, reading and writing reminders, configuring alarms, creating recurring events, observing external changes to the calendar database, and providing interfaces for events). These and other aspects are discussed in more detail at EV0001435-59. Further, timestamps associated with location containers are used, for example, to determine whether the last measured location is outdated and whether a new measurement should be taken.
		
		Publicly available information indicates that second registers are used throughout the iOS operating

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		system of the iOS-Compatible Devices. The second registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the second registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1G	an active time register for identifying times at which the container will act upon other containers, processes, systems or gateways,	The plurality of registers includes an active time register for identifying times at which the container will act upon other containers, processes, systems or gateways. For example, the "allDay", "endDate", and/or "startDate" registers of the "Event" containers identify times at which the "Event" containers will act upon other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functionality: **allDay** A Boolean value that indicates whether the event is an all-day event. `@property(nonatomic, getter=isAllDay) BOOL allDay` **Availability** Available in iOS 4.0 and later. **Declared In** EKEvent.h . . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		endDate
		The end date for the event.
		`@property(nonatomic, copy) NSDate *endDate`
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		. . . .
		startDate
		The start date of the event.
		`@property(nonatomic, copy) NSDate *startDate`
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		(EV0001425-26.)
		
		In another example, the "completionDate", "dueDateComponents", and/or "startDateComponents" registers of the "Reminder"s containers identify times at which the "Reminders" containers will act upon other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		. . . .
		dueDateComponents
		
		The date by which the reminder should be completed.
		
		`@property(nonatomic, copy) NSDateComponents *dueDateComponents`
		
		Discussion
		The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone.
		
		Special Considerations
		On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		EKReminder.h
		
		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		EKReminder.h
		
		(EV0001432-33.)
		
		In another example, the "absoluteDate" and/or "relativeOffset" registers of the "Alarms" containers identify times at which the "Alarms" containers will act upon other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		. . . .
		relativeOffset
		
		The offset from the start of an event, at which the alarm fires.
		
		`@property NSTimeInterval relativeOffset`
		
		Discussion
		If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		(EV0001419.)
		
		Publicly available information indicates that active time registers are used throughout the iOS operating system of the iOS-Compatible Devices. The active time registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the active time registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

1H	a passive time register for identifying times at which the container can be acted upon by other containers, processes, systems or gateways, and	The plurality of registers includes a passive time register for identifying times at which the container can be acted upon other containers, processes, systems or gateways.
		
		For example, the "allDay", "endDate", and/or "startDate" registers of the "Event" containers identify times at which the "Event" containers can be acted upon by other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functionality:
		
		allDay
		
		A Boolean value that indicates whether the event is an all-day event.
		
		`@property(nonatomic, getter=isAllDay) BOOL allDay`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		. . . .
		endDate
		
		The end date for the event.
		
		`@property(nonatomic, copy) NSDate *endDate`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDate
		
		The start date of the event.
		
		`@property(nonatomic, copy) NSDate *startDate`
		
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		(EV0001425-26.)
		
		In another example, the "completionDate", "dueDateComponents", and/or "startDateComponents" registers of the "Reminders" containers identify times at which the "Reminders" containers can be acted upon by other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:
		
		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		dueDateComponents The date by which the reminder should be completed. `@property(nonatomic, copy) NSDateComponents *dueDateComponents` **Discussion** The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised. This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone. **Special Considerations** On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X. **Availability** Available in iOS 6.0 and later. **Declared In**

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		EKReminder.h
		
		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		EKReminder.h
		
		(EV0001432-33.)
		
		In another example, the "absoluteDate" and/or "relativeOffset" registers of the "Alarms" containers identify times at which the "Alarms" containers can be acted upon by other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		. . . .
		relativeOffset
		
		The offset from the start of an event, at which the alarm fires.
		
		`@property NSTimeInterval relativeOffset`
		
		Discussion
		If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		(EV0001419.)
		
		Publicly available information indicates that passive time registers are used throughout the iOS operating system of the iOS-Compatible Devices. The passive time registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the passive time registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

1I	a neutral time register for identifying times at which the container may interact with other containers, processes, systems or gateways; and	The plurality of registers includes a neutral time register for identifying times at which the container may interact with other containers, processes, systems or gateways.
		
		For example, the "allDay", "endDate", and/or "startDate" registers of the "Event" containers identify times at which the "Event" containers may interact with other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functionality:
		
		allDay
		
		A Boolean value that indicates whether the event is an all-day event.
		
		`@property(nonatomic, getter=isAllDay) BOOL allDay`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		. . . .
		endDate
		
		The end date for the event.
		
		`@property(nonatomic, copy) NSDate *endDate`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDate
		
		The start date of the event.
		
		`@property(nonatomic, copy) NSDate *startDate`
		
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		(EV0001425-26.)
		
		In another example, the "completionDate", "dueDateComponents", and/or "startDateComponents" registers of the "Reminders" containers identify times at which the "Reminders" containers may interact with other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:
		
		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		. . . .

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		dueDateComponents The date by which the reminder should be completed. `@property(nonatomic, copy) NSDateComponents *dueDateComponents` **Discussion** The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised. This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone. **Special Considerations** On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X. **Availability** Available in iOS 6.0 and later. **Declared In**

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		EKReminder.h
		
		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		EKReminder.h
		
		(EV0001432-33.)
		
		In another example, the "absoluteDate" and/or "relativeOffset" registers of the "Alarms" containers identify times at which the "Alarms" containers may interact with other containers, processes, systems, and/or gateways of the iOS operating system in order to carry out calendar-related, event-related, reminder-related, and alarm-related functions:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		. . . .
		relativeOffset
		
		The offset from the start of an event, at which the alarm fires.
		
		`@property NSTimeInterval relativeOffset`
		
		Discussion
		If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKAlarm.h
		
		(EV0001419.)
		
		Publicly available information indicates that neutral time registers are used throughout the iOS operating system of the iOS-Compatible Devices. The neutral time registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the neutral time registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
1J	a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes.	Each container of each iOS-Compatible Device comprises a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes. Some examples of gateways attached to and forming part of the Event, EventStore, Alarm, and Reminder containers include eventWithEventStore:, compareStartDateWithEvent:, refresh:, authorizationStatusForEntityType:, calendarItemsWithExternalIdentifier:, calendarItemWithIdentifier:, calendarsForEntityType:, calendarWithIdentifier:, cancelFetchRequest:, commit:, enumerateEventsMatchingPredicate:usingBlock:, eventsMatchingPredicate:, eventWithIdentifer:, fetchRemindersMatchingPredicate:completion:, predicateForCompletedRemindersWithCompletionDateStarting:ending:calendars:, predicateForEventsWithStartDate:endDate:calendars:, predicateForIncompleteRemindersWithDueDateStarting:ending:calendars:, predicateForRemindersInCalendars:, refreshSourcesIfNecessary:, removeCalendar:commit:error:, removeEvent:span:commit:error:, removeEvent:span:error:, removeReminder:commit:error:, requestAccessToEntityType:completion:, reset, saveCalendar:commit:error:, saveEvent:span:commit:error:, saveEvent:span:error:, saveReminder:commit:error:, sources, sourceWithIdentifier:, addAlarm:, addRecurrenceRule:, removeAlarm:, and removeRecurrenceRule: eventWithEventStore: Creates and returns a new event belonging to a specified event store. `+ (EKEvent *)eventWithEventStore:(EKEventStore *)eventStore` **Parameters** *eventStore* The event store to which the event belongs. **Return Value** The created event. **Availability** Available in iOS 4.0 and later. **Declared In** EKEvent.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		compareStartDateWithEvent: Compares the start date of the receiving event with the start date of another event. - (NSComparisonResult)compareStartDateWithEvent:(EKEvent *)*other* **Parameters** *other* The event to compare against. **Return Value** • NSOrderedAscending if the start date of the receiver precedes the start date of other. • NSOrderedSame if the start dates of the two events are identical. • NSOrderedDescending if the start date of the receiver comes after the start date of other. **Discussion** You can pass the selector for this method to the NSArray method sortedArrayUsingSelector: to create an array of events sorted by start date. **Availability** Available in iOS 4.0 and later. **Declared In** EKEvent.h refresh Updates the event's data with the current information in the Calendar database. - (BOOL)refresh **Return Value** If the event was successfully refreshed, YES; otherwise, NO. **Discussion** You should call this method only on events that your application is editing, and only when your application receives the EKEventStoreChangedNotification notification. If this method returns NO, the event has been deleted or otherwise invalidated, and you should not continue to use it. This method does not replace the values of any properties that you have modified. **Availability** Available in iOS 4.0 and later. **Declared In** EKEvent.h (EV0001427-28.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		authorizationStatusForEntityType:
		
		Returns the authorization status for the given entity type.
		
		`+ (EKAuthorizationStatus)authorizationStatusForEntityType:(EKEntityType)entityType`
		
		Parameters
		entityType
		The event or reminder entity type.
		
		Return Value
		The app's authorization status of the given type.
		
		Availability
		Available on iOS 6 and later.
		Declared In
		`EKEventStore.h`
		
		calendarItemsWithExternalIdentifier:
		
		Returns either the event's first occurrences or the reminders with the specified external identifier.
		
		`- (NSArray *)calendarItemsWithExternalIdentifier:(NSString *)externalIdentifier`
		
		Parameters
		externalIdentifier
		The calendar item's external identifier.
		
		Return Value
		An array of calendar items with the specified identifier.
		
		Discussion
		The external identifier can be obtained from the calendarItemExternalIdentifier property. There may be more than one matching calendar item due to reasons discussed in `calendarItemExternalIdentifier`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		calendarItemWithIdentifier: Returns either the event's first occurrence or the reminder with the specified identifier. - (EKCalendarItem *)calendarItemWithIdentifier:(NSString *)*identifier* **Parameters** *identifier* The calendar item's unique identifier. **Return Value** The reminder or the first occurrence of an event with the specified identifier. **Availability** Available in iOS 6.0 and later. **Declared In** EKEventStore.h calendarsForEntityType: Returns calendars that support a given entity type, such as reminders or events. - (NSArray *)calendarsForEntityType:(EKEntityType)*entityType* **Parameters** *entityType* The calendar's entity type. **Return Value** The calendar that supports the specified entity type.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Availability
		Available in iOS 6.0 and later.
		See Also
		`EKEntityType`
		Declared In
		`EKEventStore.h`
		
		calendarWithIdentifier:
		
		Returns the calendar with the specified identifier.
		
		`- (EKCalendar *)calendarWithIdentifier:(NSString *)`*identifier*
		
		Parameters
		identifier
		The calendar's unique identifier.
		
		Return Value
		The calendar with the specified identifier.
		
		Availability
		Available in iOS 5.0 and later.
		Declared In
		`EKEventStore.h`
		
		cancelFetchRequest:
		
		Cancels the request to fetch reminders.
		
		`- (void)cancelFetchRequest:(id)`*fetchIdentifier*
		
		Parameters
		fetchIdentifier
		The identifier of the request as returned by `fetchRemindersMatchingPredicate:completion:`.
		
		Discussion
		Once called, the completion block specified in `fetchRemindersMatchingPredicate:completion:` will not be called.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		commit:
		
		Commits all unsaved changes to the event store.
		
		`- (BOOL)commit:(NSError **)error`
		
		Parameters
		error
		A pointer to an `NSError` object. You do not need to create an `NSError` object. The commit operation aborts after the first failure if you pass `NULL`.
		
		Return Value
		If the commit operation succeeded, `YES`; otherwise, `NO`. Returns `YES` even when there are no changes to commit.
		
		Discussion
		This method allows you to save batched changes to the event store. For example, if you pass `NO` as the `commit` parameter to the `saveCalendar:commit:error:`, `removeCalendar:commit:error:`, `saveEvent:span:commit:error:`, or `removeEvent:span:commit:error:` methods, the changes are not saved until this method is invoked. Likewise, if you pass `YES` as the `commit` parameter to the aforementioned methods, there is no need to call this method.
		
		Availability
		Available in iOS 5.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		enumerateEventsMatchingPredicate:usingBlock: Finds all events that match a given predicate and calls a given callback for each event found. - (void)enumerateEventsMatchingPredicate:(NSPredicate *)*predicate* usingBlock: (EKEventSearchCallback)*block* **Parameters** *predicate* The search predicate. Must be created with the predicateForEventsWithStartDate:endDate:calendars: method. *block* The block callback to call for each event. The callback must match the signature defined by EKEventSearchCallback. **Discussion** Only events that have been committed are included in enumeration. Events saved using saveEvent:span:commit:error: with the *commit* parameter set to NO must call commit: beforehand to be included. This method is synchronous. For asynchronous behavior, run the method on another thread with dispatch_async or NSOperation. **Availability** Available in iOS 4.0 and later. **See Also** – eventsMatchingPredicate: EKEventSearchCallback – predicateForEventsWithStartDate:endDate:calendars: **Declared In** EKEventStore.h eventsMatchingPredicate: Returns all events that match a given predicate. - (NSArray *)eventsMatchingPredicate:(NSPredicate *)*predicate* **Parameters** *predicate*

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The search predicate. Must be created with the `predicateForEventsWithStartDate:endDate:calendars:` method.
		Return Value
		All events that match `predicate`, as an array of `EKEvent` objects.
		Discussion
		Only events that have been committed are included in the results. Events saved using `saveEvent:span:commit:error:` with the *commit* parameter set to `NO` must call `commit:` beforehand to be included.
		This method is synchronous. For asynchronous behavior, run the method on another thread with `dispatch_async` or `NSOperation`.
		Availability
		Available in iOS 4.0 and later.
		See Also
		– `enumerateEventsMatchingPredicate:usingBlock:`
		– `predicateForEventsWithStartDate:endDate:calendars:`
		Declared In
		`EKEventStore.h`
		
		eventWithIdentifier:
		Returns the first occurrence of an event with a given identifier.
		`- (EKEvent *)eventWithIdentifier:(NSString *)identifier`
		Parameters
		identifier
		The identifier of the event.
		Return Value
		The event corresponding to `identifier`, or `nil` if no event is found.
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		fetchRemindersMatchingPredicate:completion: Fetches reminders matching a given predicate. - (id)fetchRemindersMatchingPredicate:(NSPredicate *)*predicate* completion:(void (^)(NSArray *))*completion* **Parameters** *predicate* The search predicate. *completion* An array of the matched reminders passed by reference. **Return Value** A value to be used in `cancelFetchRequest:` to cancel the request later if desired. **Discussion** Only reminders that have been committed are included in the results. Reminders saved using `saveReminder:commit:error:` with the *commit* parameter set to `NO` must call `commit:` beforehand to be included. This method fetches reminders asynchronously. **Availability** Available in iOS 6.0 and later. **See Also** – `predicateForRemindersInCalendars:` – `predicateForIncompleteRemindersWithDueDateStarting:ending:calendars:` – `predicateForCompletedRemindersWithCompletionDateStarting:ending:calendars:` **Declared In** `EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		predicateForCompletedRemindersWithCompletionDateStarting:ending:calendars: Fetches completed reminders in a set of calendars within an optional range. - (NSPredicate *)predicateForIncompleteRemindersWithDueDateStarting:(NSDate *)*startDate* ending:(NSDate *)*endDate* calendars:(NSArray *)*calendars* **Parameters** *startDate* The starting bound of the range to search. *endDate* The ending bound of the range to search. *calendars* An array of calendars to search. **Return Value** The created predicate to be used for fetchRemindersMatchingPredicate:completion:. **Discussion** Pass nil for *startDate* to find all reminders completed before *endDate*. Similarly, pass nil for both *startDate* and *endDate* to get all complete reminders in the specified calendars. **Availability** Available in iOS 6.0 and later. **Declared In** EKEventStore.h predicateForEventsWithStartDate:endDate:calendars: Creates and returns a predicate for finding events in the event store that fall within a given date range. - (NSPredicate *)predicateForEventsWithStartDate:(NSDate *)*startDate* endDate:(NSDate *)*endDate* calendars:(NSArray *)*calendars* **Parameters** *startDate* The start date of the range of events fetched. *endDate*

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The end date of the range of events fetched.
		calendars
		The calendars to search, as an array of `EKCalendar` objects. Passing `nil` indicates to search all calendars.
		Return Value
		The created predicate.
		Availability
		Available in iOS 4.0 and later.
		See Also
		– `eventsMatchingPredicate:`
		– `enumerateEventsMatchingPredicate:usingBlock:`
		Declared In
		`EKEventStore.h`
		
		predicateForIncompleteRemindersWithDueDateStarting:ending:calendars:
		Fetches incomplete reminders in a set of calendars within an optional range.
		`- (NSPredicate *)predicateForIncompleteRemindersWithDueDateStarting:(NSDate *)startDate ending:(NSDate *)endDate calendars:(NSArray *)calendars`
		Parameters
		startDate
		The starting bound of the range to search.
		endDate
		The ending bound of the range to search.
		calendars
		An array of calendars to search.
		Return Value
		The created predicate to be used for `fetchRemindersMatchingPredicate:completion:`.
		Discussion
		Pass `nil` for *startDate* to find all reminders due before *endDate*. Similarly, pass `nil` for both *startDate* and *endDate* to get all incomplete reminders in the specified calendars.
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		predicateForRemindersInCalendars: Fetches all reminders in a set of calendars. - (NSPredicate *)predicateForRemindersInCalendars:(NSArray *)*calendars* **Parameters** *calendars* An array of calendars to search. **Return Value** The created predicate to be used for fetchRemindersMatchingPredicate:completion:. **Availability** Available in iOS 6.0 and later. **Declared In** EKEventStore.h refreshSourcesIfNecessary Pulls new data from remote sources if necessary. - (void)refreshSourcesIfNecessary **Discussion** Use this method to pull new data from remote sources if the local data is out of date. **Availability** Available in iOS 5.0 and later. **Declared In** EKEventStore.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		removeCalendar:commit:error: Removes a calendar from the event store by either batching or committing the changes. - (BOOL)removeCalendar:(EKCalendar *)*calendar* commit:(BOOL)*commit* error:(NSError **)*error* **Parameters** *calendar* The calendar to be removed. *commit* YES to remove the calendar immediately; otherwise, the change is batched until the commit: method is invoked. *error* The error that occurred, if any; otherwise, nil. **Return Value** YES if successful; otherwise, NO. **Discussion** This method raises an exception if *calendar* belongs to another event store. **Availability** Available in iOS 5.0 and later. **See Also** – commit: – saveCalendar:commit:error: **Declared In** EKEventStore.h removeEvent:span:commit:error: Removes an event or recurring events from the event store by either batching or committing the changes. - (BOOL)removeEvent:(EKEvent *)*event* span:(EKSpan)*span* commit:(BOOL)*commit* error:(NSError

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		)*error* **Parameters *event* The event to remove. *span* The span to use. Indicates whether the remove affects future instances of the event in the case of a recurring event. *commit* `YES` to remove the event immediately; otherwise, the change is batched until the `commit:` method is invoked. *error* The error that occurred, if any did. Otherwise, `nil`. **Return Value** If the event has successfully removed, `YES`; otherwise, `NO`. Also returns `NO` if `event` cannot be removed because it is not in the event store. **Discussion** This method raises an exception if it is passed an event from another event store. **Availability** Available in iOS 5.0 and later. **See Also** – `commit:` – `removeEvent:span:error:` – `saveEvent:span:error:` – `saveEvent:span:commit:error:` **Declared In** `EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		removeEvent:span:error: Removes an event from the event store. - (BOOL)removeEvent:(EKEvent *)*event* span:(EKSpan)*span* error:(NSError **)*error* **Parameters** *event* The event to be removed. *span* The span to use. Indicates whether to remove future instances of the event in the case of a recurring event. *error* The error if one occurred; otherwise, nil. **Return Value** If the event has successfully removed, YES; otherwise, NO. Also returns NO if event cannot be removed because it is not in the event store. **Discussion** This method raises an exception if it is passed an event from another event store. **Availability** Available in iOS 4.0 and later. **See Also**

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		– saveEvent:span:error: – saveEvent:span:commit:error: – removeEvent:span:commit:error: **Declared In** EKEventStore.h removeReminder:commit:error: Removes a reminder from the event store by either committing or batching the changes. - (BOOL)removeReminder:(EKReminder *)*reminder* commit:(BOOL)*commit* error:(NSError **)*error* **Parameters** *reminder* The reminder to be removed. *commit* A Boolean value indicating whether to remove the reminder immediately or to batch the removals; passing NO will not commit the removal from the event store until the commit: method is invoked. *error* The error that occurred, if any; otherwise, nil. **Return Value** If successful, YES; otherwise, NO. **Discussion** This method raises an exception if *reminder* belongs to another event store. **Availability** Available in iOS 6.0 and later. **See Also** – commit: – saveReminder:commit:error: **Declared In** EKEventStore.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		requestAccessToEntityType:completion: Prompts the user to grant or deny access to event or reminder data. `- (void)requestAccessToEntityType:(EKEntityType)entityType completion:(EKEventStoreRequestAccessCompletionHandler)completion` **Parameters** *entityType* The event or reminder entity type. *completion* The block to call when the request completes. **Discussion** In iOS 6 and later, requesting access to an event store asynchronously prompts your users for permission to use their data. The user is only prompted the first time your app requests access to an entity type; any subsequent instantiations of `EKEventStore` uses existing permissions. When the user taps to grant or deny access, the completion handler will be called on an arbitrary queue. Your app is not blocked while the user decides to grant or deny permission.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		After users choose their permission level, the event store either calls the completion handler or broadcasts an `EKEventStoreChangedNotification`. The completion handler is called on iOS 6 and later, and the notification is broadcasted on iOS 5. Because users may deny access to the event store, your app should handle an empty data case.
		
		Important: If your app has never requested access before, you must request access to events or reminders before attempting to fetch or create them. If you request data before prompting the user for access with this method, you'll need to reset the event store with the `reset` method in order to start receiving data once the user grants access.
		
		Availability
		Available on iOS 6 and later.
		Declared In
		`EKEventStore.h`
		
		reset
		
		Returns the event store to its saved state.
		
		`- (void)reset`
		
		Discussion
		This method updates all the properties of all the objects with their corresponding values in the event store. Any local changes that were not saved before invoking this method will be lost. All objects that were created or retrieved using this store are disassociated from it and should be considered invalid.
		
		Availability
		Available in iOS 5.0 and later.
		See Also
		`reset`
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		saveCalendar:commit:error:
		
		Saves a calendar to the event store by either committing or batching the changes.
		
		`- (BOOL)saveCalendar:(EKCalendar *)calendar commit:(BOOL)commit error:(NSError **)error`
		
		Parameters
		calendar
		The calendar to be saved.
		commit
		`YES` to save the calendar immediately; otherwise, the change is batched until the `commit:` method is invoked.
		error
		The error that occurred, if any; otherwise, `nil`.
		
		Return Value
		`YES` if successful; otherwise, `NO`.
		
		Discussion
		This method raises an exception if *calendar* belongs to another event store.
		Availability
		Available in iOS 5.0 and later.
		See Also
		– `commit:`
		– `removeCalendar:commit:error:`
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		saveEvent:span:commit:error: Saves an event or recurring events to the event store by either batching or committing the changes. - (BOOL)saveEvent:(EKEvent *)*event* span:(EKSpan)*span* commit:(BOOL)*commit* error:(NSError **)*error* **Parameters** *event* The event to be saved. *span* The span to use. Indicates whether the save affects future instances of the event in the case of a recurring event. *commit* To save the event immediately, pass YES; otherwise, the change is batched until the commit: method is invoked. *error* The error that occurred, if any; otherwise, nil. **Return Value** If successful, YES; otherwise, NO. Also returns NO if event does not need to be saved because it has not been modified. **Discussion** This method raises an exception if it is passed an event from another event store. When an event is saved, it is updated in the Calendar database. Any fields you did not modify are updated to reflect the most recent value in the database. If the event has been deleted from the database, it is re-created as a new event. **Availability** Available in iOS 5.0 and later. **See Also** – commit: – saveEvent:span:error: – removeEvent:span:error: – removeEvent:span:commit:error: **Declared In** EKEventStore.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		saveReminder:commit:error:
		
		Saves changes to a reminder by either committing or batching the changes.
		
		`- (BOOL)saveReminder:(EKReminder *)reminder commit:(BOOL)commit error:(NSError **)error`
		
		Parameters
		reminder
		The reminder to be saved.
		commit
		A Boolean value indicating whether to save the reminder immediately or to batch the changes; passing `NO` will not commit changes to the event store until the `commit:` method is invoked.
		error
		The error that occurred, if any; otherwise, `nil`.
		
		Return Value
		If successful, `YES`; otherwise, `NO`.
		
		Discussion
		This method raises an exception if *reminder* belongs to another event store.
		
		Availability
		Available in iOS 6.0 and later.
		See Also
		`– commit:`
		`– removeReminder:commit:error:`
		
		Declared In
		`EKEventStore.h`
		
		sources
		
		Returns an unordered array of source objects.
		
		`- (NSArray *)sources`
		
		Return Value
		An unordered array of `EKSource` objects.
		
		Discussion
		An EKSource object represents an account that contains calendars.
		
		Availability
		Available in iOS 5.0 and later.
		Declared In
		`EKEventStore.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		sourceWithIdentifier: Returns a source with the specified identifier. - (EKSource *)sourceWithIdentifier:(NSString *)*identifier* **Parameters** *identifier* The source's unique identifier. **Return Value** The source with the specified identifier. **Availability** Available in iOS 5.0 and later. **Declared In** EKEventStore.h (EV0001462-75.) addAlarm: *Adds an alarm to the receiver.* - (void)addAlarm:(EKAlarm *)alarm **Parameters** alarm The alarm to be added. **Availability** Available in iOS 5.0 and later. **See Also** @property alarms (page 6) @property hasAlarms (page 9) - removeAlarm: (page 15) **Declared in** EKCalendarItem.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		addRecurrenceRule:
		
		Adds a recurrence rule to the recurrence rule array.
		
		`- (void)addRecurrenceRule:(EKRecurrenceRule *)rule`
		
		Parameters
		`rule`
		The rule to be added to `recurrenceRules` (page 12).
		
		Availability
		Available in iOS 5.0 and later.
		
		See Also
		`- removeRecurrenceRule:` (page 15)
		`@property recurrenceRules` (page 12)
		`@property hasRecurrenceRules` (page 10)
		
		Declared in
		`EKCalendarItem.h`
		
		removeAlarm:
		
		Removes an alarm from the calendar item.
		
		`- (void)removeAlarm:(EKAlarm *)alarm`
		
		Parameters
		`alarm`
		The alarm to be removed.
		
		Availability
		Available in iOS 5.0 and later.
		
		See Also
		`@property alarms` (page 6)
		`- addAlarm:` (page 14)
		`@property hasAlarms` (page 9)
		
		Declared in
		`EKCalendarItem.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		removeRecurrenceRule:
		
		Removes a recurrence rule from the recurrence rule array.
		
		`- (void)removeRecurrenceRule:(EKRecurrenceRule *)rule`
		
		Parameters
		`rule`
		The rule to be removed from `recurrenceRules` (page 12).
		
		Availability
		Available in iOS 5.0 and later.
		
		See Also
		`- addRecurrenceRule:` (page 14)
		`@property recurrenceRules` (page 12)
		`@property hasRecurrenceRules` (page 10)
		
		Declared in
		`EKCalendarItem.h`
		
		(EV0001491-92.)
		
		Some examples of gateways attached to and forming part of the CLLocation, CLLocationManager, CLPlacemark, CLRegion, and CLHeading containers include distanceFromLocation, initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:course:speed: timestamp:, initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:timestamp:, initWithLatitude:longitude:, delegate, authorizationStatus, locationServicesEnabled, deferredLocationUpdatesAvailable, significantLocationChangeMonitoringAvailable, headingAvailable, regionMonitoringAvailable, startUpdatingLocation, stopUpdatingLocation, pausesLocationUpdatesAutomatically, distanceFilter, desiredAccuracy, activityType, startMonitoringSignificantLocationChanges, stopMonitoringSignificantLocationChanges, startUpdatingHeading, stopUpdatingHeading, dismissHeadingCalibrationDisplay, headingFilter, headingOrientation, startMonitoringForRegion, stopMonitoringForRegion, monitoredRegions, maximumRegionMonitoringDistance, allowDeferredLocationUpdatesUntilTraveled:timeout:, disallowDeferredLocationUpdates, location, heading, purpose, initWithPlacemark:, containsCoordinate:, and initCircularRegionWithCenter:radius:identifier:

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		description
		
		Returns the location data in a formatted text string.
		
		`- (NSString *)description`
		
		Return Value
		A string of the form "<<*latitude*>, <*longitude*>> +/- <*accuracy*> m (speed <*speed*> kph / heading <*heading*>) @ <*date-time*>", where <*latitude*>, <*longitude*>, <*accuracy*>, <*speed*>, and <*heading*> are formatted floating point numbers and <*date-time*> is a formatted date string that includes date, time, and time zone information.
		
		Discussion
		The returned string is intended for display purposes only.
		
		Availability
		Available in iOS 2.0 and later.
		
		Declared in
		`CLLocation.h`
		
		distanceFromLocation:
		
		Returns the distance (in meters) from the receiver's location to the specified location.
		
		`- (CLLocationDistance)distanceFromLocation:(const CLLocation *)location`
		
		Parameters
		`location`
		The other location.
		
		Return Value
		The distance (in meters) between the two locations.
		
		Discussion
		This method measures the distance between the two locations by tracing a line between them that follows the curvature of the Earth. The resulting arc is a smooth curve and does not take into account specific altitude changes between the two locations.
		
		Availability
		Available in iOS 3.2 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Related Sample Code
		GeocoderDemo
		
		Declared in
		`CLLocation.h`
		
		initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:timestamp:
		
		Initializes and returns a location object with the specified coordinate information.
		
		`- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate altitude:(CLLocationDistance)altitude horizontalAccuracy:(CLLocationAccuracy)hAccuracy verticalAccuracy:(CLLocationAccuracy)vAccuracy timestamp:(NSDate *)timestamp`
		
		Parameters
		`coordinate`
		A coordinate structure containing the latitude and longitude values.
		`altitude`
		The altitude value for the location.
		`hAccuracy`
		The accuracy of the coordinate value. Specifying a negative number indicates that the coordinate value is invalid.
		`vAccuracy`
		The accuracy of the altitude value. Specifying a negative number indicates that the altitude value is invalid.
		`timestamp`
		The time to associate with the location object. Typically, you would set this to the current time.
		
		Return Value
		A location object initialized with the specified information.
		
		Discussion
		Typically, you acquire location objects from the location service, but you can use this method to create new location objects for other uses in your application.
		
		Availability
		Available in iOS 2.0 and later.
		
		Declared in
		`CLLocation.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		initWithLatitude:longitude:
		
		Initializes and returns a location object with the specified latitude and longitude.
		
		`- (id)initWithLatitude:(CLLocationDegrees)latitude longitude:(CLLocationDegrees)longitude`
		
		Parameters
		`latitude`
		The latitude of the coordinate point.
		`longitude`
		The longitude of the coordinate point.
		
		Return Value
		A location object initialized with the specified coordinate point.
		
		Discussion
		Typically, you acquire location objects from the location service, but you can use this method to create new location objects for other uses in your application. When using this method, the other properties of the object are initialized to appropriate values. In particular, the `altitude` and `horizontalAccuracy` properties are set to 0, the `verticalAccuracy` property is set to -1 to indicate that the altitude value is invalid, and the `timestamp` property is set to the time at which the instance was initialized.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		AVMovieExporter
		GeocoderDemo
		pARk
		
		Declared in
		`CLLocation.h`
		
		(EV0001518-21.)
		
		Accessing the Delegate
		
		`delegate` (page 11) *property*
		The delegate object to receive update events.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Determining the Availability of Services + `authorizationStatus` (page 17) Returns the application's authorization status for using location services. + `locationServicesEnabled` (page 18) Returns a Boolean value indicating whether location services are enabled on the device. + `deferredLocationUpdatesAvailable` (page 17) Returns a Boolean value indicating whether the device supports deferred location updates. + `significantLocationChangeMonitoringAvailable` (page 19) Returns a Boolean value indicating whether significant location change tracking is available. + `headingAvailable` (page 18) Returns a Boolean value indicating whether the location manager is able to generate heading-related events. + `regionMonitoringAvailable` (page 19) Returns a Boolean value indicating whether region monitoring is supported on the current device. **Initiating Standard Location Updates** – `startUpdatingLocation` (page 25) Starts the generation of updates that report the user's current location. – `stopUpdatingLocation` (page 27) Stops the generation of location updates. `pausesLocationUpdatesAutomatically` (page 16) *property* A Boolean value indicating whether the location manager object may pause location updates. `distanceFilter` (page 12) *property* The minimum distance (measured in meters) a device must move horizontally before an update event is generated. `desiredAccuracy` (page 12) *property* The accuracy of the location data. `activityType` (page 11) *property* The type of user activity associated with the location updates.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Initiating Significant Location Updates – `startMonitoringSignificantLocationChanges` (page 23) Starts the generation of updates based on significant location changes. – `stopMonitoringSignificantLocationChanges` (page 26) Stops the delivery of location events based on significant location changes. **Initiating Heading Updates** – `startUpdatingHeading` (page 24) Starts the generation of updates that report the user's current heading. – `stopUpdatingHeading` (page 27) Stops the generation of heading updates. – `dismissHeadingCalibrationDisplay` (page 21) Dismisses the heading calibration view from the screen immediately. `headingFilter` (page 13) *property* The minimum angular change (measured in degrees) required to generate new heading events. `headingOrientation` (page 14) *property* The device orientation to use when computing heading values.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Initiating Region Monitoring – `startMonitoringForRegion:` (page 22) Starts monitoring the specified region. – `stopMonitoringForRegion:` (page 26) Stops monitoring the specified region. `monitoredRegions` (page 16) *property* The set of shared regions monitored by all location manager objects. (read-only) `maximumRegionMonitoringDistance` (page 15) *property* The largest boundary distance that can be assigned to a region. (read-only) **Deferring Location Updates** – `allowDeferredLocationUpdatesUntilTraveled:timeout:` (page 20) Tells the location manager to defer the delivery of location data until one of the specified criteria is met. – `disallowDeferredLocationUpdates` (page 21) Cancels the deferral of location updates for this app. **Getting Recently Retrieved Data** `location` (page 14) *property* The most recently retrieved user location. (read-only) `heading` (page 13) *property* The most recently reported heading. (read-only) **Describing Your Application's Services to the User** `purpose` (page 32) *property* Deprecated in iOS 6.0 An application-provided string that describes the reason for using location services. (EV0001628-31.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		initWithPlacemark:
		
		Initializes and returns a placemark object from another placemark object.
		
		`- (id)initWithPlacemark:(CLPlacemark *)placemark`
		
		Parameters
		`placemark`
		The placemark object to use as the source of the data for the new object.
		
		Return Value
		A new placemark object.
		
		Discussion
		You can use this method to transfer information from one placemark object to another placemark object.
		
		Availability
		Available in iOS 5.0 and later.
		
		Declared in
		`CLPlacemark.h`
		
		(EV0001669.)
		
		containsCoordinate:
		
		Returns a Boolean value indicating whether the region contains the specified coordinate.
		
		`- (BOOL)containsCoordinate:(CLLocationCoordinate2D)coordinate`
		
		Parameters
		`coordinate`
		The coordinate to test against the region.
		
		Return Value
		`YES` if the coordinate lies within the region's boundaries or `NO` if it does not.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLRegion.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		initCircularRegionWithCenter:radius:identifier:
		
		Initializes and returns a region object defining a circular area.
		
		`- (id)initCircularRegionWithCenter:(CLLocationCoordinate2D)center radius:(CLLocationDistance)radius identifier:(NSString *)identifier`
		
		Parameters
		`center`
		The center point of the region.
		`radius`
		The distance (measured in meters) from the center point that marks the boundary of the region.
		`identifier`
		A unique identifier to associate with the region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.
		
		Return Value
		An initialized region object.
		
		Availability
		Available in iOS 4.0 and later.
		
		Related Sample Code
		GeocoderDemo
		Regions
		
		Declared in
		`CLRegion.h`
		
		(EV0001688-89.)
		
		Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2A	An apparatus for transmitting, receiving and manipulating information on a computer system, the apparatus including	Each iOS-Compatible Device is an apparatus for transmitting, receiving and manipulating information on a computer system. (*See* the claim elements below.) Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2B	a plurality of containers, each container being a logically defined data enclosure and comprising:	Each iOS-Compatible Device includes a plurality of containers, each container being a logically defined data enclosure. (*See* the discussion presented for claim element 1B, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that containers are used throughout the iOS operating system of the iOS-Compatible Devices. The containers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the containers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2C	an information element having information;	Each container of each iOS-Compatible Device comprises an information element having information. (*See* the discussion presented for claim element 1C, which is incorporated by reference as if fully set forth herein.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Publicly available information indicates that information elements are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the information elements are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2D	a plurality of registers, the plurality of registers forming part of the container and including	Each container of each iOS-Compatible Device comprises a plurality of registers, the plurality of registers forming part of the container. (*See* the discussion presented for claim element 1D, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that registers are used throughout the iOS operating system of the iOS-Compatible Devices. The registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2E	a first register for storing a unique container identification	The plurality of registers includes a first register for storing a unique container identification value. (*See* the discussion presented for claim element 1E, which is incorporated by reference as if fully set forth

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	value,	herein.) Publicly available information indicates that first registers are used throughout the iOS operating system of the iOS-Compatible Devices. The first registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the first registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2F	a second register having a representation designating space and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus three-dimensional space,	The plurality of registers includes a second register having a representation designating space and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus three-dimensional space. For example, at least the following registers of the "CLLocation" container designate geographical space: "altitude," "coordinate," "course," "horizontalAccuracy," and "verticalAccuracy." In another example, at least the following registers of the "CLLocationManager" container designate geographical space: "location" and "desiredAccuracy." In another example, at least the following registers of the "CLPlacemark" container designate geographical space: "ISOcountryCode," "country," "postalCode," "administrativeArea," "subAdministrativeArea," "locality," "subLocality," "thoroughfare," "subThoroughfare," "region," "inlandWater," "ocean," and "areasOfInterest". In another example, at least the following registers of the "CLRegion" container designate geographical space: "center," "identifier," and "radius". Further, the second register governs interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus three-dimensional space.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		For example, the second registers identified above govern interactions of the corresponding containers with other containers, systems, or processes according to utility of information in the information element relative to an external-to-the-apparatus three-dimensional space, in order to perform location-based services such as navigation, mapping, targeted advertising, connecting with nearby users, improving the quality of information offered to users, etc. Publicly available information indicates that second registers are used throughout the iOS operating system of the iOS-Compatible Devices. The second registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the second registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2G	an active space register for identifying space in which the container will act upon other containers, processes, systems or gateways,	The plurality of registers includes an active space register for identifying space in which the container will act upon other containers, processes, systems or gateways. For example, at least the following registers of the "LLocation," "CLLocationManager," "CLPlacemark," and "CLRegion" containers identify space in which the corresponding container will act upon other containers, processes, systems or gateways to, among other things, perform location-based services such as navigation, mapping, targeted advertising, connecting with nearby users, improving the quality of information offered to users, etc.: "altitude", "coordinate", "course", "horizontalAccuracy", "verticalAccuracy", "location", "desiredAccuracy", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", "areasOfInterest", "center", "identifier", and "radius". Publicly available information indicates that active space registers are used throughout the iOS operating system of the iOS-Compatible Devices. The active space registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		It is believed that the structure and operation of the active space registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2H	a passive register for identifying space in which the container can be acted upon by other containers, processes, systems or gateways,	The plurality of registers includes a passive register for identifying space in which the container can be acted upon other containers, processes, systems or gateways. For example, at least the following registers of the "CLLocation," "CLLocationManager," "CLPlacemark," and "CLRegion" containers identify space in which the corresponding container can be acted upon by other containers, processes, systems or gateways to, among other things, perform location-based services such as navigation, mapping, targeted advertising, connecting with nearby users, improving the quality of information offered to users, etc.: "altitude", "coordinate", "course", "horizontalAccuracy", "verticalAccuracy", "location", "desiredAccuracy", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", "areasOfInterest", "center", "identifier", and "radius". Publicly available information indicates that passive registers are used throughout the iOS operating system of the iOS-Compatible Devices. The passive registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the passive registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		function, in substantially the same way, to reach substantially the same result.
2I	a neutral space register for identifying space in which the container may interact with other containers, processes, systems, or gateways; and	The plurality of registers includes a neutral space register for identifying space in which the container may interact with other containers, processes, systems or gateways. For example, at least the following registers of the "CLLocation," "CLLocationManager," "CLPlacemark," and "CLRegion" containers identify space in which the corresponding container may interact with other containers, processes, systems or gateways to, among other things, perform location-based services such as navigation, mapping, targeted advertising, connecting with nearby users, improving the quality of information offered to users, etc.: "altitude", "coordinate", "course", "horizontalAccuracy", "verticalAccuracy", "location", "desiredAccuracy", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", "areasOfInterest", "center", "identifier", and "radius". Publicly available information indicates that neutral space registers are used throughout the iOS operating system of the iOS-Compatible Devices. The neutral space registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the neutral space registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
2J	a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other	Each container of each iOS-Compatible Device comprises a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes. (*See* the discussion presented for claim element 1J, which is incorporated by reference as if fully set forth herein.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	containers, systems or processes.	Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
3	The apparatus of claim 1 or 2, wherein the plurality of registers includes at least one container history register for storing information regarding past interaction of the container with other containers, systems or processes, the container history register being modifiable.	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes at least one container history register for storing information regarding past interaction of the container with other containers, systems or processes, the container history register being modifiable. For example, at least the following modifiable container history registers store information regarding past interaction of the container with other systems and processes responsible for updating the values of those registers: “Event” container history registers include: “allDay”, “availability”, “birthdayPersonID”, “endDate”, “eventIdentifier”, “isDetached”, “organizer”, “startDate”, and “status”. “Reminder” container history registers include: “completed”, “completionDate”, “dueDateComponent”, and “startDateComponents”. “Alarm” container history registers include: “absoluteDate”, “proximity”, “relativeOffset”, and “structuredLocation”. “CLLocation” history registers include: “altitude”, “coordinate”, “course”, “horizontalAccuracy”, “speed”, and “verticalAccuracy.”

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		"CLLocationManager" history registers include: "activityType", "delegate", "desiredAccuracy", "distanceFilter", "heading", "headingFilter", "headingOrientation", "location", "maximumRegionMonitoringDistance", "monitoredRegions", and "pausesLocationUpdatesAutomatically". "CLPlacemark" history registers include: "name", "addressDictionary", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", and "areasOfInterest". "CLRegion" history registers include: "center", "identifier", and "radius". "CLHeading" history registers include: "headingAccuracy", "magneticHeading", "trueHeading", "y", and "z". Publicly available information indicates that the container history registers are used throughout the iOS operating system of the iOS-Compatible Devices. The container history registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the container history registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
4	The apparatus of claim 1 or 2, wherein the plurality of registers includes at least one system history register for storing information	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes at least one system history register for storing information regarding past interaction of the container with different operating system and network processes. For example, at least the following system history registers store information regarding past interaction

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	regarding past interaction of the container with different operating system and network processes.	of the container with different operating system (i.e., iOS) system and network processes: "Event" system history registers include: "allDay", "availability", "birthdayPersonID", "endDate", "eventIdentifier", "isDetached", "organizer", "startDate", and "status". "Reminder" system history registers include: "completed", "completionDate", "dueDateComponent", and "startDateComponents". "Alarm" system history registers include: "absoluteDate", "proximity", "relativeOffset", and "structuredLocation". "CLLocation" system history registers include: "altitude", "coordinate", "course", "horizontalAccuracy", "speed", and "verticalAccuracy." "CLLocationManager" system history registers include: "activityType", "delegate", "desiredAccuracy", "distanceFilter", "heading", "headingFilter", "headingOrientation", "location", "maximumRegionMonitoringDistance", "monitoredRegions", and "pausesLocationUpdatesAutomatically". "CLPlacemark" system history registers include: "name", "addressDictionary", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", and "areasOfInterest". "CLRegion" system history registers include: "center", "identifier", and "radius". "CLHeading" system history registers include: "headingAccuracy", "magneticHeading", "trueHeading", "y", and "z". Publicly available information indicates that system history registers are used throughout the iOS operating system of the iOS-Compatible Devices. The system history registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the system history registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence

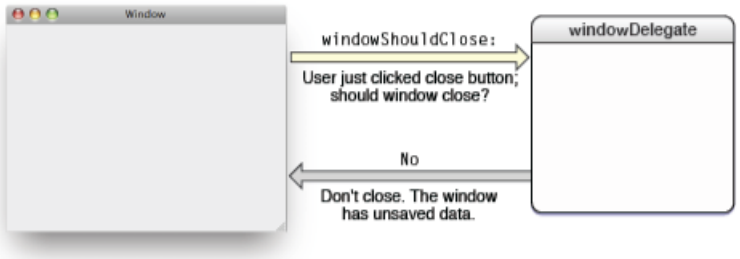
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
5	The apparatus of claim 1 or 2, wherein the plurality of registers includes at least one predefined register, the predefined register being a register associated with an editor for user selection and being appendable to any container.	Each iOS-Compatible Device infringes claim 1 and claim 2 (see above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes at least one predefined register, the predefined register being a register associated with an editor for user selection and being appendable to any container. Most registers of the iOS operating system with default values qualify. Examples include the "proximity" register of the "Alarm" container, and the "desiredAccuracy" and "distancefilter" registers of the "CLLocationManager" container. These registers can be edited by a user through the iOS gateways provided by each iOS-Compatible Device, and are appendable to containers such as the "Alarm" container and the "CLLocationManager" container. Publicly available information indicates that predefined registers are used throughout the iOS operating system of the iOS-Compatible Devices. The predefined registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the predefined registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
6	The apparatus of claim 1 or 2, wherein the plurality of registers	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes a user-created register, the user-created register being generated by the user, and being appendable to any container.

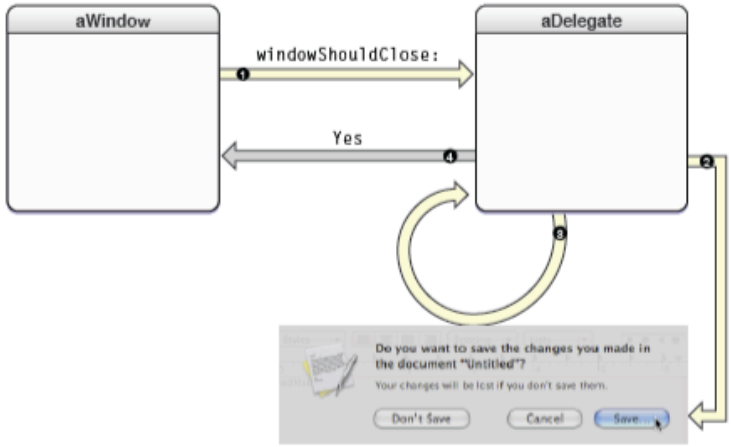
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	includes a user-created register, the user-created register being generated by the user, and being appendable to any container.	Examples of user-created registers include the following: "Event" container user-created registers include: "allDay", "availability", "birthdayPersonID", "endDate", "eventIdentifier", "isDetached", "organizer", "startDate", and "status". "Reminder" container user-created registers include: "completionDate", "dueDateComponent", and "startDateComponents". "Alarm" container user-created registers include: "absoluteDate", "proximity", "relativeOffset", and "structuredLocation". "CLLocation" container user-created registers include: "altitude", "coordinate", "course", "horizontalAccuracy", "speed", and "verticalAccuracy." "CLLocationManager" container user-created registers include: "activityType", "delegate", "desiredAccuracy", "distanceFilter", "heading", "headingFilter", "headingOrientation", "location", "maximumRegionMonitoringDistance", "monitoredRegions", and "pausesLocationUpdatesAutomatically". "CLPlacemark" container user-created registers include: "name", "addressDictionary", "ISOcountryCode", "country", "postalCode", "administrativeArea", "subAdministrativeArea", "locality", "subLocality", "thoroughfare", "subThoroughfare", "region", "inlandWater", "ocean", and "areasOfInterest". "CLRegion" container user-created registers include: "center", "identifier", and "radius". "CLHeading" container user-created registers include: "headingAccuracy", "magneticHeading", "trueHeading", "y", and "z". Publicly available information indicates that user-created registers are used throughout the iOS operating system of the iOS-Compatible Devices. The user-created registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the user-created registers are more fully set forth in the

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
7	The apparatus of claim 1 or 2, wherein the plurality of registers includes a system-defined register, the system-defined register being set, controlled and used by the system, and being appendable to any container.	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes a system-defined register, the system-defined register being set, controlled and used by the system, and being appendable to any container. Most registers of the iOS operating system with default values qualify. Examples include the "proximity" register of the "Alarm" container, and the "desiredAccuracy" and "distancefilter" registers of the "CLLocationManager" container. Publicly available information indicates that system-defined registers are used throughout the iOS operating system of the iOS-Compatible Devices. The predefined registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the system-defined registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
8	The apparatus of claim 1 or 2, wherein the plurality of registers	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the plurality of registers includes at least one acquire register for controlling whether the container adds a register from other containers or adds a container from other containers when

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	includes at least one acquire register for controlling whether the container adds a register from other containers or adds a container from other containers when interacting with them.	interacting with them. For example, using an iOS Data Source, a container may add a register from other containers, or may add a container from other containers when interacting with them: Data Sources A data source is like a delegate except that, instead of being delegated control of the user interface, it is delegated control of data. A data source is an outlet held by `NSView` and `UIView` objects such as table views and outline views that require a source from which to populate their rows of visible data. The data source for a view is usually the same object that acts as its delegate, but it can be any object. As with the delegate, the data source must implement one or more methods of an informal protocol to supply the view with the data it needs and, in more advanced implementations, to handle data that users directly edit in such views. As with delegates, data sources are objects that must be present to receive messages from the objects requesting data. The application that uses them must ensure their persistence, retaining them if necessary in memory-managed code. Data sources are responsible for the persistence of the objects they hand out to user-interface objects. In other words, they are responsible for the memory management of those objects. However, whenever a view object such as an outline view or table view accesses the data from a data source, it retains the objects as long as it uses the data. But it does not use the data for very long. Typically it holds on to the data only long enough to display it. (EV0002209.) Publicly available information indicates that acquire registers are used throughout the iOS operating system of the iOS-Compatible Devices. The acquire registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the acquire registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		function, in substantially the same way, to reach substantially the same result.
9	The apparatus of claim 1 or 2, wherein the gateway includes means for acting upon another container, the means for acting upon another container using the plurality of registers to determine whether and how the container acts upon other containers.	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the gateway includes means for acting upon another container, the means for acting upon another container using the plurality of registers to determine whether and how the container acts upon other containers. For example, the gateways of iOS containers include means for acting upon Delegates, using the registers to determine whether and how the Delegate acts upon other iOS containers: **Delegates and Data Sources** A delegate is an object that acts on behalf of, or in coordination with, another object when that object encounters an event in a program. The delegating object is often a responder object—that is, an object inheriting from `NSResponder` in AppKit or `UIResponder` in UIKit—that is responding to a user event. The delegate is an object that is delegated control of the user interface for that event, or is at least asked to interpret the event in an application-specific manner. To better appreciate the value of delegation, it helps to consider an off-the-shelf Cocoa object such as a text field (an instance of `NSTextField` or `UITextField`) or a table view (an instance of `NSTableView` or `UITableView`). These objects are designed to fulfill a specific role in a generic fashion; a window object in the AppKit framework, for example, responds to mouse manipulations of its controls and handles such things as closing, resizing, and moving the physical window. This restricted and generic behavior necessarily limits what the object can know about how an event affects (or will affect) something elsewhere in the application, especially when the affected behavior is specific to your application. Delegation provides a way for your custom object to communicate application-specific behavior to the off-the-shelf object.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The programming mechanism of delegation gives objects a chance to coordinate their appearance and state with changes occurring elsewhere in a program, changes usually brought about by user actions. More importantly, delegation makes it possible for one object to alter the behavior of another object without the need to inherit from it. The delegate is almost always one of your custom objects, and by definition it incorporates application-specific logic that the generic and delegating object cannot possibly know itself. ## How Delegation Works The design of the delegation mechanism is simple (Figure 5-1). The delegating class has an outlet or property, usually one that is named `delegate`; if it is an outlet, it includes methods for setting and accessing the value of the outlet. It also declares, without implementing, one or more methods that constitute a formal protocol or an informal protocol. A formal protocol that uses optional methods—a feature of Objective-C 2.0—is the preferred approach, but both kinds of protocols are used by the Cocoa frameworks for delegation. In the informal protocol approach, the delegating class declares methods on a category of `NSObject`, and the delegate implements only those methods in which it has an interest in coordinating itself with the delegating object or affecting that object's default behavior. If the delegating class declares a formal protocol, the delegate may choose to implement those methods marked optional, but it must implement the required ones. Delegation follows a common design, illustrated by Figure 5-1. Figure 5-1 The mechanism of delegation 

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The methods of the protocol mark significant events handled or anticipated by the delegating object. This object wants either to communicate these events to the delegate or, for impending events, to request input or approval from the delegate. For example, when a user clicks the close button of a window in OS X, the window object sends the `windowShouldClose:` message to its delegate; this gives the delegate the opportunity to veto or defer the closing of the window if, for example, the window has associated data that must be saved (see Figure 5-2). Figure 5-2 A more realistic sequence involving a delegate  The delegating object sends a message only if the delegate implements the method. It makes this discovery by invoking the `NSObject` method `respondsToSelector:` in the delegate first.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The Form of Delegation Messages
		Delegation methods have a conventional form. They begin with the name of the AppKit or UIKit object doing the delegating—application, window, control, and so on; this name is in lower-case and without the "NS" or "UI" prefix. Usually (but not always) this object name is followed by an auxiliary verb indicative of the temporal status of the reported event. This verb, in other words, indicates whether the event is about to occur ("Should" or "Will") or whether it has just occurred ("Did" or "Has"). This temporal distinction helps to categorize those messages that expect a return value and those that don't. Listing 5-1 includes a few AppKit delegation methods that expect a return value.
		Listing 5-1 Sample delegation methods with return values
		`- (BOOL)application:(NSApplication *)sender`
		`  openFile:(NSString *)filename; // NSApplication`
		`- (BOOL)application:(UIApplication *)application`
		`  handleOpenURL:(NSURL *)url; // UIApplicationDelegate`
		`- (UITableRowIndexSet *)tableView:(NSTableView *)tableView`
		`  willSelectRows:(UITableRowIndexSet *)selection; // UITableViewDelegate`
		`- (NSRect)windowWillUseStandardFrame:(NSWindow *)window`
		`  defaultFrame:(NSRect)newFrame; // NSWindow`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The delegate that implements these methods can block the impending event (by returning `NO` in the first two methods) or alter a suggested value (the index set and the frame rectangle in the last two methods). It can even defer an impending event; for example, the delegate implementing the `applicationShouldTerminate:` method can delay application termination by returning `NSTerminateLater`.
		
		Other delegation methods are invoked by messages that don't expect a return value and so are typed to return `void`. These messages are purely informational, and the method names often contain "Did", "Will", or some other indication of a transpired or impending event. Listing 5-2 shows a few examples of these kinds of delegation method.
		
		Listing 5-2 Sample delegation methods returning `void`
		
		`- (void) tableView:(NSTableView*)tableView`
		`mouseDownInHeaderOfTableColumn:(NSTableColumn *)tableColumn; // NSTableView`
		`- (void)windowDidMove:(NSNotification *)notification; // NSWindow`
		`- (void)application:(UIApplication *)application`
		`willChangeStatusBarFrame:(CGRect)newStatusBarFrame; // UIApplication`
		`- (void)applicationWillBecomeActive:(NSNotification *)notification; // NSApplication`
		
		There are a couple of things to note about this last group of methods. The first is that an auxiliary verb of "Will" (as in the third method) does not necessarily mean that a return value is expected. In this case, the event is imminent and cannot be blocked, but the message gives the delegate an opportunity to prepare the program for the event.
		
		The other point of interest concerns the second and last method declarations in Listing 5-2 . The sole parameter of each of these methods is an `NSNotification` object, which means that these methods are invoked as the result of the posting of a particular notification. For example, the `windowDidMove:` method is associated with the `NSWindow` notification `NSWindowDidMoveNotification`. The section "Notifications" (page 227) discusses notifications in detail, but here it's important to understand the relationship of notifications to delegation messages in AppKit. The delegating object automatically makes its delegate an observer of all notifications it posts. All the delegate needs to do is implement the associated method to get the notification.
		
		To make an instance of your custom class the delegate of an AppKit object, simply connect the instance to the `delegate` outlet or property in Interface Builder. Or you can set it programmatically through the delegating object's `setDelegate:` method or `delegate` property, preferably early on, such as in the `awakeFromNib` or `applicationDidFinishLaunching:` method.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Delegation and the Cocoa Application Frameworks The delegating object in a Cocoa application is often a responder object such as a `UIApplication`, `NSWindow`, or `NSTableView` object. The delegate object itself is typically, but not necessarily, an object, often a custom object, that controls some part of the application (that is, a coordinating controller object). The following AppKit classes define a delegate: • `NSApplication` • `NSBrowser` • `NSControl` • `NSDrawer` • `NSFontManager` • `NSFontPanel` • `NSMatrix` • `NSOutlineView` • `NSSplitView` • `NSTableView` • `NSTabView` • `NSText` • `NSTextField` • `NSTextView` • `NSWindow` The UIKit framework also uses delegation extensively and always implements it using formal protocols. The application delegate is extremely important in an application running in iOS because it must respond to application-launch, application-quit, low-memory, and other messages from the application object. The application delegate must adopt the `UIApplicationDelegate` protocol. Delegating objects do not (and should not) retain their delegates. However, clients of delegating objects (applications, usually) are responsible for ensuring that their delegates are around to receive delegation messages. To do this, they may have to retain the delegate in memory-managed code. This precaution applies equally to data sources, notification observers, and targets of action messages. Note that in a garbage-collection environment, the reference to the delegate is strong because the retain-cycle problem does not apply.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Some AppKit classes have a more restricted type of delegate called a *modal delegate*. Objects of these classes (`NSOpenPanel`, for example) run modal dialogs that invoke a handler method in the designated delegate when the user clicks the dialog's OK button. Modal delegates are limited in scope to the operation of the modal dialog.
		
		The existence of delegates has other programmatic uses. For example, with delegates it is easy for two coordinating controllers in the same program to find and communicate with each other. For example, the object controlling the application overall can find the controller of the application's inspector window (assuming it's the current key window) using code similar to the following:
		
		`id winController = [[NSApp keyWindow] delegate];`
		
		And your code can find the application-controller object—by definition, the delegate of the global application instance—by doing something similar to the following:
		
		`id appController = [NSApp delegate];`
		
		(EV0002204-08.)
		
		Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

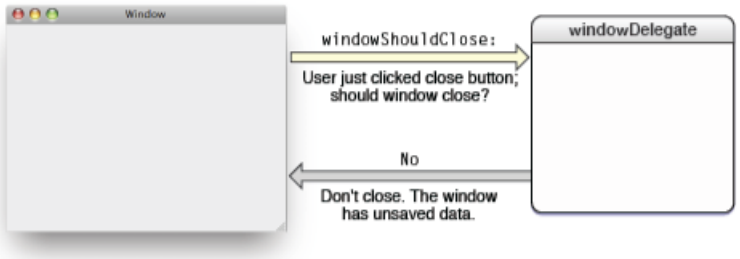
10	The apparatus of claim 1 or 2, wherein the gateway includes means	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the gateway includes means for allowing interaction, the means for allowing interaction using the plurality of registers to determine whether and how another container can act upon the

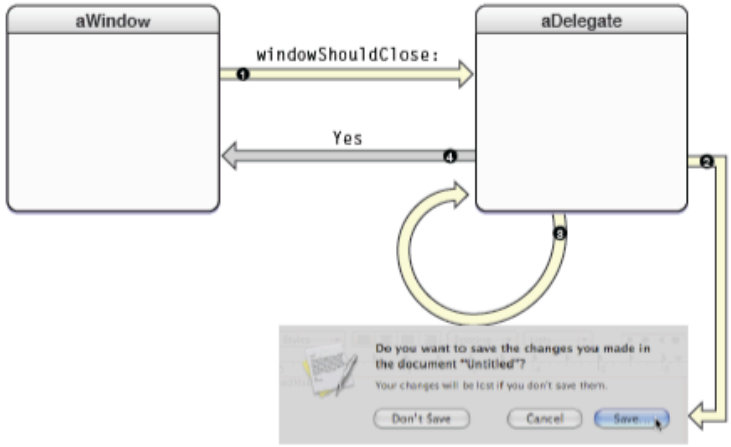
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	for allowing interaction, the means for allowing interaction using the plurality of registers to determine whether and how another container can act upon the container.	container. For example, the "CLLocationManager" container has a startMonitoringForRegion: gateway with means for allowing interaction with a "CLRegion container," using the plurality of registers: Overview The `CLRegion` class defines a geographical area that can be tracked. When an instance of this class is registered with a `CLLocationManager` object, the location manager generates an appropriate event whenever the user crosses the boundaries of the defined area. To use this class, create an instance of it and use the `startMonitoringForRegion:desiredAccuracy:` method of a `CLLocationManager` object to begin monitoring it. (EV0001686.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startMonitoringForRegion:
		
		Starts monitoring the specified region.
		
		`- (void)startMonitoringForRegion:(CLRegion *)region`
		
		Parameters
		`region`
		The region object that defines the boundary to monitor. This parameter must not be `nil`.
		
		Discussion
		You must call this method or the `startMonitoringForRegion:desiredAccuracy:` (page 34) method separately for each region you want to monitor. If an existing region with the same identifier is already being monitored by the application, the old region is replaced by the new one. The regions you add using this method are shared by all location manager objects in your application and stored in the `monitoredRegions` (page 16) property.
		
		Region events are delivered to the `locationManager:didEnterRegion:` and `locationManager:didExitRegion:` methods of your delegate. If there is an error, the location manager calls the `locationManager:monitoringDidFailForRegion:withError:` method of your delegate instead.
		
		An app can register up to 20 regions at a time. In order to report region changes in a timely manner, the region monitoring service requires network connectivity.
		
		In iOS 6, regions with a radius between 1 and 400 meters work better on iPhone 4S or later devices. (In iOS 5, regions with a radius between 1 and 150 meters work better on iPhone 4S and later devices.) On these devices, an app can expect to receive the appropriate region entered or region exited notification within 3 to 5 minutes on average, if not sooner.
		
		Availability
		Available in iOS 5.0 and later.
		
		See Also
		`- stopMonitoringForRegion:` (page 26)
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		(EV0001643-44.) In another example, iOS Delegates have gateway with means for allowing interaction using registers to determine whether and how other containers can act upon the Delegates: **Delegates and Data Sources** A delegate is an object that acts on behalf of, or in coordination with, another object when that object encounters an event in a program. The delegating object is often a responder object—that is, an object inheriting from `NSResponder` in AppKit or `UIResponder` in UIKit—that is responding to a user event. The delegate is an object that is delegated control of the user interface for that event, or is at least asked to interpret the event in an application-specific manner. To better appreciate the value of delegation, it helps to consider an off-the-shelf Cocoa object such as a text field (an instance of `NSTextField` or `UITextField`) or a table view (an instance of `NSTableView` or `UITableView`). These objects are designed to fulfill a specific role in a generic fashion; a window object in the AppKit framework, for example, responds to mouse manipulations of its controls and handles such things as closing, resizing, and moving the physical window. This restricted and generic behavior necessarily limits what the object can know about how an event affects (or will affect) something elsewhere in the application, especially when the affected behavior is specific to your application. Delegation provides a way for your custom object to communicate application-specific behavior to the off-the-shelf object.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The programming mechanism of delegation gives objects a chance to coordinate their appearance and state with changes occurring elsewhere in a program, changes usually brought about by user actions. More importantly, delegation makes it possible for one object to alter the behavior of another object without the need to inherit from it. The delegate is almost always one of your custom objects, and by definition it incorporates application-specific logic that the generic and delegating object cannot possibly know itself. ## How Delegation Works The design of the delegation mechanism is simple (Figure 5-1). The delegating class has an outlet or property, usually one that is named `delegate`; if it is an outlet, it includes methods for setting and accessing the value of the outlet. It also declares, without implementing, one or more methods that constitute a formal protocol or an informal protocol. A formal protocol that uses optional methods—a feature of Objective-C 2.0—is the preferred approach, but both kinds of protocols are used by the Cocoa frameworks for delegation. In the informal protocol approach, the delegating class declares methods on a category of `NSObject`, and the delegate implements only those methods in which it has an interest in coordinating itself with the delegating object or affecting that object's default behavior. If the delegating class declares a formal protocol, the delegate may choose to implement those methods marked optional, but it must implement the required ones. Delegation follows a common design, illustrated by Figure 5-1. Figure 5-1 The mechanism of delegation 

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The methods of the protocol mark significant events handled or anticipated by the delegating object. This object wants either to communicate these events to the delegate or, for impending events, to request input or approval from the delegate. For example, when a user clicks the close button of a window in OS X, the window object sends the `windowShouldClose:` message to its delegate; this gives the delegate the opportunity to veto or defer the closing of the window if, for example, the window has associated data that must be saved (see Figure 5-2). Figure 5-2 A more realistic sequence involving a delegate  The delegating object sends a message only if the delegate implements the method. It makes this discovery by invoking the `NSObject` method `respondsToSelector:` in the delegate first.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The Form of Delegation Messages
		Delegation methods have a conventional form. They begin with the name of the AppKit or UIKit object doing the delegating—application, window, control, and so on; this name is in lower-case and without the "NS" or "UI" prefix. Usually (but not always) this object name is followed by an auxiliary verb indicative of the temporal status of the reported event. This verb, in other words, indicates whether the event is about to occur ("Should" or "Will") or whether it has just occurred ("Did" or "Has"). This temporal distinction helps to categorize those messages that expect a return value and those that don't. Listing 5-1 includes a few AppKit delegation methods that expect a return value.
		
		Listing 5-1 Sample delegation methods with return values
		`- (BOOL)application:(NSApplication *)sender`
		`  openFile:(NSString *)filename;  // NSApplication`
		`- (BOOL)application:(UIApplication *)application`
		`  handleOpenURL:(NSURL *)url;  // UIApplicationDelegate`
		`- (UITableRowIndexSet *)tableView:(NSTableView *)tableView`
		`  willSelectRows:(UITableRowIndexSet *)selection;  // UITableViewDelegate`
		`- (NSRect)windowWillUseStandardFrame:(NSWindow *)window`
		`  defaultFrame:(NSRect)newFrame;  // NSWindow`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The delegate that implements these methods can block the impending event (by returning `NO` in the first two methods) or alter a suggested value (the index set and the frame rectangle in the last two methods). It can even defer an impending event; for example, the delegate implementing the `applicationShouldTerminate:` method can delay application termination by returning `NSTerminateLater`.
		
		Other delegation methods are invoked by messages that don't expect a return value and so are typed to return `void`. These messages are purely informational, and the method names often contain "Did", "Will", or some other indication of a transpired or impending event. Listing 5-2 shows a few examples of these kinds of delegation method.
		
		Listing 5-2 Sample delegation methods returning `void`
		
		`- (void) tableView:(NSTableView*)tableView`
		`mouseDownInHeaderOfTableColumn:(NSTableColumn *)tableColumn; // NSTableView`
		`- (void)windowDidMove:(NSNotification *)notification; // NSWindow`
		`- (void)application:(UIApplication *)application`
		`willChangeStatusBarFrame:(CGRect)newStatusBarFrame; // UIApplication`
		`- (void)applicationWillBecomeActive:(NSNotification *)notification; // NSApplication`
		
		There are a couple of things to note about this last group of methods. The first is that an auxiliary verb of "Will" (as in the third method) does not necessarily mean that a return value is expected. In this case, the event is imminent and cannot be blocked, but the message gives the delegate an opportunity to prepare the program for the event.
		
		The other point of interest concerns the second and last method declarations in Listing 5-2 . The sole parameter of each of these methods is an `NSNotification` object, which means that these methods are invoked as the result of the posting of a particular notification. For example, the `windowDidMove:` method is associated with the `NSWindow` notification `NSWindowDidMoveNotification`. The section "Notifications" (page 227) discusses notifications in detail, but here it's important to understand the relationship of notifications to delegation messages in AppKit. The delegating object automatically makes its delegate an observer of all notifications it posts. All the delegate needs to do is implement the associated method to get the notification.
		
		To make an instance of your custom class the delegate of an AppKit object, simply connect the instance to the `delegate` outlet or property in Interface Builder. Or you can set it programmatically through the delegating object's `setDelegate:` method or `delegate` property, preferably early on, such as in the `awakeFromNib` or `applicationDidFinishLaunching:` method.

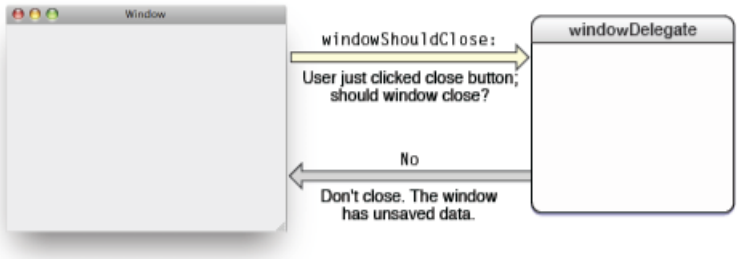
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Delegation and the Cocoa Application Frameworks The delegating object in a Cocoa application is often a responder object such as a `UIApplication`, `NSWindow`, or `NSTableView` object. The delegate object itself is typically, but not necessarily, an object, often a custom object, that controls some part of the application (that is, a coordinating controller object). The following AppKit classes define a delegate: • `NSApplication` • `NSBrowser` • `NSControl` • `NSDrawer` • `NSFontManager` • `NSFontPanel` • `NSMatrix` • `NSOutlineView` • `NSSplitView` • `NSTableView` • `NSTabView` • `NSText` • `NSTextField` • `NSTextView` • `NSWindow` The UIKit framework also uses delegation extensively and always implements it using formal protocols. The application delegate is extremely important in an application running in iOS because it must respond to application-launch, application-quit, low-memory, and other messages from the application object. The application delegate must adopt the `UIApplicationDelegate` protocol. Delegating objects do not (and should not) retain their delegates. However, clients of delegating objects (applications, usually) are responsible for ensuring that their delegates are around to receive delegation messages. To do this, they may have to retain the delegate in memory-managed code. This precaution applies equally to data sources, notification observers, and targets of action messages. Note that in a garbage-collection environment, the reference to the delegate is strong because the retain-cycle problem does not apply.

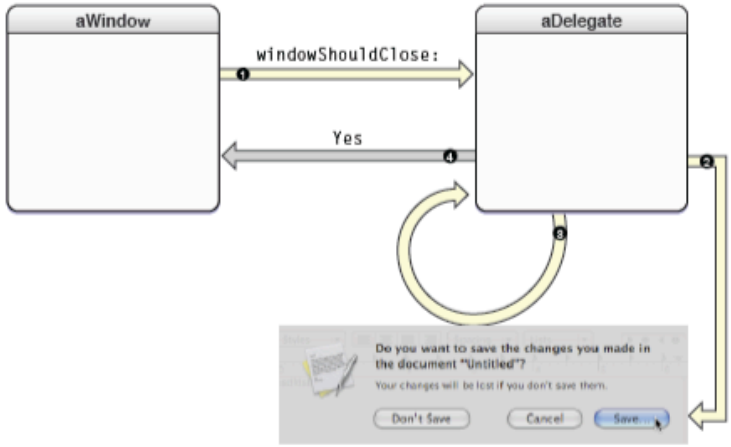
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Some AppKit classes have a more restricted type of delegate called a *modal delegate*. Objects of these classes (`NSOpenPanel`, for example) run modal dialogs that invoke a handler method in the designated delegate when the user clicks the dialog's OK button. Modal delegates are limited in scope to the operation of the modal dialog.
		
		The existence of delegates has other programmatic uses. For example, with delegates it is easy for two coordinating controllers in the same program to find and communicate with each other. For example, the object controlling the application overall can find the controller of the application's inspector window (assuming it's the current key window) using code similar to the following:
		
		`id winController = [[NSApp keyWindow] delegate];`
		
		And your code can find the application-controller object—by definition, the delegate of the global application instance—by doing something similar to the following:
		
		`id appController = [NSApp delegate];`
		
		(EV0002204-08.)
		
		Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

11	The apparatus of claim 1 or 2, wherein the gateway includes means	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the gateway includes means for gathering information, the means for gathering information recording register information from other containers, systems or processes that interact with

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	for gathering information, the means for gathering information recording register information from other containers, systems or processes that interact with the container.	the container. For example, iOS Delegates have gateways that gather information by recording register information from other containers, systems, or processes that interact with the Delegates: **Delegates and Data Sources** A delegate is an object that acts on behalf of, or in coordination with, another object when that object encounters an event in a program. The delegating object is often a responder object—that is, an object inheriting from `NSResponder` in AppKit or `UIResponder` in UIKit—that is responding to a user event. The delegate is an object that is delegated control of the user interface for that event, or is at least asked to interpret the event in an application-specific manner. To better appreciate the value of delegation, it helps to consider an off-the-shelf Cocoa object such as a text field (an instance of `NSTextField` or `UITextField`) or a table view (an instance of `NSTableView` or `UITableView`). These objects are designed to fulfill a specific role in a generic fashion; a window object in the AppKit framework, for example, responds to mouse manipulations of its controls and handles such things as closing, resizing, and moving the physical window. This restricted and generic behavior necessarily limits what the object can know about how an event affects (or will affect) something elsewhere in the application, especially when the affected behavior is specific to your application. Delegation provides a way for your custom object to communicate application-specific behavior to the off-the-shelf object.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The programming mechanism of delegation gives objects a chance to coordinate their appearance and state with changes occurring elsewhere in a program, changes usually brought about by user actions. More importantly, delegation makes it possible for one object to alter the behavior of another object without the need to inherit from it. The delegate is almost always one of your custom objects, and by definition it incorporates application-specific logic that the generic and delegating object cannot possibly know itself. ## How Delegation Works The design of the delegation mechanism is simple (Figure 5-1). The delegating class has an outlet or property, usually one that is named `delegate`; if it is an outlet, it includes methods for setting and accessing the value of the outlet. It also declares, without implementing, one or more methods that constitute a formal protocol or an informal protocol. A formal protocol that uses optional methods—a feature of Objective-C 2.0—is the preferred approach, but both kinds of protocols are used by the Cocoa frameworks for delegation. In the informal protocol approach, the delegating class declares methods on a category of `NSObject`, and the delegate implements only those methods in which it has an interest in coordinating itself with the delegating object or affecting that object's default behavior. If the delegating class declares a formal protocol, the delegate may choose to implement those methods marked optional, but it must implement the required ones. Delegation follows a common design, illustrated by Figure 5-1. Figure 5-1 The mechanism of delegation 

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The methods of the protocol mark significant events handled or anticipated by the delegating object. This object wants either to communicate these events to the delegate or, for impending events, to request input or approval from the delegate. For example, when a user clicks the close button of a window in OS X, the window object sends the `windowShouldClose:` message to its delegate; this gives the delegate the opportunity to veto or defer the closing of the window if, for example, the window has associated data that must be saved (see Figure 5-2). Figure 5-2 A more realistic sequence involving a delegate  The delegating object sends a message only if the delegate implements the method. It makes this discovery by invoking the `NSObject` method `respondsToSelector:` in the delegate first.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The Form of Delegation Messages
		Delegation methods have a conventional form. They begin with the name of the AppKit or UIKit object doing the delegating—application, window, control, and so on; this name is in lower-case and without the "NS" or "UI" prefix. Usually (but not always) this object name is followed by an auxiliary verb indicative of the temporal status of the reported event. This verb, in other words, indicates whether the event is about to occur ("Should" or "Will") or whether it has just occurred ("Did" or "Has"). This temporal distinction helps to categorize those messages that expect a return value and those that don't. Listing 5-1 includes a few AppKit delegation methods that expect a return value.
		Listing 5-1 Sample delegation methods with return values
		`- (BOOL)application:(NSApplication *)sender`
		`  openFile:(NSString *)filename;  // NSApplication`
		`- (BOOL)application:(UIApplication *)application`
		`  handleOpenURL:(NSURL *)url;  // UIApplicationDelegate`
		`- (UITableRowIndexSet *)tableView:(NSTableView *)tableView`
		`  willSelectRows:(UITableRowIndexSet *)selection;  // UITableViewDelegate`
		`- (NSRect)windowWillUseStandardFrame:(NSWindow *)window`
		`  defaultFrame:(NSRect)newFrame;  // NSWindow`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The delegate that implements these methods can block the impending event (by returning `NO` in the first two methods) or alter a suggested value (the index set and the frame rectangle in the last two methods). It can even defer an impending event; for example, the delegate implementing the `applicationShouldTerminate:` method can delay application termination by returning `NSTerminateLater`.
		
		Other delegation methods are invoked by messages that don't expect a return value and so are typed to return `void`. These messages are purely informational, and the method names often contain "Did", "Will", or some other indication of a transpired or impending event. Listing 5-2 shows a few examples of these kinds of delegation method.
		
		Listing 5-2 Sample delegation methods returning `void`
		
		`- (void) tableView:(NSTableView*)tableView`
		`mouseDownInHeaderOfTableColumn:(NSTableColumn *)tableColumn; // NSTableView`
		`- (void)windowDidMove:(NSNotification *)notification; // NSWindow`
		`- (void)application:(UIApplication *)application`
		`willChangeStatusBarFrame:(CGRect)newStatusBarFrame; // UIApplication`
		`- (void)applicationWillBecomeActive:(NSNotification *)notification; // NSApplication`
		
		There are a couple of things to note about this last group of methods. The first is that an auxiliary verb of "Will" (as in the third method) does not necessarily mean that a return value is expected. In this case, the event is imminent and cannot be blocked, but the message gives the delegate an opportunity to prepare the program for the event.
		
		The other point of interest concerns the second and last method declarations in Listing 5-2 . The sole parameter of each of these methods is an `NSNotification` object, which means that these methods are invoked as the result of the posting of a particular notification. For example, the `windowDidMove:` method is associated with the `NSWindow` notification `NSWindowDidMoveNotification`. The section "Notifications" (page 227) discusses notifications in detail, but here it's important to understand the relationship of notifications to delegation messages in AppKit. The delegating object automatically makes its delegate an observer of all notifications it posts. All the delegate needs to do is implement the associated method to get the notification.
		
		To make an instance of your custom class the delegate of an AppKit object, simply connect the instance to the `delegate` outlet or property in Interface Builder. Or you can set it programmatically through the delegating object's `setDelegate:` method or `delegate` property, preferably early on, such as in the `awakeFromNib` or `applicationDidFinishLaunching:` method.

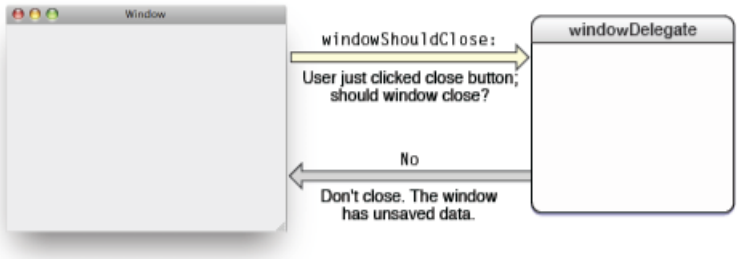
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Delegation and the Cocoa Application Frameworks The delegating object in a Cocoa application is often a responder object such as a `UIApplication`, `NSWindow`, or `NSTableView` object. The delegate object itself is typically, but not necessarily, an object, often a custom object, that controls some part of the application (that is, a coordinating controller object). The following AppKit classes define a delegate: • `NSApplication` • `NSBrowser` • `NSControl` • `NSDrawer` • `NSFontManager` • `NSFontPanel` • `NSMatrix` • `NSOutlineView` • `NSSplitView` • `NSTableView` • `NSTabView` • `NSText` • `NSTextField` • `NSTextView` • `NSWindow` The UIKit framework also uses delegation extensively and always implements it using formal protocols. The application delegate is extremely important in an application running in iOS because it must respond to application-launch, application-quit, low-memory, and other messages from the application object. The application delegate must adopt the `UIApplicationDelegate` protocol. Delegating objects do not (and should not) retain their delegates. However, clients of delegating objects (applications, usually) are responsible for ensuring that their delegates are around to receive delegation messages. To do this, they may have to retain the delegate in memory-managed code. This precaution applies equally to data sources, notification observers, and targets of action messages. Note that in a garbage-collection environment, the reference to the delegate is strong because the retain-cycle problem does not apply.

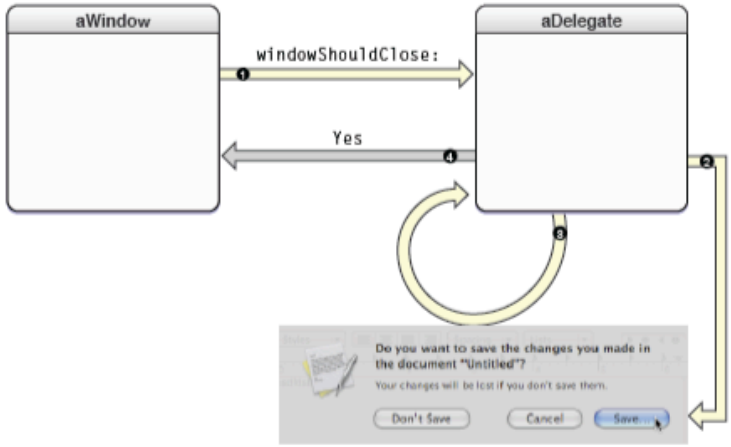
Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Some AppKit classes have a more restricted type of delegate called a *modal delegate*. Objects of these classes (`NSOpenPanel`, for example) run modal dialogs that invoke a handler method in the designated delegate when the user clicks the dialog's OK button. Modal delegates are limited in scope to the operation of the modal dialog.
		
		The existence of delegates has other programmatic uses. For example, with delegates it is easy for two coordinating controllers in the same program to find and communicate with each other. For example, the object controlling the application overall can find the controller of the application's inspector window (assuming it's the current key window) using code similar to the following:
		
		`id winController = [[NSApp keyWindow] delegate];`
		
		And your code can find the application-controller object—by definition, the delegate of the global application instance—by doing something similar to the following:
		
		`id appController = [NSApp delegate];`
		
		(EV0002204-08.)
		
		Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

12	The apparatus of claim 1 or 2, wherein the gateway includes means	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, with respect to both claims 1 and 2, the gateway includes means for reporting information, the means for reporting information by providing register information to other containers, systems or processes that interact with

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	for reporting information, the means for reporting information providing register information to other containers, systems or processes that interact with the container.	the container. For example, gateways of the iOS Delegates include means for reporting information by providing register information to other containers, systems or processes that interact with the Delegates: **Delegates and Data Sources** A delegate is an object that acts on behalf of, or in coordination with, another object when that object encounters an event in a program. The delegating object is often a responder object—that is, an object inheriting from `NSResponder` in AppKit or `UIResponder` in UIKit—that is responding to a user event. The delegate is an object that is delegated control of the user interface for that event, or is at least asked to interpret the event in an application-specific manner. To better appreciate the value of delegation, it helps to consider an off-the-shelf Cocoa object such as a text field (an instance of `NSTextField` or `UITextField`) or a table view (an instance of `NSTableView` or `UITableView`). These objects are designed to fulfill a specific role in a generic fashion; a window object in the AppKit framework, for example, responds to mouse manipulations of its controls and handles such things as closing, resizing, and moving the physical window. This restricted and generic behavior necessarily limits what the object can know about how an event affects (or will affect) something elsewhere in the application, especially when the affected behavior is specific to your application. Delegation provides a way for your custom object to communicate application-specific behavior to the off-the-shelf object.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The programming mechanism of delegation gives objects a chance to coordinate their appearance and state with changes occurring elsewhere in a program, changes usually brought about by user actions. More importantly, delegation makes it possible for one object to alter the behavior of another object without the need to inherit from it. The delegate is almost always one of your custom objects, and by definition it incorporates application-specific logic that the generic and delegating object cannot possibly know itself. ## How Delegation Works The design of the delegation mechanism is simple (Figure 5-1). The delegating class has an outlet or property, usually one that is named `delegate`; if it is an outlet, it includes methods for setting and accessing the value of the outlet. It also declares, without implementing, one or more methods that constitute a formal protocol or an informal protocol. A formal protocol that uses optional methods—a feature of Objective-C 2.0—is the preferred approach, but both kinds of protocols are used by the Cocoa frameworks for delegation. In the informal protocol approach, the delegating class declares methods on a category of `NSObject`, and the delegate implements only those methods in which it has an interest in coordinating itself with the delegating object or affecting that object's default behavior. If the delegating class declares a formal protocol, the delegate may choose to implement those methods marked optional, but it must implement the required ones. Delegation follows a common design, illustrated by Figure 5-1. Figure 5-1 The mechanism of delegation 

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		The methods of the protocol mark significant events handled or anticipated by the delegating object. This object wants either to communicate these events to the delegate or, for impending events, to request input or approval from the delegate. For example, when a user clicks the close button of a window in OS X, the window object sends the `windowShouldClose:` message to its delegate; this gives the delegate the opportunity to veto or defer the closing of the window if, for example, the window has associated data that must be saved (see Figure 5-2). Figure 5-2 A more realistic sequence involving a delegate  The delegating object sends a message only if the delegate implements the method. It makes this discovery by invoking the `NSObject` method `respondsToSelector:` in the delegate first.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The Form of Delegation Messages
		Delegation methods have a conventional form. They begin with the name of the AppKit or UIKit object doing the delegating—application, window, control, and so on; this name is in lower-case and without the "NS" or "UI" prefix. Usually (but not always) this object name is followed by an auxiliary verb indicative of the temporal status of the reported event. This verb, in other words, indicates whether the event is about to occur ("Should" or "Will") or whether it has just occurred ("Did" or "Has"). This temporal distinction helps to categorize those messages that expect a return value and those that don't. Listing 5-1 includes a few AppKit delegation methods that expect a return value.
		
		Listing 5-1 Sample delegation methods with return values
		`- (BOOL)application:(NSApplication *)sender`
		`  openFile:(NSString *)filename;  // NSApplication`
		`- (BOOL)application:(UIApplication *)application`
		`  handleOpenURL:(NSURL *)url;  // UIApplicationDelegate`
		`- (UITableRowIndexSet *)tableView:(NSTableView *)tableView`
		`  willSelectRows:(UITableRowIndexSet *)selection;  // UITableViewDelegate`
		`- (NSRect)windowWillUseStandardFrame:(NSWindow *)window`
		`  defaultFrame:(NSRect)newFrame;  // NSWindow`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		The delegate that implements these methods can block the impending event (by returning `NO` in the first two methods) or alter a suggested value (the index set and the frame rectangle in the last two methods). It can even defer an impending event; for example, the delegate implementing the `applicationShouldTerminate:` method can delay application termination by returning `NSTerminateLater`.
		
		Other delegation methods are invoked by messages that don't expect a return value and so are typed to return `void`. These messages are purely informational, and the method names often contain "Did", "Will", or some other indication of a transpired or impending event. Listing 5-2 shows a few examples of these kinds of delegation method.
		
		Listing 5-2 Sample delegation methods returning `void`
		
		`- (void) tableView:(NSTableView*)tableView`
		`mouseDownInHeaderOfTableColumn:(NSTableColumn *)tableColumn; // NSTableView`
		`- (void)windowDidMove:(NSNotification *)notification; // NSWindow`
		`- (void)application:(UIApplication *)application`
		`willChangeStatusBarFrame:(CGRect)newStatusBarFrame; // UIApplication`
		`- (void)applicationWillBecomeActive:(NSNotification *)notification; // NSApplication`
		
		There are a couple of things to note about this last group of methods. The first is that an auxiliary verb of "Will" (as in the third method) does not necessarily mean that a return value is expected. In this case, the event is imminent and cannot be blocked, but the message gives the delegate an opportunity to prepare the program for the event.
		
		The other point of interest concerns the second and last method declarations in Listing 5-2 . The sole parameter of each of these methods is an `NSNotification` object, which means that these methods are invoked as the result of the posting of a particular notification. For example, the `windowDidMove:` method is associated with the `NSWindow` notification `NSWindowDidMoveNotification`. The section "Notifications" (page 227) discusses notifications in detail, but here it's important to understand the relationship of notifications to delegation messages in AppKit. The delegating object automatically makes its delegate an observer of all notifications it posts. All the delegate needs to do is implement the associated method to get the notification.
		
		To make an instance of your custom class the delegate of an AppKit object, simply connect the instance to the `delegate` outlet or property in Interface Builder. Or you can set it programmatically through the delegating object's `setDelegate:` method or `delegate` property, preferably early on, such as in the `awakeFromNib` or `applicationDidFinishLaunching:` method.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Delegation and the Cocoa Application Frameworks The delegating object in a Cocoa application is often a responder object such as a `UIApplication`, `NSWindow`, or `NSTableView` object. The delegate object itself is typically, but not necessarily, an object, often a custom object, that controls some part of the application (that is, a coordinating controller object). The following AppKit classes define a delegate: • `NSApplication` • `NSBrowser` • `NSControl` • `NSDrawer` • `NSFontManager` • `NSFontPanel` • `NSMatrix` • `NSOutlineView` • `NSSplitView` • `NSTableView` • `NSTabView` • `NSText` • `NSTextField` • `NSTextView` • `NSWindow` The UIKit framework also uses delegation extensively and always implements it using formal protocols. The application delegate is extremely important in an application running in iOS because it must respond to application-launch, application-quit, low-memory, and other messages from the application object. The application delegate must adopt the `UIApplicationDelegate` protocol. Delegating objects do not (and should not) retain their delegates. However, clients of delegating objects (applications, usually) are responsible for ensuring that their delegates are around to receive delegation messages. To do this, they may have to retain the delegate in memory-managed code. This precaution applies equally to data sources, notification observers, and targets of action messages. Note that in a garbage-collection environment, the reference to the delegate is strong because the retain-cycle problem does not apply.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Some AppKit classes have a more restricted type of delegate called a *modal delegate*. Objects of these classes (`NSOpenPanel`, for example) run modal dialogs that invoke a handler method in the designated delegate when the user clicks the dialog's OK button. Modal delegates are limited in scope to the operation of the modal dialog.
		
		The existence of delegates has other programmatic uses. For example, with delegates it is easy for two coordinating controllers in the same program to find and communicate with each other. For example, the object controlling the application overall can find the controller of the application's inspector window (assuming it's the current key window) using code similar to the following:
		
		`id winController = [[NSApp keyWindow] delegate];`
		
		And your code can find the application-controller object—by definition, the delegate of the global application instance—by doing something similar to the following:
		
		`id appController = [NSApp delegate];`
		
		(EV0002204-08.)
		
		Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.

13	13. The apparat[u]s of claim 1 or 2, wherein the gateway includes an	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above.) In addition, on information and belief, the gateway includes an expert system including rules defining the interaction of the container with other containers, systems or processes.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	expert system including rules defining the interaction of the container with other containers, systems or processes.	In particular, on information and belief, Apple's SIRI has at least one gateway with at least one expert system with rules defining the interaction of the container with other containers, systems or processes in order to process and interpret spoken words of a user, to choose an appropriate response to those spoken words, and to cause the appropriate response to be communicated to the user. It is believed that gateways with expert systems are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
14	14. The apparatus of claim 1 or 2, wherein the information element is one from the group of text, graphic images, video, audio, a digital pattern, a process, a nested container, bit, natural number and a system.	Each iOS-Compatible Device infringes claim 1 and claim 2 (*see* above). In addition, with respect to both claims 1 and 2, the information element is one from the group of text, graphic images, video, audio, a digital pattern, a process, a nested container, bit, natural number and a system. For example, the containers below include information elements that are at least text (e.g., word(s)), digital patterns (e.g., unique identifiers; dates), bits (e.g., booleans), natural numbers (e.g., integers), and processes and systems (e.g., delegates):

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		## Properties
		
		### allDay
		
		A Boolean value that indicates whether the event is an all-day event.
		
		`@property(nonatomic, getter=isAllDay) BOOL allDay`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		### availability
		
		The availability setting for the event.
		
		`@property(nonatomic) EKEventAvailability availability`
		
		Discussion
		This setting is used by CalDAV and Exchange servers to indicate how the event should be treated for scheduling purposes.
		
		If the event's calendar does not support availability settings, this property's value is `EKEventAvailabilityNotSupported`.
		
		Availability
		Available in iOS 4.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		See Also
		`EKEventAvailability`
		Declared In
		`EKEvent.h`
		
		## birthdayPersonID
		
		The Address Book framework record identifier of the person for this birthday event. (read-only)
		
		`@property(nonatomic, readonly) NSInteger birthdayPersonID`
		
		Discussion
		This property is only set if this is a birthday event; otherwise the property is `nil`.
		
		Special Considerations
		
		Note: This property is equivalent to the `birthdayPersonUniqueID` property on OS X.
		
		Availability
		Available in iOS 5.0 and later.
		Declared In
		`EKEvent.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		endDate
		
		The end date for the event.
		
		`@property(nonatomic, copy) NSDate *endDate`
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		eventIdentifier
		
		A unique identifier for the event. (read-only)
		
		`@property(nonatomic, readonly) NSString *eventIdentifier`
		
		Discussion
		You can use this identifier to look up an event with the EKEventStore method `eventWithIdentifier:`.
		
		If the calendar of an event changes, its identifier most likely changes as well.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		isDetached
		
		A Boolean value that indicates whether an event is a detached instance of a repeating event. (read-only)
		
		`@property(nonatomic, readonly) BOOL isDetached`
		
		Discussion
		This value is `YES` if and only if the event is part of a repeating event and one or more of its attributes have been modified from the repeating event's default attributes.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`
		
		organizer
		
		The organizer associated with the event. (read-only)
		
		`@property(nonatomic, readonly) EKParticipant *organizer`
		
		Discussion
		This property is `nil` if the event has no organizer.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKEvent.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDate
		The start date of the event.
		`@property(nonatomic, copy) NSDate *startDate`
		Discussion
		Floating events such as all-day events are returned in the default time zone.
		Availability
		Available in iOS 4.0 and later.
		Declared In
		EKEvent.h
		
		status
		The status of the event. (read-only)
		`@property(nonatomic, readonly) EKEventStatus status`
		Discussion
		You should act based on an event's status only if the status is EKEventStatusCanceled, which indicates that the event has been canceled. Other statuses should be considered informational.
		Availability
		Available in iOS 4.0 and later.
		See Also
		EKEventStatus
		Declared In
		EKEvent.h
		
		(EV0001424-26.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		completed
		
		A Boolean value determining whether or not the reminder is marked completed.
		
		`@property(nonatomic, getter=isCompleted) BOOL completed`
		
		Discussion
		Setting this property to `YES` will set `completionDate` to the current date; setting this property to `NO` will set `completionDate` to `nil`.
		
		Special Considerations
		If the reminder was completed using a different client, you may encounter the case where this property is `YES`, but `completionDate` is `nil`.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		completionDate
		
		The date on which the reminder was completed.
		
		`@property(nonatomic, copy) NSDate *completionDate`
		
		Discussion
		Setting this property to a date will set `completed` to `YES`; setting this property to `nil` will set `completed` to `NO`.
		
		Availability
		Available in iOS 6.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Declared In
		`EKReminder.h`
		
		## dueDateComponents
		
		The date by which the reminder should be completed.
		
		`@property(nonatomic, copy) NSDateComponents *dueDateComponents`
		
		Discussion
		The use of date components allows the due date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		This components's `timeZone` property is independent of time zone properties on `startDateComponents` and its super `EKCalendarItem` object. By default, the due date is set to the system time zone.
		
		Special Considerations
		On iOS, Event Kit requires that a start date is set if the due date is set, however this is not a requirement on OS X.
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		startDateComponents
		
		The start date of the task.
		
		`@property(nonatomic, copy) NSDateComponents *startDateComponents`
		
		Discussion
		The use of date components allows the start date and its time zone to be represented in a single property. A `nil` time zone represents a floating date. Setting a date component without an hour, minute and second component will set the reminder to be an all-day reminder. If this property is set, the calendar must be set to `NSGregorianCalendar`; otherwise an exception is raised.
		
		The start date components's `timeZone` property corresponds to the `timeZone` property on `EKCalendarItem`. A change in one value will cause a change in the other. Setting the time zone directly on the components does not guarantee that your changes will be saved; instead, pull this property from the reminder, set the time zone on it, and assign it back to the reminder:
		
		`NSDateComponents *start = myEKReminder.startDateComponents;`
		`start.timeZone = myNSTimeZone;`
		`myEKReminder.startDateComponents = start;`
		
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKReminder.h`
		
		(EV0001431-33.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		## Properties
		
		### absoluteDate
		
		The absolute date for the alarm.
		
		`@property(copy) NSDate *absoluteDate`
		
		Discussion
		If you set this property for a relative offset alarm, it loses the relative offset and becomes an absolute alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKAlarm.h`
		
		### proximity
		
		A value indicating how a location-based alarm is triggered.
		
		`@property EKAlarmProximity *proximity`
		
		Discussion
		Alarms can be set to trigger when entering or exiting a location specified by `structuredLocation`. By default, alarms are not affected by location.
		
		Availability
		Available in iOS 6.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		See Also
		`EKAlarmProximity`
		Declared In
		`EKAlarm.h`
		
		## relativeOffset
		
		The offset from the start of an event, at which the alarm fires.
		
		`@property NSTimeInterval relativeOffset`
		
		Discussion
		If you set this value for an absolute alarm, it loses its absolute date and becomes a relative offset alarm.
		
		Availability
		Available in iOS 4.0 and later.
		Declared In
		`EKAlarm.h`
		
		## structuredLocation
		
		The location to trigger an alarm.
		
		`@property EKStructuredLocation *structuredLocation`
		
		Discussion
		This property is used in conjunction with `proximity` to perform geofence-based triggering of reminders.
		Availability
		Available in iOS 6.0 and later.
		Declared In
		`EKAlarm.h`
		
		(EV0001418-20.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

`coordinate` (page 5) *property*

The geographical coordinate information. (read-only)

`altitude` (page 5) *property*

The altitude measured in meters. (read-only)

`horizontalAccuracy` (page 6) *property*

The radius of uncertainty for the location, measured in meters. (read-only)

`verticalAccuracy` (page 8) *property*

The accuracy of the altitude value in meters. (read-only)

`timestamp` (page 7) *property*

The time at which this location was determined. (read-only)

– `description` (page 9)

Returns the location data in a formatted text string.

(EV0001513.)

activityType

The type of user activity associated with the location updates.

```
@property(assign, nonatomic) CLActivityType activityType
```

Discussion

The location manager uses the information in this property as a cue to determine when location updates may be automatically paused. Pausing updates gives the system the opportunity to save power in situations where the user's location is not likely to be changing. For example, if the activity type is `CLActivityTypeAutomotiveNavigation` (page 29) and no location changes have occurred recently, the radios might be powered down until movement is detected again.

Availability

Available in iOS 6.0 and later.

Declared in

`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		delegate
		
		The delegate object to receive update events.
		
		`@property(assign, nonatomic) id<CLLocationManagerDelegate> delegate`
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		Teslameter
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		desiredAccuracy
		
		The accuracy of the location data.
		
		`@property(assign, nonatomic) CLLocationAccuracy desiredAccuracy`
		
		Discussion
		The receiver does its best to achieve the requested accuracy; however, the actual accuracy is not guaranteed.
		
		You should assign a value to this property that is appropriate for your usage scenario. In other words, if you need the current location only within a few kilometers, you should not specify `kCLLocationAccuracyBest` for the accuracy. Determining a location with greater accuracy requires more time and more power.
		
		When requesting high-accuracy location data, the initial event delivered by the location service may not have the accuracy you requested. The location service delivers the initial event as quickly as possible. It then continues to determine the location with the accuracy you requested and delivers additional events, as necessary, when that data is available.
		
		The default value of this property is `kCLLocationAccuracyBest`.
		
		This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		
		Declared in
		`CLLocationManager.h`
		
		distanceFilter
		
		The minimum distance (measured in meters) a device must move horizontally before an update event is generated.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		`@property(assign, nonatomic) CLLocationDistance distanceFilter`
		
		Discussion
		This distance is measured relative to the previously delivered location. Use the value `kCLDistanceFilterNone` to be notified of all movements. The default value of this property is `kCLDistanceFilterNone`.
		
		This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.
		
		Special Considerations
		In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.
		
		Availability
		Available in iOS 2.0 and later.
		
		Related Sample Code
		LocateMe
		Regions
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		heading
		
		The most recently reported heading. (read-only)
		
		`@property(readonly, nonatomic) CLHeading *heading`
		
		Discussion
		The value of this property is `nil` if heading updates have never been initiated.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`
		
		headingFilter
		
		The minimum angular change (measured in degrees) required to generate new heading events.
		`@property(assign, nonatomic) CLLocationDegrees headingFilter`
		
		Discussion
		The angular distance is measured relative to the last delivered heading event. Use the value `kCLHeadingFilterNone` to be notified of all movements. The default value of this property is `1` degree.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		headingOrientation
		
		The device orientation to use when computing heading values.
		
		`@property(assign, nonatomic) CLDeviceOrientation headingOrientation`
		
		Discussion
		When computing heading values, the location manager assumes that the top of the device in portrait mode represents due north (0 degrees) by default. For applications that run in other orientations, this may not always be the most convenient orientation. This property allows you to specify which device orientation you want the location manager to use as the reference point for due north.
		
		Although you can set the value of this property to `CLDeviceOrientationUnknown`, `CLDeviceOrientationFaceUp`, or `CLDeviceOrientationFaceDown`, doing so has no effect on the orientation reference point. The original reference point is retained instead.
		
		Changing the value in this property affects only those heading values reported after the change is made.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`
		
		location
		
		The most recently retrieved user location. (read-only)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		`@property(readonly, nonatomic) CLLocation *location`
		
		Discussion
		The value of this property is `nil` if no location data has ever been retrieved.
		
		In iOS 4.0 and later, this property may contain a more recent location object at launch time. Specifically, if significant location updates are running and your application is terminated, this property is updated with the most recent location data when your application is relaunched (and you create a new location manager object). This location data may be more recent than the last location event processed by your application.
		
		It is always a good idea to check the timestamp of the location stored in this property. If the receiver is currently gathering location data, but the minimum distance filter is large, the returned location might be relatively old. If it is, you can stop the receiver and start it again to force an update.
		
		Availability
		Available in iOS 2.0 and later.
		
		See Also
		– `startUpdatingLocation` (page 25)
		
		Related Sample Code
		AVMovieExporter
		Locations
		PhotoLocations
		TaggedLocations
		
		Declared in
		`CLLocationManager.h`
		maximumRegionMonitoringDistance
		The largest boundary distance that can be assigned to a region. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDistance maximumRegionMonitoringDistance`
		
		Discussion
		This property defines the largest boundary distance allowed from a region's center point. Attempting to monitor a region with a distance larger than this value causes the location manager to send a `kCLErrorRegionMonitoringFailure` error to the delegate.
		
		If region monitoring is unavailable or not supported, the value in this property is `-1`.
		
		Availability
		Available in iOS 4.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Declared in
		`CLLocationManager.h`
		
		monitoredRegions
		
		The set of shared regions monitored by all location manager objects. (read-only)
		
		`@property(readonly, nonatomic) NSSet *monitoredRegions`
		
		Discussion
		You cannot add regions to this property directly. Instead, you must register regions by calling the `startMonitoringForRegion:desiredAccuracy:` (page 34) method. The regions in this property are shared by all instances of the `CLLocationManager` class in your application.
		
		The objects in this set may not necessarily be the same objects you specified at registration time. Only the region data itself is maintained by the system. Therefore, the only way to uniquely identify a registered region is using its `identifier` property.
		
		The location manager persists region data between launches of your application. If your application is terminated and then relaunched, the contents of this property are repopulated with region objects that contain the previously registered data.
		
		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLLocationManager.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		pausesLocationUpdatesAutomatically
		
		A Boolean value indicating whether the location manager object may pause location updates.
		
		`@property(assign, nonatomic) BOOL pausesLocationUpdatesAutomatically`
		
		Discussion
		Allowing the location manager to pause updates can improve battery life on the target device without sacrificing location data. When this property is set to `YES`, the location manager pauses updates (and powers down the appropriate hardware) at times when the location data is unlikely to change. For example, if the user stops for food while using a navigation app, the location manager might pause updates for a period of time. You can help the determination of when to pause location updates by assigning a value to the `activityType` property.
		
		The default value of this property is `YES`.
		
		Availability
		Available in iOS 6.0 and later.
		
		See Also
		`@property activityType` (page 11)
		
		Declared in
		`CLLocationManager.h`
		
		(EV0001632-38.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		`name` (page 8) *property* The name of the placemark. (read-only) `addressDictionary` (page 5) *property* A dictionary containing the Address Book keys and values for the placemark. (read-only) `ISOcountryCode` (page 7) *property* The abbreviated country name. (read-only) `country` (page 6) *property* The name of the country associated with the placemark. (read-only) `postalCode` (page 9) *property* The postal code associated with the placemark. (read-only) `administrativeArea` (page 6) *property* The state or province associated with the placemark. (read-only) `subAdministrativeArea` (page 10) *property* Additional administrative area information for the placemark. (read-only) `locality` (page 7) *property* The city associated with the placemark. (read-only) `subLocality` (page 10) *property* Additional city-level information for the placemark. (read-only) `thoroughfare` (page 11) *property* The street address associated with the placemark. (read-only) `subThoroughfare` (page 10) *property* Additional street-level information for the placemark. (read-only) `region` (page 9) *property* The geographic region associated with the placemark. (read-only) `inlandWater` (page 7) *property* The name of the inland water body associated with the placemark. (read-only) `ocean` (page 9) *property* The name of the ocean associated with the placemark. (read-only) `areasOfInterest` (page 6) *property* The relevant areas of interest associated with the placemark. (read-only) (EV0001662-63.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		center
		
		The center point of the region. (read-only)
		
		`@property(readonly, nonatomic) CLLocationCoordinate2D center`
		
		Availability
		Available in iOS 4.0 and later.
		
		Related Sample Code
		Regions
		
		Declared in
		`CLRegion.h`
		
		identifier
		
		The identifier for the region object. (read-only)
		
		`@property(readonly, nonatomic) NSString *identifier`
		
		Discussion
		This is a value that you specify and can use to identify this region inside your application.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Availability
		Available in iOS 4.0 and later.
		
		Declared in
		`CLRegion.h`
		
		radius
		The radius (measured in meters) that defines the region's outer boundary. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDistance radius`
		
		Availability
		Available in iOS 4.0 and later.
		
		Related Sample Code
		Regions
		
		Declared in
		`CLRegion.h`
		
		(EV0001688.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		headingAccuracy
		
		The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDirection headingAccuracy`
		
		Discussion
		A positive value in this property represents the potential error between the value reported by the `magneticHeading` (page 5) property and the actual direction of magnetic north. Thus, the lower the value of this property, the more accurate the heading. A negative value means that the reported heading is invalid, which can occur when the device is uncalibrated or there is strong interference from local magnetic fields.
		
		Availability
		Available in iOS 3.0 and later.
		
		Declared in
		`CLHeading.h`

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		magneticHeading *The heading (measured in degrees) relative to magnetic north. (read-only)* `@property(readonly, nonatomic) CLLocationDirection magneticHeading` **Discussion** The value in this property represents the heading relative to the magnetic North Pole, which is different from the geographic North Pole. The value `0` means the device is pointed toward magnetic north, `90` means it is pointed east, `180` means it is pointed south, and so on. The value in this property should always be valid. In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` property in *CLLocationManager Class Reference*. If the `headingAccuracy` property contains a negative value, the value in this property should be considered unreliable. **Availability** Available in iOS 3.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		See Also
		`@property headingAccuracy` (page 5)
		`@property trueHeading` (page 6)
		
		Declared in
		`CLHeading.h`
		
		timestamp
		
		The time at which this heading was determined. (read-only)
		
		`@property(readonly, nonatomic) NSDate *timestamp`
		
		Availability
		Available in iOS 3.0 and later.
		
		Declared in
		`CLHeading.h`
		
		trueHeading
		
		The heading (measured in degrees) relative to true north. (read-only)
		
		`@property(readonly, nonatomic) CLLocationDirection trueHeading`
		
		Discussion
		The value in this property represents the heading relative to the geographic North Pole. The value `0` means the device is pointed toward true north, `90` means it is pointed due east, `180` means it is pointed due south, and so on. A negative value indicates that the heading could not be determined.
		
		In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` property in *CLLocationManager Class Reference*.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Important: This property contains a valid value only if location updates are also enabled for the corresponding location manager object. Because the position of true north is different from the position of magnetic north on the Earth's surface, Core Location needs the current location of the device to compute the value of this property.
		
		Availability
		Available in iOS 3.0 and later.
		
		See Also
		`@property magneticHeading` (page 5)
		
		Declared in
		`CLHeading.h`
		x
		The geomagnetic data (measured in microteslas) for the x-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue x`
		
		Discussion
		This value represents the x-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		y
		The geomagnetic data (measured in microteslas) for the y-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue y`
		
		Discussion
		This value represents the y-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices

		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		
		z
		
		The geomagnetic data (measured in microteslas) for the z-axis. (read-only)
		
		`@property(readonly, nonatomic) CLHeadingComponentValue z`
		
		Discussion
		This value represents the z-axis deviation from the magnetic field lines being tracked by the device.
		
		Availability
		Available in iOS 3.0 and later.
		
		Related Sample Code
		Teslameter
		
		Declared in
		`CLHeading.h`
		
		Publicly available information indicates that information elements are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation.
		
		It is believed that the structure and operation of the information elements are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code.
		
		Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15A	An apparatus for transmitting, receiving and manipulating information on a computer system, the apparatus including	Each iOS-Compatible Device is an apparatus for transmitting, receiving and manipulating information on a computer system. (*See* the claim elements below.) Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15B	a plurality of containers, each container being a logically defined data enclosure and comprising:	Each iOS-Compatible Device includes a plurality of containers, each container being a logically defined data enclosure. (*See* the discussion presented for claim element 1B, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that containers are used throughout the iOS operating system of the iOS-Compatible Devices. The containers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the containers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15C	an information element having information;	Each iOS-Compatible Device comprises an information element having information. (*See* the discussion presented for claim element 1C, which is incorporated by reference as if fully set forth herein.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Publicly available information indicates that information elements are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the information elements are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15D	a plurality of registers, the plurality of registers forming part of the container and including	Each container of each iOS-Compatible Device comprises a plurality of registers, the plurality of registers forming part of the container. (*See* the discussion presented for claim element 1D, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that registers are used throughout the iOS operating system of the iOS-Compatible Devices. The registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15E	a first register for storing a unique container identification	The plurality of registers includes a first register for storing a unique container identification value. (*See* the discussion presented for claim element 1E, which is incorporated by reference as if fully set forth

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	value,	herein.) Publicly available information indicates that first registers are used throughout the iOS operating system of the iOS-Compatible Devices. The first registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the first registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15F	a second register having a representation designating time and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus event time, and	The plurality of registers includes a second register having a representation designating time and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus event time. (*See* the discussion presented for claim element 1F, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that second registers are used throughout the iOS operating system of the iOS-Compatible Devices. The second registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the second registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15G	at least one	The plurality of registers includes at least one acquire register for controlling whether the container adds

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	acquire register for controlling whether the container adds a register from other containers or adds a container from other containers when interacting with them; and	a register from other containers or adds a container from other containers when interacting with them. (*See* the discussion presented for claim element 8, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that acquire registers are used throughout the iOS operating system of the iOS-Compatible Devices. The acquire registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the acquire registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
15H	a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes.	Each container of each iOS-Compatible Device comprises a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes. (*See* the discussion presented for claim element 1J, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16A	16. An apparatus for transmitting, receiving and manipulating information on a computer system, the apparatus including	Each iOS-Compatible Device is an apparatus for transmitting, receiving and manipulating information on a computer system. (*See* the claim elements below.) Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16B	a plurality of containers, each container being a logically defined data enclosure and comprising:	Each iOS-Compatible Device includes a plurality of containers, each container being a logically defined data enclosure. (*See* the discussion presented for claim element 1B, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that containers are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the containers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16C	an information element having information;	Each container of each iOS-Compatible Device comprises an information element having information. (*See* the discussion presented for claim element 1C, which is incorporated by reference as if fully set forth herein.)

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		Publicly available information indicates that information elements are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the information elements are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16D	a plurality of registers, the plurality of registers forming part of the container and including	Each container of each iOS-Compatible Device comprises a plurality of registers, the plurality of registers forming part of the container. (*See* the discussion presented for claim element 1D, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that registers are used throughout the iOS operating system of the iOS-Compatible Devices. The information elements corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16E	a first register for storing a unique container identification	The plurality of registers includes a first register for storing a unique container identification value. (*See* the discussion presented for claim element 1E, which is incorporated by reference as if fully set forth

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	value,	herein.) Publicly available information indicates that first registers are used throughout the iOS operating system of the iOS-Compatible Devices. The first registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the first registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16F	a second register having a representation designating space and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus three dimensional space time, and	The plurality of registers includes a second register having a representation designating space and governing interactions of the container with other containers, systems or processes according to utility of information in the information element relative to an external-to-the-apparatus three-dimensional space. (*See* the discussion presented for claim element 2F, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that second registers are used throughout the iOS operating system of the iOS-Compatible Devices. The second registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the second registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16G	at least one	The plurality of registers includes at least one acquire register for controlling whether the container adds

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
	acquire register for controlling whether the container adds a register from other containers or adds a container from other containers when interacting with them; and	a register from other containers or adds a container from other containers when interacting with them. (*See* the discussion presented for claim element 8, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that acquire registers are used throughout the iOS operating system of the iOS-Compatible Devices. The acquire registers corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the acquire registers are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.
16H	a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes.	Each container of each iOS-Compatible Device comprises a gateway attached to and forming part of the container, the gateway controlling the interaction of the container with other containers, systems or processes. (*See* the discussion presented for claim element 1J, which is incorporated by reference as if fully set forth herein.) Publicly available information indicates that gateways are used throughout the iOS operating system of the iOS-Compatible Devices. The gateways corresponding to the above evidence are only examples. Other examples are described in the Apple iOS Developer Library documentation. It is believed that the structure and operation of the gateways are more fully set forth in the source code of the iOS operating system, which is not publicly available. Evolutionary Intelligence reserves its right to supplement and/or modify this claim chart after obtaining discovery of this source code. Evolutionary Intelligence presently contends that this element is literally present in the accused instrumentality. Evolutionary Intelligence reserves its right to contend that this element is satisfied under the doctrine of equivalents because any differences between this claim element and any accused

Element	U.S. Patent No. 7,010,536	iOS-Compatible Devices
		instrumentality are insubstantial and the accused instrumentality preforms substantially the same function, in substantially the same way, to reach substantially the same result.